UNIVERSIDAD DE MURCIA

Proyecto Docente

Concurso Oposición al Cuerpo de Profesores Titulares de Escuelas Universitarias

Área de Lenguajes y Sistemas Informáticos
Departamento de Informática y Sistemas

Docencia en
"Ampliación de Algoritmos y Estructuras de Datos"

Documentación presentada por el candidato

Ginés García Mateos

Murcia, marzo de 2004

Datos de la Plaza objeto de Concurso

Plaza nº:	292/2001
Fecha de la convocatoria:	29 de noviembre de 2001
Fecha de publicación en B.O.E.:	13 de diciembre de 2001
Cuerpo docente:	Profesores Titulares de Escuelas Universitarias
Área de conocimiento:	Lenguajes y Sistemas Informáticos
Departamento:	Informática y Sistemas
Centro:	Facultad de Informática Universidad de Murcia
Actividad a realizar:	Docencia en "Ampliación de Algoritmos y Estructuras de Datos"

Merci pour le bon séjour passé avec nous;
aurevoir, bon voyage, que Dieu vous protège.
Je penserai toujour à vous.

La Cherie, Dschang, 9-aout-2001

Agradecimientos

Ahora que aspiro a convertirme en profesor, no puedo dejar de acordarme de aquellos que han sido mis profesores y maestros a lo largo de mi formación, empezando por mis padres, auténticos maestros en el arte de la vida.

Gracias a todos los profesores y compañeros con los que he tenido la oportunidad de compartir labores docentes durante estos seis años: **Jesús**, **Norberto**, **Domingo** y **Quino**, por sus consejos, por sus críticas y por su ayuda.

Gracias también a los restantes compañeros del Departamento de Informática y Sistemas –especialmente a **José Luis**, **Begoña** y **Marcos**–, a los colegas del Departamento de Ingeniería y Tecnología de Computadores –del que formé parte siquiera por unas semanas–, y a los demás compañeros y habitantes habituales de esta Facultad.

Y, finalmente, gracias a ti, anónimo lector, por considerar que este humilde documento merece el tiempo que le dedicas.

G.G.M.

Presentación

'¿Podría usted decirme, por favor, qué camino debo tomar ahora?'
'Eso depende en gran medida de dónde quieras llegar', dijo el Gato.
'No me importa dónde llegar', dijo Alicia.
'Entonces da igual el camino que tomes'.

Lewis Carroll, Alicia en el país de las maravillas

"La razón por la que las estructuras de datos y los algoritmos
pueden trabajar conjuntamente y sin fisuras es...
que ninguno sabe nada del otro."

Alex Stepanov

Los artículos 35 a 39 de la Ley Orgánica 11/1983 de Reforma Universitaria (LRU) establecen los procedimientos de acceso a los cuerpos de funcionarios docentes universitarios, indicando que las Universidades deciden la categoría y denominación de las plazas que se convocan, el tipo de concurso y el lugar de la convocatoria. Esta Ley es desarrollada en los Reales Decretos 1888/1984 de 26 de septiembre (BOE de 26 de octubre) y su posterior modificación, en los Reales Decretos 1427/1986 de 13 de junio, y RD 1788/1997 de 1 de diciembre de 1997, que regulan los concursos para la provisión de plazas de los cuerpos docentes universitarios. El apartado 1 del artículo 9 de este último Real Decreto establece que los concursantes deben entregar en el acto de presentación al Presidente de la Comisión la siguiente documentación:

- Currículum vitae, por quintuplicado, según el modelo que determine la convocatoria del concurso y un ejemplar de las publicaciones y documentos acreditativos de lo consignado en el mismo.
- Proyecto docente, por quintuplicado, que el aspirante propone desarrollar en el caso de serle adjudicada la plaza a concurso.

Los candidatos deberán exponer y defender, mediante una prueba oral ante el tribunal, los méritos y la propuesta presentada para la asignatura objeto de la convocatoria de la plaza.

En este documento se presenta el **proyecto docente** propuesto por el candidato para concursar en la plaza número 292/2001, de Profesor Titular de Escuelas Universitarias, convocada por Resolución de la Universidad de Murcia el 29 de noviembre de 2001, y publicada en el BOE de 13 de diciembre de 2001. De acuerdo con la convocatoria, la labor a desempeñar en esta plaza es la docencia en "Ampliación de Algoritmos y Estructuras de Datos", en el área de conocimiento de Lenguajes y Sistemas Informáticos del Departamento de Informática y Sistemas de la Universidad de Murcia.

Nota sobre la reforma de planes de estudios

La asignatura "Ampliación de Algoritmos y Estructura de Datos", a la que se refiere la plaza 292/2001, está incluida en el segundo curso de la titulación Ingeniero en Informática del plan de estudios de 1996 de la FIUM. Tras la reforma de este plan de estudios, la asignatura citada se impartió por última vez durante el curso 2002/03. La implantación de los nuevos planes ha sido progresiva y desde el presente curso, 2003/04, se viene impartiendo el plan de 2002 en el segundo curso de II.

En los nuevos planes la antigua asignatura "Ampliación de Algoritmos y Estructura de Datos" (semestral, con 6 créditos de teoría y 2 de prácticas) se combina con la asignatura "Laboratorio de Programación" (trimestral, con 0 créditos de teoría y 6 de prácticas) para dar lugar a la asignatura "Algoritmos y Estructuras de Datos" (anual, con 6 créditos de teoría y 6 de prácticas). Además, los planes nuevos dan uniformidad a la enseñanza de la materia en las tres titulaciones de informática, la Ingeniería Superior y las Ingenierías Técnicas en Informática de Gestión y de Sistemas, para las cuales se establece la misma asignatura.

El presente proyecto docente se ajusta a los requisitos legalmente exigibles, en cuanto a la propuesta de una acción docente para la materia denominada en la plaza, "Ampliación de Algoritmos y Estructuras de Datos". Pero, al mismo tiempo, creemos que sería baldía la labor de planificar una asignatura ya extinta. Por ello, hemos considerado conveniente incluir, y hasta centrar, el presente documento en la organización de la enseñanza de acuerdo con los planes actualmente vigentes en la FIUM. Grosso modo, podemos avanzar que nuestra propuesta coincide en ambas asignaturas en la parte teórica, mientras que en las prácticas la mayor carga crediticia de AED permite la planificación de un mayor número de actividades.

Estructura del proyecto docente

La elaboración de un proyecto docente no es un mero requisito administrativo, sino que constituye una oportunidad inmejorable para mirar retrospectivamente la labor realizada hasta ese momento, para reflexionar críticamente sobre los aspectos en los que hemos percibido una necesidad de mejora o de cambio, y para planificar de cara al futuro la enseñanza de la materia con el objetivo de lograr los mayores niveles de calidad docente. El esfuerzo que supone su elaboración resulta necesariamente en un perfeccionamiento de las capacidades de organización y programación docentes del candidato.

Siguiendo un hilo conductor que va desde lo más general hasta lo más específico, empezamos estableciendo los fundamentos disciplinares de los algoritmos y las estructuras de datos en el Capítulo 1. Se analizan los conceptos, la historia y los principios subyacentes de la materia como disciplina científica y, por el momento, independientemente de consideraciones pedagógicas.

A continuación, en el Capítulo 2 se analiza el contexto en el que tendrá lugar el desarrollo de la actividad a la que se refiere la plaza. Este contexto incluye las dimensiones institucional, curricular, personal y profesional, todas ellas sujetas al dinamismo que impone el devenir de los tiempos. Este contexto determina las circunstancias bajo las que se ha de aplicar la docencia en la asignatura, desde los requisitos legales hasta las aspiraciones e intereses profesionales de los estudiantes.

En el Capítulo 3 se establecen las finalidades y objetivos educativos, desde un punto de vista general y en la materia objeto de concurso en particular. Si los anteriores capítulos eran meros análisis de la situación existente, éste marca el inicio de la toma de decisiones de planificación, entendiendo que el diseño de la asignatura debe guiarse por la consecución de los propósitos aquí fijados.

Una vez determinados los objetivos, en el Capítulo 4 se aborda la selección y organización de contenidos de la asignatura. Para llevar a cabo esta tarea no partimos de cero, sino que nos apoyamos en la tradición y trabajos previos en la docencia de la materia AED: el programa actual de la asignatura en la FIUM, las recomendaciones internacionales disponibles, y los programas de asignaturas relacionadas en otras universidades españolas.

Los contenidos propuestos para la asignatura son desarrollados a un mayor nivel de detalle en los tres siguientes capítulos. En particular, el Capítulo 5 describe los contenidos teóricos de la parte de estructuras de datos, el Capítulo 6 trata los contenidos de la parte de algoritmos, y el Capítulo 7 desarrolla los contenidos impartidos en los seminarios de prácticas. En todos estos capítulos se llega a un nivel de descripción bastante detallado. Con esto no queremos imponer un método o desarrollo

inflexible de las clases, sino más bien sugerir y dar recomendaciones sobre las explicaciones y actividades a realizar en cada apartado. De hecho, la temporización de los temas ofrece un margen flexible de actuación entre un ritmo más rápido o más lento.

Por último, en el Capítulo 8 se abordan las cuestiones pedagógicas del proyecto docente, discutiendo y planificando la metodología docente y las estrategias de evaluación a aplicar en la asignatura. Tanto en una como en otra es necesario conocer el amplio espectro de posibilidades, y utilizar una adecuada combinación de actividades docentes y mecanismos y criterios de evaluación, con el propósito de garantizar el cumplimiento de los objetivos marcados. La evaluación incluye la revisión del propio proceso docente, en función del cual el proyecto no es un documento cerrado sino que está sujeto a la revisión continua.

Los apéndices contienen alguna información adicional, como los programas actuales de las asignaturas relacionadas con AED, los resultados de algunos tests de evaluación inicial realizados a los alumnos, y un pequeño ejemplo del material de trabajo utilizado en clase (transparencias de clase, boletines de ejercicios, exámenes, página web de la asignatura, etc.).

A lo largo de todos los capítulos del proyecto docente aparecen de forma recurrente una serie de temas que caracterizan y condicionan la docencia universitaria en general y en la materia AED en especial, y que merece la pena destacar:

- La **calidad** de la docencia, como objetivo esencial e irrenunciable de la educación universitaria. La calidad y la excelencia aparecen en todos los apartados del proyecto, e implican el máximo nivel de exigencia en la preparación de la asignatura. Este principio universitario ha tomado cuerpo con la aprobación de la LOU, y constituye un eje básico en las tendencias que impulsan el desarrollo del Espacio Europeo de Educación Superior.

- El **dinamismo** como elemento caracterizador de nuestra sociedad moderna, y muy particularmente en la disciplina de la informática. La continua evolución en todos los órdenes de la vida ha dado lugar a numerosas implicaciones concretas en la elaboración de este proyecto: cambio del marco legal, revisión de los planes de estudios, nuevas recomendaciones curriculares, etc.

- Como consecuencia del dinamismo, es necesario ofrecer la suficiente **flexibilidad** en las propuestas realizadas, tanto en objetivos, como en contenidos, metodología y evaluación. También se hace necesaria la revisión y actualización continua del programa docente, ya que ninguna propuesta podrá perdurar indefinidamente.

- De forma más específica, las **recomendaciones internacionales** disponibles, y muy especialmente el informe Computing Curricula 2001, constituyen un elemento muy a tener en cuenta en cualquier titulación de informática, en cualquier lugar del mundo[1]. El informe es fruto del consenso y de la aportación de numerosos expertos en todos los ámbitos de la disciplina, por lo que no debe ser pasado por alto. Además, su elevado nivel de detalle resulta muy útil no sólo en el diseño de planes de estudios sino también en la elaboración de los programas de las asignaturas. Analizaremos estas recomendaciones en el contexto curricular, en la determinación de objetivos y en la selección de contenidos.

[1]De hecho, el citado informe ha sido utilizado como referencia e inspiración en la preparación del "Libro Blanco sobre el diseño de las titulaciones de informática", que marca los criterios con los que deberán ser elaborados los nuevos planes de estudios.

Índice general

1. Fundamentación disciplinar **1**
1.1. Algoritmos y estructuras de datos . . . 2
1.2. Perspectiva histórica . . . 8
1.2.1. La evolución de la programación . . . 8
1.2.2. Los principales avances en algoritmos y estructuras de datos . . . 15
1.3. Los principios básicos de la materia . . . 23
1.3.1. Abstracción . . . 23
1.3.2. Formalización . . . 26
1.3.3. Eficiencia . . . 29
1.4. El presente y el futuro de la disciplina . . . 31

2. El contexto de desarrollo **35**
2.1. El contexto institucional . . . 36
2.1.1. Marco legal . . . 37
2.1.2. El Espacio Europeo de Educación Superior . . . 43
2.1.3. La Universidad de Murcia . . . 48
2.1.4. La Facultad de Informática . . . 57
2.1.5. El Departamento de Informática y Sistemas . . . 63
2.2. El contexto curricular . . . 65
2.2.1. Directrices generales propias de las titulaciones de informática . . . 66
2.2.2. Las recomendaciones del informe Computing Curricula 2001 . . . 70
2.2.3. Planes de estudio de las titulaciones de informática en la FIUM . . . 79
2.2.4. Asignaturas relacionadas con la materia . . . 90
2.3. El contexto personal . . . 93
2.3.1. La procedencia de los alumnos . . . 94
2.3.2. Análisis retrospectivo del alumnado . . . 98
2.3.3. Cómo aprenden los alumnos . . . 101
2.4. El contexto social y profesional . . . 102
2.4.1. Las cifras de la sociedad de la información . . . 103
2.4.2. Perspectivas laborales en la Región de Murcia . . . 105
2.4.3. El informe del consorcio Career Space . . . 107

3. Finalidades y objetivos **113**
3.1. Finalidades de la educación universitaria . . . 113
3.2. Proceso y criterios de selección de objetivos . . . 116
3.2.1. La importancia de los objetivos educativos . . . 116
3.2.2. Clasificación de los objetivos . . . 117

3.2.3. Consideraciones sobre los objetivos propios de la materia . . . 118
3.3. Objetivos específicos de la materia . . . 124
3.3.1. Objetivos globales de la asignatura . . . 124
3.3.2. Objetivos puntuales de la asignatura . . . 127

4. Selección y organización de contenidos 131
4.1. Programa actual de la asignatura . . . 132
4.2. Recomendaciones internacionales . . . 134
4.2.1. La propuesta ACM-IEEE CC2001 . . . 135
4.2.2. Las propuestas UNESCO-IFIP . . . 142
4.3. La asignatura en otras universidades españolas . . . 144
4.3.1. La introducción a la programación . . . 144
4.3.2. La materia Algoritmos y Estructuras de Datos . . . 145
4.3.3. Conclusiones del análisis de planes de estudios . . . 165
4.4. Propuesta de la materia . . . 166
4.4.1. Decisiones más relevantes . . . 167
4.4.2. Propuesta de materia teórica . . . 180
4.4.3. Propuesta de materia práctica . . . 189
4.4.4. Coordinación entre teoría y prácticas . . . 207

5. Programa de Teoría: Parte I - Estructuras de Datos 209
Tema 0 - Estructuras de datos y algoritmos . . . 210
Tema 1 - Abstracciones y especificaciones . . . 213
Tema 2 - Conjuntos y diccionarios . . . 217
Tema 3 - Representación de conjuntos mediante árboles . . . 220
Tema 4 - Grafos . . . 224

6. Programa de Teoría: Parte II - Algoritmos 231
Tema 5 - Análisis de algoritmos . . . 231
Tema 6 - Ecuaciones de recurrencia . . . 234
Tema 7 - Divide y vencerás . . . 238
Tema 8 - Algoritmos voraces . . . 241
Tema 9 - Programación dinámica . . . 245
Tema 10 - Backtracking . . . 249
Tema 11 - Ramificación y poda . . . 254
Tema 12 - Resolución de juegos . . . 258

7. Programa de prácticas 261
7.1. Seminario de C . . . 261
Sesión 1 - Introducción, sintaxis y compilación de código C . . . 262
Sesión 2 - Tipos estructurados y punteros . . . 263
Sesión 3 - Funciones estándar y memoria dinámica . . . 264
Sesión 4 - Programación modular en C . . . 265
7.2. Seminario de C++ . . . 266
Sesión 1 - El lenguaje C++ como una extensión de C . . . 267
Sesión 2 - Clases y objetos en C++ . . . 268
Sesión 3 - Parametrización y excepciones . . . 269
7.3. Seminario de especificaciones formales . . . 270

8. Metodología docente y evaluación **273**
8.1. Objetivos y calidad de la enseñanza . . . 274
8.2. Métodos didácticos . . . 280
8.2.1. La lección magistral . . . 280
8.2.2. Las clases de problemas . . . 288
8.2.3. Los seminarios de prácticas . . . 290
8.2.4. Prácticas de laboratorio abierto . . . 291
8.2.5. Las tutorías . . . 293
8.3. Material docente . . . 294
8.3.1. Tipos de materiales docentes . . . 294
8.3.2. Bibliografía recomendada para la asignatura . . . 299
8.3.3. Material en formato electrónico . . . 306
8.4. Evaluación de la asignatura . . . 308
8.4.1. Objetivos y fases de la evaluación . . . 308
8.4.2. Evaluación de la teoría . . . 310
8.4.3. Evaluación de las prácticas . . . 312
8.4.4. Calificación global de la asignatura . . . 316
8.4.5. Evaluación de la actividad docente . . . 317

I Apéndices **321**

A. Programas de las asignaturas relacionadas **323**
A.1. Metodología y Tecnología de la Programación . . . 323
A.2. Bases de Datos . . . 327
A.3. Programación Orientada a Objetos . . . 330
A.4. Fundamentos de Ingeniería del Software . . . 334
A.5. Algoritmos y Programación Paralela . . . 338

B. Tests de evaluación inicial **341**
B.1. Resultados del test de evaluación inicial, curso 2001/02 . . . 341
B.2. Resultados de la encuesta inicial, curso 2002/03 . . . 347

C. Material de trabajo de clase **351**
C.1. Transparencias de clase del Tema 4 - Grafos . . . 351
C.2. Boletín de ejercicios del Tema 4 - Grafos . . . 356
C.3. Ejercicios del Seminario de especificaciones algebraicas . . . 359
C.4. Enunciado de la Práctica 1 - Análisis y diseño de estructuras de datos . . . 360
C.5. Examen de AAED de septiembre de 2003 . . . 364
C.6. Página web de la asignatura . . . 367

Índice de figuras

2.1. Dimensiones influyentes en el proceso de toma de decisiones curriculares. 36
2.2. Estructuración cíclica de las titulaciones. Situación actual en España y esquema de niveles propuesto en la Declaración de Bolonia. 46
2.3. Escudo de la Universidad de Murcia, con la leyenda: UNIVERSITAS STUDIORUM MURCIANA - ANNO MCCLXXII. 49
2.4. Los principales acontecimientos en la historia de la Facultad de Informática de la Universidad de Murcia. 58
2.5. Estrategias de implementación propuestas en el informe CC2001 para cada nivel. . . . 75
2.6. Relaciones de AED con otras asignaturas de su plan de estudios. Plan de II de 2002. . 90
2.7. El sistema educativo español actual, en todos sus niveles de cualificación. 95
2.8. Número de alumnos matriculados y aprobados en AAED. 99
2.9. Notas de examen de la asignatura AAED. 100
2.10. Ámbito de competencias del modelo de currículum propuesto por Career Space. 110

4.1. Paradigmas y lenguajes de programación usados en 33 titulaciones de 25 universidades españolas. 145
4.2. Coordinación entre teoría y práctica en la asignatura AED. 208

8.1. Mapa de notas de la asignatura AED. 317

Índice de tablas

2.1. Vicerrectorados existentes en la Universidad de Murcia y sus funciones. 57
2.2. Implantación y reforma de los planes de estudios impartidos en la FIUM. 59
2.3. Profesores con docencia en la Facultad de Informática de la UM durante el curso 2002/03, por departamento y categoría profesional. 61
2.4. Distribución de los profesores de DIS, por área y categoría profesional. 65
2.5. Directrices de la titulación Ingeniero en Informática, RD 1459/1990. 67
2.6. Directrices de I.T.I. de Gestión e I.T.I. de Sistemas, RD 1460 y 1461/1990. 68
2.7. Plan de estudios de Ingeniería Informática, plan de 1996, curso Primero. 80
2.8. Plan de estudios de Ingeniería Informática, plan de 1996, curso Segundo. 81
2.9. Plan de estudios de Ingeniería Informática, plan de 1996, curso Tercero. 81
2.10. Plan de estudios de Ingeniería Informática, plan de 1996, curso Cuarto. 82
2.11. Plan de estudios de Ingeniería Informática, plan de 1996, curso Quinto. 82
2.12. Plan de estudios de Ingeniería Informática, plan de 1996, asignaturas optativas. 83
2.13. Plan de estudios de Ingeniería Informática, plan de 2002, curso Primero. 86
2.14. Plan de estudios de Ingeniería Informática, plan de 2002, curso Segundo. 86
2.15. Plan de estudios de Ingeniería Informática, plan de 2002, curso Tercero. 87
2.16. Plan de estudios de Ingeniería Informática, plan de 2002, curso Cuarto. 87
2.17. Plan de estudios de Ingeniería Informática, plan de 2002, curso Quinto. 88
2.18. Plan de estudios de Ingeniería Informática, plan de 2002, asignaturas optativas de segundo ciclo, organizadas por intensificaciones. 88
2.19. Planes de estudio de Ingeniería Técnica en Informática de Gestión e Ingeniería Técnica en Informática de Sistemas. Planes de 2002. 89
2.20. Número de plazas y notas de corte de los diferentes cupos, para las titulaciones ofertadas por la FIUM, durante el curso 2002/03. 97
2.21. Número de alumnos, por cupo, que accedieron a las titulaciones ofertadas por la FIUM, durante el curso 2002/03. 97
2.22. Indicadores del sector de las TIC en Murcia y en España, durante el año 2002. 103
2.23. Distribución de la facturación de los distintos sectores de las TIC, en España, durante los años 2001 y 2002. 105
2.24. Inserción laboral de los egresados de la FIUM en los cursos del 1996/97 al 99/00. . . . 107

4.1. Unidades cubiertas por las asignaturas del plan de estudios de II, de 2002, en la FIUM, y por las asignaturas definidas en el CC2001, acercamiento imperativo primero. 138
4.2. Carga de trabajo de las asignaturas relacionada con la materia AAED. 167
4.3. Estructuración de temas de teoría propuesta para la asignatura AAED. 181
4.4. Asignación de objetivos educativos puntuales para cada tema de teoría. 182
4.5. Tabla de dedicación personal a las distintas etapas del proceso de desarrollo de software. 195

4.6. Planificación de las prácticas de AED. 197

5.1. Planificación temporal del Tema 0. 213
5.2. Planificación temporal del Tema 1, Parte I. 216
5.3. Planificación temporal del Tema 2, Parte I. 220
5.4. Planificación temporal del Tema 3, Parte I. 224
5.5. Planificación temporal del Tema 4, Parte I. 229

6.1. Planificación temporal del Tema 5, Parte II. 235
6.2. Planificación temporal del Tema 6, Parte II. 238
6.3. Planificación temporal del Tema 7, Parte II. 242
6.4. Planificación temporal del Tema 8, Parte II. 246
6.5. Planificación temporal del Tema 9, Parte II. 250
6.6. Planificación temporal del Tema 10, Parte II. 253
6.7. Planificación temporal del Tema 11, Parte II. 257
6.8. Planificación temporal del Tema 12, Parte II. 260

Acrónimos más comunes

AAED	Ampliación de Algoritmos y Estructura de Datos
ACM	Association for Computer Machinery
AED	Algoritmos y Estructuras de Datos
ANECA	Agencia Nacional de Evaluación de la Calidad y Acreditación
CC2001	IEEE & ACM Computing Curricula 2001
EEES	Espacio Europeo de Educación Superior
FIUM	Facultad de Informática de la Universidad de Murcia
IEEE	Institute of Electrical and Electronic Engineers
II	Ingeniería en Informática
ITIG	Ingeniería Técnica en Informática de Gestión
ITIS	Ingeniería Técnica en Informática de Sistemas
MTP	Metodología y Tecnología de la Programación
OO	Orientado a objetos
POO	Programación Orientada a Objetos
TAD	Tipo Abstracto de Datos
TIC	Tecnologías de la Información y las Comunicaciones
UM	Universidad de Murcia

Capítulo 1

Fundamentación disciplinar

"Tan pronto como exista una Máquina Analítica, no me cabe duda de que fijará los futuros derroteros de la ciencia. Y siempre que se busque un resultado por este medio, surgirá la pregunta: ¿cuál es el curso de computación mediante el cual puede la máquina obtener estos resultados en el menor tiempo posible?"

Charles Babbage, 1864

"Los conceptos fundamentales que emergieron hace 30 años no han cambiado significativamente; aunque la tecnología evoluciona, los conceptos permanecen."

Bertrand Meyer, Software Engineering in the Academy, 2001

El propósito último de un proyecto docente es planificar la enseñanza de una disciplina, a través de la definición de todos los aspectos que intervienen en una programación docente, desde los objetivos educativos, hasta la metodología y evaluación, pasando por los contenidos a tratar. Pero, independientemente de su posterior aplicación pedagógica, los propios fundamentos teóricos de la disciplina constituyen el primer ámbito de análisis a tener en cuenta. Sólo estableciendo claramente qué es lo que entendemos por "algoritmos y estructuras de datos" y estudiando las áreas propias de la materia y sus principios subyacentes, se podrá diseñar adecuadamente la enseñanza de dicha materia.

En este primer Capítulo se fundamenta la disciplina de los algoritmos y las estructuras de datos, como una de las áreas centrales de la informática. En primer lugar se hace un breve análisis de los **conceptos básicos** con los que nos vamos a enfrentar, intentando delimitar hasta dónde llegan los límites de nuestro objeto de estudio. A continuación damos una **perspectiva histórica**, seleccionando los avances más destacados y reconociendo a los pioneros que los llevaron a cabo. Después, volviendo a un plano intemporal, presentamos los que consideramos que constituyen los **principios elementales** subyacentes en el ámbito de los algoritmos y las estructuras de datos. Finalmente adoptamos una **visión de futuro**, embarcándonos en la compleja tarea de describir el estado actual del arte y la arriesgada misión de predecir las tendencias futuras de la disciplina.

1.1. Algoritmos y estructuras de datos

Todos nos hemos visto sometidos alguna vez, por parte de cierto amigo o conocido, a la pregunta –entre el compromiso de realizarla y la indiferencia por la respuesta– de ¿qué es lo que enseñamos en nuestra asignatura? No es fácil explicar a un completo profano en informática cuál es el ámbito teórico propio de los algoritmos y las estructuras de datos. Sin embargo, la contestación que surge de inmediato es la tan recurrida analogía de los algoritmos y las **recetas de cocina**: "*un algoritmo es como una receta de cocina, que está compuesta por una serie de pasos o instrucciones predefinidas que indican cómo hacer una comida*". La explicación funciona con los estudiantes novatos, y podría funcionar también con el inquisidor profano si no fuera por que decide mover la cabeza afirmativamente mientras desconecta completamente de nuestras palabras[1].

El papel de los algoritmos y de las estructuras de datos

Pero es bien conocido que la analogía de las recetas de cocina puede dar mucho más de sí para facilitar la comprensión de otros conceptos relacionados. Porque si un algoritmo es una receta de cocina, el cocinero que la interpreta y la lleva a cabo sería el procesador, la cocina sería el hardware –con sus periféricos y sus buses–, los ingredientes serían la entrada del algoritmo, y el pastel resultante sería la salida. ¿Dónde estarían entonces las estructuras de datos? Las estructuras de datos serían los recipientes –cada uno con su tamaño, forma y distribución particular– donde se almacenan los ingredientes de entrada, los resultados de salida y cualquier subproducto utilizado en los pasos intermedios.

Está, pues, bastante claro que los algoritmos no pueden funcionar sin las estructuras de datos, de la misma forma que las recetas no se pueden llevar a cabo sin los recipientes necesarios donde meter la masa y los ingredientes. Algoritmos y estructuras de datos trabajan conjuntamente, y se complementan mutuamente en la tarea que conduce a la resolución de un problema. La relación entre ambos conceptos se expresa en la magistral fórmula del profesor Niklaus Wirth: Algoritmos + Estructuras de datos = Programas. Cualquier algoritmo trabajará siempre con datos almacenados en determinada estructura, y cualquier estructura de datos necesitará los algoritmos de manipulación adecuados.

Sin embargo, eso no quiere decir que ambos conceptos sean indistinguibles. Son, más bien, los dos pilares básicos de la programación que establecen: cómo se almacena de información, y cómo se manipula la información. En algunos problemas las estructuras de datos tendrán una mayor relevancia, como en las aplicaciones de bases de datos y sistemas de información. En otros ámbitos predominará la componente algorítmica, como en inteligencia artificial o computación gráfica. Pero en cualquier caso siempre estarán presentes ambos elementos.

Retomando nuevamente la analogía de las recetas de cocina, podemos establecer que la receta no sólo *es como* un algoritmo, sino que de hecho *es* un algoritmo si las instrucciones que contiene no dejan margen a la interpretación subjetiva sobre su ejecución. Por ejemplo, las indicaciones "cocinar hasta que esté bien hecho", o "añadir harina hasta que la masa se compacte", están entre las que dan lugar a los mayores fracasos culinarios, y son inadmisibles en un algoritmo. Entonces surge inmediatamente otra cuestión, casi de índole filosófico: ¿es que pueden haber algoritmos aparte de los ordenadores? ¡Evidentemente! La existencia y validez de un algoritmo es independiente del actor externo encargado de interpretarlo y llevarlo a cabo. La tarea de *ejecutar* un algoritmo es un proceso objetivo, que podría ser realizado igualmente por un humano, un ordenador, o la máquina analítica de Charles Babbage –en caso de que hubiera llegado a construirla–, siempre que esté descrito con un conjunto de reglas

[1] Aunque, sin conocerlo, él mismo está aplicando el algoritmo del célebre programa de psicoanálisis ELIZA, creado en 1966 por Joseph Weizenbaum, consistente en seleccionar una palabra clave de nuestro discurso con la que formular la siguiente pregunta irrelevante.

comprensible por el actor correspondiente. La ejecución de un algoritmo no implica inteligencia, sino que la inteligencia reside en el propio diseño del algoritmo. Se puede decir que el algoritmo es una *codificación* de la inteligencia de un humano en la resolución de cierto problema.

No debemos perder la visión práctica, ni olvidar que nuestro objetivo principal es la construcción de algoritmos para ser traducidos a lenguajes de programación y ejecutados en ordenadores. Pero tampoco debemos dejar de considerar que los algoritmos tienen su existencia propia, separada de la de los ordenadores. El análisis y diseño de algoritmos y estructuras de datos constituye, por lo tanto, un campo central y básico de la disciplina de la informática, dentro del lado software, frente al otro gran ámbito constituido por el hardware.

Etimología e historia antigua

Es ampliamente reconocido que buena parte de los desarrollos teóricos de conceptos y técnicas que se utilizan aún hoy en día en el ámbito de los algoritmos fueron realizados antes de que se inventaran los ordenadores, algunos de ellos muchos siglos antes. De hecho, podríamos datar la aparición de los primeros algoritmos con la primera vez que los humanos decidieron expresar de forma genérica sus conocimientos sobre el mecanismo de resolución de cierto problema. Por ejemplo, el proceso para encender fuego requiere aplicar una serie de pasos que se pueden expresar de forma objetiva y genérica: coger dos palos secos, frotar durante cierto tiempo, acercar la yesca, soplar, etc. No obstante, esto no implica la existencia explícita de un algoritmo, ya que el proceso podría haber sido transmitido mediante ejemplos. Por ello, y por la necesidad de omitir todo aquello carente de objetividad, normalmente se buscan los primeros ejemplos de algoritmos en los tratados antiguos de aritmética y geometría.

Ciñéndonos al ámbito de las matemáticas, los primeros algoritmos conocidos datan de hace unos 4000 años, en la por aquel entonces influyente Babilonia. Los babilonios ya trabajaban con el sistema sexagesimal en punto flotante y con primitivos algoritmos, aunque no contaban con operaciones de control de flujo. Uno de estos algoritmos es el conocido como **método babilónico para el cálculo de la raíz cuadrada**. Supongamos que queremos calcular la raíz cuadrada de un número positivo a cualquiera. En cierto momento tendremos una aproximación más o menos precisa x. Basándonos en el hecho de que a/x es otra estimación posible del valor buscado, $\sqrt{a}$, tenemos el siguiente algoritmo.

1. Sea a un número positivo, y sea $x = 1$
2. Repetir varias veces
    2.1. Calcular a/x
    2.2. Calcular la media entre x y el valor calculado en el paso 2.1
    2.3. Asignar a x el valor calculado en el paso 2.2
3. La estimación de $\sqrt{a}$ es x

El paso 2 se repetiría más o menos veces en función de la precisión deseada. Esta cantidad no está perfectamente definida, aunque podemos obviar este pequeño detalle. Realmente, lo que ocurre es que el número de ejecuciones es otro parámetro del algoritmo.

La mayoría de los algoritmos desarrollados y utilizados en estos tiempos remotos tienen una apariencia similar, se aplican en la resolución de problemas aritméticos y geométricos, y se basan principalmente en mecanismos de iteración. Por ejemplo, el método de **Arquímedes de Siracusa** (Sicilia) para calcular el valor de π, escrito sobre el siglo III a.C., se basa en una iteración que converge poco a poco al valor buscado.

El salto cualitativo que constituye la introducción de instrucciones de control de flujo tiene lugar también durante estas épocas pretéritas. Es **Euclides de Alejandría** (Egipto) –de quien fuera

discípulo Arquímedes–, el que aplica por primera vez la idea, en algún momento en torno al año 300 a.C., en el diseño de su famoso algoritmo para calcular el máximo común divisor de dos números. Supongamos que queremos calcular el máximo común divisor de dos números enteros, a y b, siendo $a > b$. El algoritmo de Euclides propone el siguiente método de cálculo.

1. Hacer r:= a módulo b
2. Si $r = 0$ entonces el máximo común divisor es b
3. En otro caso:
    3.1. Hacer a:= b
    3.2. Hacer b:= r
    3.3. Volver al paso 1

Pero el honor de dar nombre al término *algoritmo* lo recibe el matemático persa del siglo IX **Muhammad ibn Musa al-Khwarizmi**[2]. Sus tratados sobre la resolución de ecuaciones de primer y segundo grado ponen de relieve la idea de resolver cálculos matemáticos a través de una serie de pasos predefinidos. Entre sus principales contribuciones está el libro "Hisab al-jabr w'al-muqabala" ("El cálculo de reducción y restauración"), escrito sobre el año 825 d.C., que constituye una colección de algoritmos de gran trascendencia en el desarrollo científico de la época.

A modo de ejemplo, vamos a ver el método que al-Khwarizmi propone para resolver ecuaciones de segundo grado del tipo: $x^2 + bx = c$. Mostramos la aplicación sobre el ejemplo $x^2 + 10x = 39$, que es el mismo que al-Khwarizmi presenta originalmente en uno de sus libros.

1. Tomar la mitad de b. (En el ejemplo, $10/2 = 5$)
2. Multiplicarlo por sí mismo. (En el ejemplo, $5 * 5 = 25$)
3. Sumarle c. (En el ejemplo, $25 + 39 = 64$)
4. Tomar la raíz cuadrada. (En el ejemplo, $\sqrt{64} = 8$)
5. Restarle la mitad de b. (En el ejemplo, $8 - 10/2 = 8 - 5 = 3$)

El resultado es: $\sqrt{(b/2)^2 - c} - b/2$, expresado *algorítmicamente* –y valga la redundancia–. Tenemos que indicar que la notación que hemos usado es adaptada, ya que al-Khwarizmi no utiliza símbolos, expresiones ni variables en su libro, sino una descripción completamente textual. Por ejemplo, a b lo llama "las raíces" y a c "las unidades".

Aunque, como ya hemos visto, la idea que propone al-Khwarizmi no es realmente nueva, sus trabajos tuvieron una enorme repercusión cuando **Fibonacci** (Leonardo de Pisa, Italia) traduce su obra alrededor del año 1200 d.C., dando a conocer sus propuestas al mundo occidental. El título de uno de los tratados es traducido al latín como "Algoritmi de numero Indorum", algo así como "El libro de al-Khwarizmi sobre el arte hindú de calcular". Es esta deformación la que da origen al término *algoritmo*. A pesar de que la idea no fuera completamente original, sin duda alguna las aportaciones de al-Khwarizmi como introductor en Europa[3] del sistema de numeración posicional en base 10, así como del uso del 0 como marcador de posición, merecen tal honor.

Por su parte, el término *estructura de datos* tiene una etimología mucho menos interesante. La palabra procede del latín *structûra*, y según el diccionario de la R.A.E. significa "*distribución de las partes del cuerpo o de otra cosa*". Ninguno de los algoritmos que hemos mencionado antes requería el uso de más de unos pocos números, por lo que el problema de estructurar la información sólo se hizo evidente cuando, con la llegada de los ordenadores, el incremento en el volumen de información hizo

[2]Cuya traducción significaría algo así como Mohamed hijo de Moisés de Khorezm, lugar situado en algún punto de Oriente Medio.

[3]Sin olvidar tampoco la labor de Fibonacci como traductor.

necesario buscar formas eficientes de representar los datos. Desarrollos teóricos en ámbitos tales como las colas y las pilas tuvieron lugar durante el pasado siglo XX.

Algo más de tradición tiene la teoría de grafos, cuyo inicio se data en el siglo XVIII, cuando el matemático suizo **Leonhard Euler** propone y resuelve el problema conocido como "los puentes de Königsberg": encontrar si existe un recorrido en una ciudad que pase por todos los puentes de la misma una sola vez, sin repetir ninguno, y acabando en el sitio donde se empezó. El problema es modelado a través de un concepto abstracto, un grafo, siendo la primera vez aparece de forma explícita. Euler establece las condiciones necesarias y suficientes para determinar si el recorrido existe o no.

Definiciones básicas relacionadas

Si bien, como ya hemos visto, la historia de los algoritmos se remonta muy atrás, el estudio de los algoritmos y las estructuras de datos no se concibió como una disciplina propia hasta bien entrada la segunda mitad del pasado siglo XX. Actualmente, entendemos la **algoritmia** como la disciplina –ciencia y arte–, dentro del ámbito de la informática, que estudia técnicas para construir algoritmos eficientes y técnicas para medir la eficiencia de los algoritmos[4]. En consecuencia, la algoritmia consta de dos grandes áreas de estudio: el análisis y el diseño de algoritmos.

El **análisis de algoritmos** es la parte de la algoritmia que estudia la forma de medir la eficiencia de los algoritmos. Utilizando un punto de vista empresarial, podemos definir la eficiencia como la relación entre los recursos consumidos –fundamentalmente tiempo y memoria– y los productos obtenidos –resolución exacta o aproximada de problemas–. Por su parte, el **diseño de algoritmos** es la parte de la algorítmica que estudia técnicas generales de construcción de algoritmos eficientes. El objetivo no es, por lo tanto, proponer algoritmos concretos sino desarrollar técnicas generales que se puedan aplicar sobre una amplia variedad de problemas. Hay que reconocer que, a pesar de los grandes avances en la materia, el número existente de estas técnicas generales sigue siendo muy reducido.

Ya hemos hablado informalmente del concepto de algoritmo, así que ha llegado el momento de definirlo con más rigor.

Definición 1. Un **algoritmo** es una serie de reglas que dado un conjunto de **datos de entrada** (posiblemente vacío) produce unos **datos de salida** (por lo menos una) y cumple las siguientes propiedades:

- **Definibilidad.** El conjunto de reglas debe estar definido sin ambigüedad, es decir, no pueden existir dudas ni subjetividad en su interpretación.
- **Finitud.** El algoritmo debe constar de un número finito de pasos, que se ejecutan en un tiempo finito y requieren un consumo finito de recursos.

Donald E. Knuth, uno de los grandes pioneros en el estudio de los algoritmos y la programación, propone la siguiente definición formal de algoritmo.

Definición 2. Un **método de cálculo** es una cuaterna (Q, I, W, f), en la cual:

- I es el conjunto de **estados de entrada**;
- W es el conjunto de **estados de salida**;
- Q es el conjunto de **estados de cálculo**, que contiene a I y W; y

[4]De acuerdo con el diccionario de la R.A.E., el término *algoritmia* se refiere a la ciencia de los algoritmos, mientras que *algorítmica* es todo lo "*perteneciente o relativo al algoritmo*".

- f es una función: $f : Q \to Q$, con $f(w) = w; \forall\ w \in W$.

Definición 3. Una **secuencia de cálculo** es una serie de estados: $x_0, x_1, x_2, ..., x_n, ...$, donde $x_0 \in I$ y $\forall k \geq 0; f(x_k) = x_{k+1}$. Se dice que la secuencia de cálculo acaba en n pasos si n es el menor entero con $x_n \in W$.

Una definición formal como esta puede ser útil para comprobar, de manera rigurosa, si un algoritmo cumple las propiedades de finitud y definibilidad. Sin embargo, intentar resolver un problema definiendo la función f puede resultar una tarea bastante compleja y totalmente inviable. Y, lo que es peor, el resultado puede que nos aporte pocas ideas sobre cómo implementar el algoritmo en un lenguaje de programación. En este sentido creemos que no es adecuado formalizar más de lo conveniente.

Por otro lado, también debemos referirnos a los conceptos relacionados con tipos y estructuras: tipos abstractos de datos, tipos de datos y estructuras de datos. Estos conceptos no son, en absoluto, redundantes ni incoherentes, sino que son ideas que aparecen en los distintos niveles de desarrollo. Un tipo abstracto es un concepto de diseño y por lo tanto previo e independiente de cualquier implementación. Un tipo de datos es un concepto de programación y se puede ver como la realización o materialización de un tipo abstracto. La estructura de datos es la organización o disposición en memoria de la información almacenada para cierto tipo de datos. Veamos, por partes, las definiciones.

Definición 4. Un **Tipo Abstracto de Datos** (**TAD**) está compuesto por un dominio abstracto de valores y un conjunto de operaciones definidas sobre ese dominio, con un comportamiento específico.

Definición 5. El **tipo de datos** de una variable, un parámetro o una expresión, determina el conjunto de valores que puede tomar esa variable, parámetro o expresión, y las operaciones que se pueden aplicar sobre el mismo.

Definición 6. La **estructura de datos** de un tipo de datos es la disposición en memoria de los datos necesarios para almacenar los valores de ese tipo.

Una vez hecho este repaso breve y necesario de los conceptos más relevantes en la materia objeto de concurso, debemos hacer dos consideraciones en referencia a la denominación de la actividad a desarrollar en la plaza "Ampliación de algoritmos y estructuras de datos":

1. Aunque la denominación habla de "estructuras de datos" –como, por otro lado, es común en la literatura relacionada–, la materia incluye también los aspectos relacionados con tipos abstractos. Es decir, no quedan fuera del ámbito de estudio los conceptos abstractos de la teoría de listas, colas, árboles y grafos.

2. La denominación comienza con la indicación "ampliación de", haciendo referencia a una asignatura concreta de un plan de estudios. El hecho de tratarse de una ampliación indica la suposición de un nivel inicial de conocimientos, y el foco en técnicas más o menos avanzadas.

Problemas algorítmicos

¿Cuál es el objetivo último de la algoritmia? ¿Por qué es tan importante su estudio? Sin duda alguna, el propósito final es la **resolución de problemas**: dado un problema particular que nos resulta de interés, ser capaz de resolverlo de la mejor manera posible, ofreciendo una solución rápida, corta, elegante y fácil de programar; y recordemos que en esta resolución entran en juego tanto los algoritmos como las estructuras de datos.

Pero, desde un punto de vista amplio, cualquier campo de la ciencia o de la ingeniería podría atribuirse el objetivo de perseguir la resolución de problemas. Deberíamos hablar más específicamente

de la resolución de **problemas algorítmicos o computacionales**. Un problema algorítmico consta de los siguientes elementos, [Harel'92]:

1. Una caracterización del conjunto de todas las posibles entradas legales del problema, posiblemente infinito.

2. Una especificación de las salidas deseadas, como función de las entradas.

Un algoritmo se dice que es correcto si obtiene los resultados deseados para todas las posibles entradas. Por lo tanto, la corrección de un algoritmo sólo se puede determinar en función de su especificación.

Diseñar algoritmos que resuelvan problemas de forma eficiente es la meta de la algoritmia. Sin embargo, no siempre podemos estar seguros de que existan tales algoritmos capaces de resolver los problema de interés, o al menos algoritmos *razonables*. Se dice que un algoritmo es **razonable** si su tiempo de ejecución en función del tamaño de la entrada es una función polinomial. Un problema se dice que es **tratable** si puede ser resuelto con algún algoritmo razonable. Si el problema no admite soluciones que funcionen en tiempo razonable –por ejemplo, sólo puede ser resuelto en tiempos factoriales o exponenciales– se dice que es **intratable**.

Los problemas tratables se encuentran dentro de lo que la teoría de la computabilidad denota como la **clase P**. Frente a esto, la **clase NP** denota los problemas que pueden ser resueltos en tiempo polinomial usando un hipotético algoritmo no determinista[5], que fuera capaz de tomar decisiones acertadas en determinados puntos clave. Dentro de esta clase, los **problemas NP-completos** son un subconjunto conocido, y cuya particularidad es que resultan equivalentes entre sí –o, más específicamente, se pueden reducir de unos a otros mediante una transformación que requiere un tiempo polinomial–. Es más, cualquier problema NP se puede reducir a un NP-completo.

Mientras que los problemas tratables están en la clase P, como ya hemos comentado, no se puede asegurar que los problemas NP sean tratables o intratables. La pregunta ¿P=NP? es una de las grandes cuestiones abiertas de la informática teórica desde que fuera propuesta en 1971; y una de las más intrigantes. Muchos sospechan que P≠NP, pero el hallazgo de un algoritmo razonable que resolviera alguno de los problemas NP-completos, haría que todos los problemas de NP fueran tratables.

La algoritmia aborda la resolución de problemas tanto de la clase P como de la NP. La dificultad implícita de la segunda no significa que no resulte interesante su estudio. Más bien al contrario, estos problemas requieren usar toda la creatividad y los conocimientos disponibles para diseñar algoritmos adecuados. Muchas veces una solución "no completamente correcta" es mejor que no tener ninguna solución. En esos casos se hace necesario diseñar algoritmos de aproximación, basados en probabilidad, heurísticas u otras ideas que no siempre garantizan el resultado esperado. Entre los problemas NP-completos clásicos en algoritmia podemos señalar: el problema del puzzle o del embaldosado, del ciclo hamiltoniano, del viajante o agente de comercio, de empaquetamiento en recipientes, de asignación, de coloración de grafos, de isomorfismo de grafos, de la mochila 0/1, y de satisfacibilidad de una expresión booleana.

Pero la categoría de problemas no acaba aquí, y podemos encontrar otros tipos de problemas aun más complejos. Los **problemas indecidibles**, o no computables, son aquellos para los cuales no puede existir ningún algoritmo que los resuelva. Un ejemplo de esta clase es el conocido como problema del bloqueo, consistente en determinar si un programa dado finaliza su ejecución o no –es decir, se queda colgado–. Se puede demostrar, por contradicción, que tal programa no puede existir.

[5]Recordemos que NP no significa "no polinomial" sino "polinomial no-determinista".

A su vez, los problemas indecidibles se clasifican en: parcialmente decidibles –para los que se podría obtener respuesta en algunos casos–, y los altamente indecidibles –que podrían no producir respuesta en ningún caso–. Por ejemplo, si no se permite recursividad el problema del bloqueo es parcialmente decidible, y en caso de que se permita entonces es altamente indecidible. No entraremos más en profundidad en estas cuestiones, ya que consideramos que forman parte más propiamente del campo de la computabilidad.

1.2. Perspectiva histórica

Ya hemos visto que la historia de los algoritmos se remonta a mucho antes de la aparición de los primeros ordenadores. Pero es evidente que la mayoría de los avances relevantes se concentran en la segunda mitad del pasado siglo XX. Conocer la evolución histórica de la disciplina, cuándo tuvieron lugar los principales avances, y cuáles fueron los ámbitos de trabajo en el pasado, siempre resulta interesante, no sólo por la curiosidad histórica y el reconocimiento hacia los pioneros, sino también para adquirir una visión de futuro bien fundamentada.

En este apartado vamos a hacer una breve reseña histórica de las tendencias que han marcado la evolución de la disciplina, desde sus orígenes. Empezaremos en primer lugar haciendo referencia a la historia de la programación, relacionada de forma muy directa y estrecha con la materia objeto de estudio. Para esta revisión hemos hecho uso de [Fernández'02], donde se hace un repaso de los primeros años de la programación. A continuación trazamos una línea temporal, sobre la que señalamos los principales avances en el diseño de algoritmos y estructuras de datos.

1.2.1. La evolución de la programación

Si distinguimos y separamos entre la historia de los algoritmos y la de la programación, podemos situar el comienzo de la segunda en el momento en que aparecieron las primeras máquinas capaces de ser programadas. Existen precedentes mecánicos de los ordenadores actuales, como el telar controlado con tarjetas perforadas del francés Joseph Jaquard, en 1801, la máquina diferencial del matemático inglés Charles Babbage, de 1833, la máquina tabuladora del ingeniero americano Herman Hollerith, sobre 1890, o las varias calculadoras electromecámicas del ingeniero español Leonardo Torres Quevedo –incluyendo un jugador de ajedrez simple–. Los primeros ordenadores electrónicos de uso general no aparecen hasta la década de 1940, motivados por las necesidades de cálculo de los físicos y astrónomos, pero fundamentalmente por la influencia de la Segunda Guerra Mundial. Es ahí cuando fijamos el inicio de la corta pero trepidante historia de la programación.

Los primeros lenguajes y el arte de programar

Los primeros ordenadores eran máquinas pesadas y muy voluminosas, que ocupaban varias habitaciones. Cada computadora era única en su diseño y conjunto de instrucciones, por lo que el código no trascendía más allá de la habitación donde estaba ubicada. Además, eran lentas y tenían una memoria escasa, por lo que los programadores debían ingeniárselas para sacar el mayor partido de la máquina. Por este motivo, en sus inicios, la programación consistía básicamente en optimizar la eficiencia del proceso computacional, donde los programadores construían sus algoritmos con las instrucciones de una máquina particular. No existían aún los lenguajes de alto nivel, por lo que la **programación en lenguaje máquina** es la característica fundamental de este primer periodo.

Al aumentar la capacidad de cálculo de los ordenadores, la sociedad empezó a demandar más aplicaciones, trasladando la responsabilidad al programador que no disponía de medios suficientes para

una tarea de tal envergadura. Los programadores se sentían desbordados por la gran cantidad de errores y deficiencias, que les era imposible controlar. La solución se buscó en el diseño de nuevos lenguajes que ofrecieran más facilidades para programar, los **lenguajes de alto nivel**. FORTRAN primero, en 1957, y más tarde ALGOL 60 vinieron a mitigar la situación. FORTRAN se difundió ampliamente en EE.UU., mientras que ALGOL 60 tuvo mayor aceptación en Europa.

El concepto de **subrutina** es incorporado por primera vez en el lenguaje de programación FORTRAN, y se concebía como un medio para reducir la longitud del programa, facilitar futuros cambios y reflejar el análisis del programador. Como señala Edsger W. Dijkstra en su conferencia de 1972 "*The Humble Programmer*"[6], "*la subrutina es el gran patrón de abstracción, y sólo los problemas que admiten una solución factorizada serán resueltos de forma satisfactoria*". Con ALGOL 60 aparecieron los **procedimientos**, y todo lo que ello implica: las variables locales, los parámetros, la recursión, y los conceptos relacionados de ámbito y extensión. Muchas de las nuevas ideas, como la recursión, eran tan sofisticadas para los programadores que no llegaban a comprender su utilidad, y cuando lo hacían las encontraban poco eficientes.

En los años 60 aumentaron la complejidad y el tamaño de los proyectos, sin ir acompañados de una mejora en las herramientas y en las habilidades de los programadores. Esto provocó un auténtico caos en la industria del software: proyectos que no cumplían la fecha ni los costes previstos (retrasándose una y otra vez), software poco fiable e incorrecto, programas con excesivas líneas de código, software lento, incapacidad para satisfacer los requisitos y para tener en cuenta la realimentación de los clientes, siendo el software muy difícil de mantener. En la programación predominaba la intuición y las capacidades personales del programador, siendo notable la falta de un metodología ingenieril. Además, los lenguajes ofrecían un nivel muy bajo de abstracción, siendo muy recurrido el uso de instrucciones *goto* y punteros. En definitiva, es la etapa de la denominada "crisis del software", expresión acuñada en una conferencia de la OTAN de 1968 donde nació la disciplina de Ingeniería del Software.

Programación estructurada

Como respuesta a esta situación de crisis del software, E.W. Dijkstra, C.A.R. Hoare y O.J. Dahl desarrollaron y propusieron hacia finales de los 60 un método sistemático de programación que iba a dominar la disciplina durante los años ulteriores, **la programación estructurada**, además de introducir conceptos e ideas que, a la postre, serían el germen del paradigma orientado a objetos. Wirth definió la programación estructurada como "*la formalización de programas como estructuras jerárquicas y anidadas de instrucciones y objetos de computación*". Básicamente, las dos características clave relacionadas con la programación estructurada son las **sentencias estructuradas** de control del flujo y el **refinamiento por pasos sucesivos**.

Cuando en 1968 Dijkstra propone la eliminación de la instrucción *goto* en todos los lenguajes de alto nivel –en su influyente artículo "*Go to Statement Considered Harmful*"–, surge un auténtico revuelo entre los programadores de aquellos años. Dijkstra considera bastante limitada la capacidad intelectual humana para visualizar procesos que evolucionan en el tiempo, estando mejor adaptada para dominar relaciones estáticas. Buena parte de la crisis del software era debida a la *lógica espagueti* que guiaba la construcción de los primeros programas. En proyectos de cierto tamaño se hacía completamente imposible controlar la corrección de un programa después de haberlo escrito. En su lugar, se propone el uso de sentencias de control estructuradas, como las condicionales y los iteradores *mientras*, *para*, y *repetir*. Estas sentencias acortan el salto conceptual existente entre el proceso dinámico y el proceso estático de un programa, es decir, entre la estructura de un programa, extendida en un texto, y su proceso de ejecución, desplegado en el tiempo.

[6] Preparada con motivo de la entrega del Premio Turing de 1972.

Por su parte, el refinamiento por pasos sucesivos es propuesto por Niklaus Wirth en 1971. Esta técnica intuitiva consiste en establecer una secuencia de decisiones de diseño al descomponer tareas en subtareas y datos en estructuras de datos. El refinamiento sucesivo de tareas finaliza cuando todas las instrucciones se expresan en términos de un lenguaje de programación. La decisión tomada en cada refinamiento depende de criterios, a veces contrapuestos, sobre eficiencia de espacio y memoria, claridad y regularidad de la estructura. De igual manera, los datos son también refinados en paralelo, permitiendo la comunicación entre las subtareas obtenidas.

Los padres de la programación estructurada abogaban por un lenguaje simple y sistemático que pudiese expresar de forma natural la estructura del programa y de los datos que surgen en el proceso de refinamiento. El lenguaje debería permitir reflejar en la estructura de lo que escribimos todas las abstracciones necesarias para tratar conceptualmente la complejidad de lo que estamos diseñando. Dijkstra señala que el lenguaje de programación debería mantener el "control intelectual" del programa, por ejemplo, imponiendo la exclusión de instrucciones *goto* y de procedimientos con más de un parámetro de salida. El lenguaje **Pascal**, propuesto por Wirth en 1971, constituirá el instrumento esperado por los programadores para seguir esta corriente de pensamiento.

Con el desarrollo de la programación estructurada aparecieron cada vez más lenguajes de programación que soportaban sus principios. A nivel académico, Pascal se convirtió en el auténtico dominador dentro los cursos introductorios de programación. Simultáneamente, se realizaron numerosos esfuerzos en el campo más teórico de la **verificación formal de programas**. La investigación en este área, que continúa aún hoy día, intenta encontrar mecanismos de especificación precisos que permitan demostrar formalmente la corrección de los programas. Las propuestas se ven sujetas a moverse en la difícil disyuntiva entre el rigor matemático y la aplicabilidad en sistemas grandes y complejos.

Paradigma declarativo

Los lenguajes declarativos se centran en lo que la computadora debe hacer, a diferencia de los imperativos donde el énfasis se pone en *cómo* debería hacerlo. Dentro de los lenguajes declarativos podemos distinguir los **lenguajes lógicos** y los **funcionales**. Los lenguajes funcionales emplean un modelo computacional inspirado en el *lambda cálculo*, basado en la definición recursiva de funciones; un programa es una función que a partir de unas entradas produce unas salidas. Los lenguajes lógicos se basan en la definición de una serie de axiomas, a partir de los cuales se pueden demostrar teoremas.

El modelo funcional, al igual que el imperativo, procede del trabajo de matemáticos como Alan Turing y Alonzo Church en los años 1930. Trabajando de forma independiente, se desarrollaron formalizaciones muy diferentes de la noción de algoritmo, basadas en autómatas, manipulación simbólica, definiciones de funciones recursivas, etc. Se mostró que cualquier algoritmo que podía ser construido en una, lo podía ser en las otras. En base a este hecho, Church conjeturó que cualquier modelo de computación tenía la misma expresividad: la conocida como tesis de Church. El modelo de computación de Turing estaba compuesto por un autómata con la capacidad de acceder a las celdas de una cinta con capacidad ilimitada. La máquina de Turing actuaba en un estilo imperativo, cambiando los valores de las celdas de su cinta, de forma similar a como en un lenguaje imperativo se cambian los valores de las variables. Por su parte, el modelo de computación de Church, llamado *lambda cálculo*, se basaba en la noción de expresiones parametrizadas y fue el modelo en el que se inspiró la programación funcional. Las computaciones se realizan al sustituir los parámetros dentro de las expresiones, de forma similar a como se efectuaban al comunicar los argumentos de una función en un lenguaje funcional. Ejemplos de este tipo de lenguajes son ML, Haskell y LISP, muy utilizado en el campo de la inteligencia artificial por su capacidad para manejar datos simbólicos.

Por otra parte, los lenguajes lógicos están fundamentados en la lógica proposicional. En un

programa lógico, el programador establece una serie de axiomas a partir de los que se pueden demostrar teoremas. La computación se modela como un intento de encontrar valores que satisfacen un objetivo, mediante los axiomas introducidos y ciertas relaciones especificadas a través de una serie de reglas lógicas. Los lenguajes lógicos se han aplicado tradicionalmente en especificaciones formales y para realizar demostración de teoremas. Prolog es el lenguaje lógico más conocido y utilizado, siendo en muchos casos la base de otros lenguajes.

Programación modular y tipos abstractos de datos

Al mismo tiempo que Dijkstra exponía sus ideas sobre la descomposición funcional de los programas, Hoare hizo lo propio en lo que se refiere a la estructuración de los datos. Estaba claro que la abstracción de funciones no resultaba suficiente para facilitar la construcción de grandes programas, y era necesario centrar la descomposición en los datos, más que en el código. Hoare pensaba que el programador no sólo debía ser capaz de definir sus propias operaciones sino también sus propios tipos de datos. Estableció una diferencia entre la especificación y la implementación de un tipo de datos, y propuso un método para verificar la corrección de las representaciones de los datos, esto es, si la implementación representa el modelo abstracto matemático.

Estos trabajos fueron el germen para la aparición del concepto de **tipo abstracto de datos (TAD)**, que es usado por primera vez por Barbara Liskov y S. Zilles en un artículo de 1974. En este artículo se introducía la abstracción de datos como un nuevo tipo de datos útil en el dominio del problema a resolver. El TAD constaba de un conjunto de objetos y un conjunto de operaciones que caracterizaban el comportamiento de los objetos. Se distingue claramente entre el uso y la construcción de la nueva abstracción de datos. En el primer caso el programador únicamente está interesado en el comportamiento de los elementos del nuevo tipo, es decir, qué tipo de información se puede almacenar, recuperar y modificar de ellos. En el segundo caso el programador debe considerar cuestiones como la representación de los valores del tipo y los algoritmos que implementan las operaciones de almacenamiento y recuperación de información. En definitiva, existen dos perspectivas de un TAD: de cara al usuario del tipo, y de cara al implementador. Además, se establece el principio de que los tipos definidos se deberían poder usar igual que los tipos ya incorporados en el lenguaje.

La especificación de un TAD es lo que relaciona al usuario del tipo con el implementador. En 1977 Guttag propone el uso del enfoque axiomático para definir formalmente la semántica de cada operación de un tipo, con independencia de su implementación, empleando especificaciones algebraicas. De esta forma, la especificación de un TAD se podía complementar perfectamente con el modelo de desarrollo *top-down*. Mediante la especificación del TAD, en las primeras etapas del refinamiento de un programa se conocería perfectamente el significado de cada operación del tipo, con el fin de ser capaz de establecer la comunicación entre diferentes funciones. Cuando se tuviera suficiente información en niveles de refinamiento posteriores, se llevaría a cabo la implementación del TAD.

Por otro lado, en 1972 David Parnas presentó la idea de los **módulos**, concebidos como un conjunto de subrutinas que producen cambios en el estado de un proceso computacional, junto con otra serie de subrutinas encargadas de proporcionar los valores de las variables que constituyen tal estado. Cada subrutina vendría acompañada de una especificación propia.

El principal objetivo que se perseguía con esta organización era ocultar los detalles de la implementación a través de una especificación bien definida de las entradas y salidas del módulo. El usuario del módulo debía tener suficiente información para poder usarlo, mientras que el implementador del módulo debería disponer de lo propio para construir el programa que satisficiese la especificación. En ningún caso, se debía proporcionar más información de la estrictamente necesaria. Parnas es el que utiliza por primera vez el término "ocultamiento de la información". La división de los módulos, que

tradicionalmente había sido dirigida por una descomposición funcional basada en pasos algorítmicos, adquirió entonces un nuevo enfoque: la descomposición modular basada en los datos. Se ofrecieron guías para seguir esta estrategia como, por ejemplo, agrupar en un único módulo cada estructura de datos: sus enlaces internos, operaciones de acceso y modificación, etc.

La descomposición modular basada en datos ofrecía diversas ventajas:

- Simplificación de las interfaces entre los módulos, de manera que los equipos de desarrollo podían distribuirse el trabajo y trabajar independientemente en cada parte.
- Minimización de los efectos y repercusiones debidos a la modificación de un módulo. Si los cambios no afectan a la interface ni a la especificación del módulo, los otros módulos no se ven afectados.
- Los módulos podían ser estudiados y comprendidos independientemente.

Varios lenguajes se crean a raíz de las ideas aportadas por los módulos y los TAD, entre los que podemos destacar CLU y Modula. Pero, a pesar de las notorias mejoras que había experimentado el proceso de desarrollo de software durante la década de los 70, los programas continuaban siendo poco fiables, incapaces de dar una buena respuesta a los cambios, poco eficientes, caros y los programadores incumplían los plazos inicialmente previstos. En palabras de Floyd, se había pasado de la "crisis del software" a la "depresión del software". Sin embargo, todos los avances llevados a cabo conducirían a la siguiente gran revolución del software: la programación orientada a objetos.

La programación orientada a objetos

El desarrollo de la programación orientada a objetos (POO) produjo una auténtica revolución, comparable con la que en su día ocurriera con la programación estructurada. La POO se erige actualmente como el principal paradigma para garantizar la calidad del software, fundamentalmente centrado en la mejora de la extensibilidad y la reutilización del código. Como bien expresó M. V. Wilkes en 1993 "*la programación orientada a objetos es el desarrollo más destacado acaecido en los lenguajes de programación desde hace mucho tiempo*".

Recogiendo e integrando muchos de los principios teóricos provenientes de la programación estructurada y modular, la clave de la programación orientada a objetos se basa en la definición de un mecanismo de clases con una **naturaleza dual**: una clase es, a la vez, un módulo –que ofrece encapsulación y ocultamiento de la información–, y un tipo abstracto de datos –del cual se pueden crear diferentes instancias–. Además, surge otro concepto fundamental: la herencia, que permite extender, ampliar o modificar el comportamiento de cierto tipo de objetos sin necesidad de afectar a la implementación de los usuarios del tipo.

Encapsulación, ocultamiento de información, herencia y ligadura dinámica son las propiedades fundamentales que caracterizan a la POO. La **encapsulación** permite agrupar bajo una simple interfaz el conjunto de operaciones aplicables sobre cierta abstracción de datos. El **ocultamiento** ofrece la posibilidad de decidir qué partes de la definición de la clase son accesibles desde fuera y cuáles no. La **herencia** permite crear una nueva abstracción como una extensión o refinamiento de alguna abstracción existente. La **ligadura dinámica** es un concepto vinculado a la herencia, que permite a la nueva abstracción mostrar su nuevo comportamiento, incluso al usarla en un contexto donde se espera la antigua abstracción.

El comienzo de la orientación a objetos se suele situar con la aparición del lenguaje SIMULA I (1962-65) y SIMULA 67. El lenguaje SIMULA 67 –desarrollado en el Centro de Computación Noruego, en Oslo, por Ole-Johan Dahl y Kristen Nygaard– ya introducía la mayoría de los conceptos básicos de

la programación orientada a objetos: objetos y clases, subclases (herencia) y procedimientos virtuales. Sin embargo, las clases de SIMULA no proporcionaban encapsulación, pudiendo acceder desde fuera a sus atributos y procedimientos internos. Sin duda alguna, el gran impulso de la POO tiene lugar a partir de los años 1970, con la creación del lenguaje Smalltalk, en 1972, y fundamentalmente en la siguiente década con la aparición de lenguajes como C++, en 1983, Object Pascal y Eiffel, en 1985, y más recientemente Java, en 1995.

Pero más que un simple conjunto de aportaciones conceptuales, la POO constituye un paradigma de programación propio y diferenciado del paradigma imperativo. De esta manera, la POO establece un nuevo modelo de ejecución, frente al modelo imperativo donde la clave es el flujo de ejecución de un programa y la instrucción básica es la asignación. El modelo orientado a objetos está compuesto por un mundo de objetos que se comunican entre sí, a través del paso de mensajes. En consecuencia, el nuevo modelo implica otra forma de *ver* y entender la programación.

Aparte del modelo de ejecución OO, los elementos más destacables del paradigma OO, según Bertrand Meyer –creador de Eiffel y uno de los grandes impulsores de la POO–, son:

- **Ocultación de la información**, que debería ser usada de manera estricta para garantizar la aplicación adecuada de los principios del diseño de librerías. Todos los datos miembros de una clase deberían ser privados, ofreciendo funciones de consulta y modificación en caso necesario. Esto permitiría cambiar la representación de la información sin necesidad de afectar a los usuarios.

- **Principio de acceso uniforme.** Relacionado con el punto anterior, este principio establece que un cliente que accede a una propiedad de una clase no debe conocer si esta se encuentra almacenada o se calcula bajo demanda.

- **Principio abierto-cerrado.** La separación entre interfaz e implementación y el mecanismo de herencia permiten la capacidad de usar un componente software tal como es (cerrado), pero con la posibilidad de extenderlo a través de la herencia manteniendo intactos a sus clientes preexistentes (abierto).

- **Uso sistemático de la herencia.** La herencia da lugar a una considerable cantidad de conceptos nuevos, como: polimorfismo, ligadura dinámica, renombramiento, redefinición, clases diferidas o abstractas –total o parcialmente–, genericidad restringida, herencia múltiple, interfaces, etc. Los programadores deben asimilar la filosofía subyacente en el mecanismo de herencia, evitando el extremo de caer en una *taxomanía*.

- **Diseño por contrato.** La programación por contrato es un concepto introducido por Meyer, para guiar el reparto de responsabilidades en el diseño e implementación de las clases. El enfoque tradicional –la llamada "programación defensiva"– aconsejaba construir rutinas *blindadas*, con tantas comprobaciones de las precondiciones como fueran posibles, aunque fueran redundantes con las realizadas por los clientes de la rutina. Frente a esto, Meyer propone la analogía de una especificación con un contrato: la especificación es un contrato que se realiza entre el cliente y el proveedor; el programa que invoca a una rutina actúa como cliente, mientras que la rutina juega el papel de proveedor. La rutina ofrece un servicio, especificado mediante una aserción en su **postcondición**, y exige ciertas condiciones, especificadas mediante una aserción en la **precondición**, para llevar a cabo tal servicio. La responsabilidad u obligación del proveedor se convierte en un beneficio para el cliente, mientras que las suposiciones del proveedor constituyen una obligación para el cliente. El reparto claro de responsabilidades mejora la fiabilidad del código, paradójicamente, reduciendo el número de comprobaciones.

- **Cuidar el diseño de las clases**, por ejemplo evitando la existencia de las llamadas "*clases Dios*", separando entre operaciones de consulta y modificación, y fomentando la simplicidad y legibilidad, fundamentales para ofrecer calidad.

En la actualidad, el modelo orientado a objetos ha alcanzado casi todas las áreas de la tecnología del software: metodologías de análisis y diseño, bases de datos, gráficos, enfoques formales, redes, concurrencia, programación distribuida, desarrollo Web, lenguajes de programación. No obstante, y aunque ha sido aplicada también al campo de los algoritmos, no está claro si el paradigma OO aporta realmente alguna mejora significativa en este ámbito particular.

A pesar de que son numerosos los lenguajes de programación OO existentes, cada uno presenta ciertas características que lo distinguen de los demás. Es difícil encontrar un lenguaje OO perfecto. Realmente, cada uno está mejor adaptado para abordar cierto tipo de aplicaciones: Java para algunos casos como aplicaciones Web; Eiffel para desarrollos de calidad y reutilización a largo plazo; C++ para mantener la compatibilidad con aplicaciones escritas en C; Smalltalk para la construcción de prototipos rápidos; Delphi para *Windows RAD* (*Rapid Application Development*).

Estado actual de la programación

En un ámbito tan amplio, diverso y dinámico como se ha llegado a convertir el mundo del desarrollo de software, acertar en encontrar las tendencias predominantes y con más futuro es una labor compleja. Dos corrientes se perfilan como las nuevas impulsoras del área: la **tecnología de componentes** y la **programación distribuida**. La tendencia actual en las industrias de software se dirige cada vez más hacia economías de escala, originadas a partir de la reutilización de programas. Aunque no hay una clara definición de componente aceptada universalmente, podemos establecer la siguiente definición siguiendo a Bertrand Meyer: un componente software es un elemento de programa con las siguientes características:

- Otros elementos de programa, clientes, podrían utilizar el componente. Por ejemplo, esto excluye a programas para uso exclusivamente humano.
- Los autores del componente no necesitan conocer a los clientes y sus autores. Por ejemplo, esta propiedad excluye a un módulo usado por otros módulos sin la característica de ser reutilizable.

Los componentes constituyen, por lo tanto, el nuevo elemento de reutilización. Realmente, la filosofía subyacente en los componentes es una idea antigua en el mundo del software: la posibilidad de construir programas uniendo partes prefabricadas, de la misma manera que una casa se construye juntando ladrillos, poniendo uno sobre otro. Meyer se pronuncia respecto a los componentes en los siguientes términos: "*los componentes deben tener un certificado de calidad, corroborado por la aplicación del diseño por contrato, junto con un riguroso proceso de evaluación a través de herramientas de validación y verificación y de un examen público meticuloso*".

Existen diversas clasificaciones de componentes, que no se ciñen exclusivamente a código. Bajo el paraguas de la tecnología de componentes podemos encontrar clases, patrones, *frameworks*, documentos de análisis, etc. Entre los actuales sistemas de componentes disponibles podemos señalar WindowsDNA/COM+, .NET, CORBA, VCL, JavaBeans y EJB (*Enterprise JavaBeans*). Relacionado muy estrechamente con el uso de componentes en entornos OO, la introducción de la programación distribuida se perfila como otra de las claves del futuro de la programación. El modelo de ejecución OO, compuesto por un universo de objetos que se comunican entre sí, adquiere una nueva dimensión cuando estos objetos puedan residir en máquinas distintas, en diferentes países y continentes, y la comunicación se realiza a través de Internet. Esta es la idea básica en herramientas ya existentes, aunque aún no muy extendidas, como .NET y CORBA.

1.2.2. Los principales avances en algoritmos y estructuras de datos

En comparación con la historia de la programación, la evolución de los algoritmos y las estructuras de datos se presenta de forma más fraccionada y discontinua, marcada por la aparición de pequeños y numerosos *hitos* que han transcendido en el progreso de la disciplina. Por este motivo, presentamos la historia de los AED señalando sobre una escala temporal la aparición de los principales avances en la disciplina. Esta enumeración no pretende ser un repaso de la historia de la informática; tampoco de la historia de la programación, aunque en muchas ocasiones los protagonistas de ambos ámbitos son las mismas personas. Nos fijamos exclusivamente en los avances relacionados directamente con el análisis y diseño de algoritmos y estructuras de datos, omitiendo todos los hitos relativos al hardware o a la aparición de nuevos lenguajes o métodos de programación.

Grosso modo, podemos distinguir tres grandes periodos en la historia de la disciplina de AED:

- **La pre-historia.** Este periodo abarcaría desde los primeros algoritmos de los tiempos de la antigua Babilonia hasta la aparición de los primeros ordenadores. En esta fase tienen lugar numerosos avances de índole teórico, que preparan el camino y las bases de la disciplina. Los algoritmos están principalmente orientados a la resolución de problemas matemáticos, de aritmética y geometría, y pensados para ser ejecutados "a mano". Se establecen también los fundamentos de la teoría de grafos, de árboles, etc.

- **El periodo clásico.** Durante los años comprendidos entre 1950 y 1975 tiene lugar un desarrollo espectacular de la algorítmica, que empieza a ser reconocida como una disciplina independiente. Se inventan los principales algoritmos de ordenación eficientes, estructuras de datos y técnicas de diseño de algoritmos, que siguen siendo hoy día los puntos clave de la materia. No hay una clara separación de fases temporales centradas en objetos de estudio distintos, sino que los avances en un mismo tema –como por ejemplo grafos, análisis de algoritmos o programación dinámica– tienen lugar a lo largo de un amplio periodo de tiempo. Son muchos los autores que aportan sus contribuciones, entre los que sería difícil destacar alguno por encima de otro.

- **La historia moderna.** Este último periodo, que llega desde mediados de la década de 1970 hasta nuestros días, se caracteriza por el asentamiento de la disciplina. Alrededor de 1980 aparecen numerosos libros docentes en los que se perfila el estado actual de la disciplina. Para bien o para mal, la época de las grandes ideas revolucionarias y de uso general ha acabado. Predominan más bien los trabajos centrados en dominios específicos de aplicación, reiventando y adaptando a los clásicos. En cierto sentido se puede decir que la disciplina se desmiembra: los nuevos algoritmos y estructuras de datos tienen lugar en el ámbito de las bases de datos, la inteligencia artificial, la criptografía, las redes, los gráficos, la visión artificial, etc. Todos tienen un origen común en la algorítmica, pero han adoptado ya sus propios métodos y objetivos.

A continuación vamos a destacar los que consideramos que son los principales avances en la disciplina de los algoritmos y las estructuras de datos. Nos centramos básicamente en lo que hemos denominado "el periodo clásico", que contiene la mayoría de los fundamentos del actual estado de la materia.

2000 a.C. (aprox.)

- Surgen en Babilonia los primeros algoritmos de cálculo numérico, dedicados a la resolución de problemas como la suma, resta, multiplicación y raíces cuadradas.

300 a.C. (aprox.)

- Euclides de Alejandría propone el algoritmo para el cálculo de máximo común divisor de dos números, conocido como el algoritmo de Euclides, en el que aparece por primera vez la idea del salto condicional.

825 d.C. (aprox.)

- Muhammad ibn Musa al-Khwarizmi escribe el libro "Hisab al-jabr w'al-muqabala" ("El cálculo de reducción y restauración") que contiene numerosos algoritmos para la resolución de ecuaciones de primer y segundo grado.

1200 d.C. (aprox.)

- Leonardo de Fibonacci (Leonardo de Pisa) traduce la obra de al-Khwarizmi, introduciendo en Europa el sistema hindú de numeración posicional y los métodos de resolución de problemas basados en la aplicación de una serie de pasos.

1736

- Leonhard Euler propone y resuelve el problema conocido como "los puentes de Königsberg", lo cual se considera como el punto de partida de la teoría de grafos.

1847

- Aparece por primera vez el concepto de árbol como una entidad matemática definida formalmente, en un trabajo del alemán G. Kirchhoff (*Anales de física y química*, 1847). Los árboles son usados para encontrar ciclos en circuitos eléctricos. El nombre de árbol, así como varios resultados relacionados, aparecerían unos diez años más tarde, en una serie de artículos debidos a Arthur Cayley.

1848

- El problema de las ocho reinas es inventado por Bezzel.

1857

- Sir William R. Hamilton plantea el problema conocido como del ciclo hamiltoniano.

1883

- El matemático francés Édouard Lucas inventa un juguete conocido como Las Torres de Hanoi, consistente en tres palos sobre los que se pueden insertar discos de varios tamaños.

1894

- El matemático alemán Paul G.H. Bachmann inventa la notación asintótica O.

1926

- O. Borûvka propone el primer algoritmo para el problema de los árboles de expansión de coste mínimo en un grafo no dirigido con pesos.

1930

- V. Jarník inventa un algoritmo para resolver el problema de los árboles de expansión de coste mínimo, que sería redescubierto por R.C. Prim casi treinta años después.

1935

- H. Whitney desarrolla la conocida como teoría de *matroides*, que sería la base para el desarrollo de algoritmos voraces óptimos.

1942

- G.C. Danielson y C. Lanczos publican el primer artículo donde se propone un método eficiente para calcular las transformadas discretas de Fourier.

1945

- Konrad Zuse desarrolla un conjunto de algoritmos de manipulación de listas, con listas almacenadas en posiciones consecutivas de memoria y cuya longitud podía variar dinámicamente.
- A.M. Turing desarrolla un mecanismo –conocido como "*Reversion Storage*"– que sería el precursor de las pilas. Los valores eran almacenados en posiciones consecutivas. Otros mecanismos similares eran usados durante estos primeros años.
- J.W. Mauchly, J.P. Eckert y J.v. Neumann escriben el "*First Draft*", documento que describe los fundamentos de los programas almacenados, con numerosas técnicas que se usan aún hoy día, como la ordenación por mezcla o *mergesort*.

1946

- John von Neumann escribe uno de los primeros tratados sobre algoritmos básicos de recorrido sobre arrays y listas almacenadas en posiciones consecutivas de memoria.
- J.W. Mauchly propone el llamado "direccionamiento uno-más-uno", donde se intuye por primera vez la idea de las listas enlazadas, aunque aún en estado embrionario. Las técnicas de inserción y eliminación dinámica eran aún desconocidas.

1947

- G. Dantzig inventa el algoritmo simplex de programación lineal.

1948

- M. Hall estudia el problema de pareamiento de grafos, también conocido como problema de asignación.

1949

- I.N. Metropolis y S. Ulam introducen por primera vez el término "Monte Carlo" para referirse a los algoritmos probabilistas de cálculo numérico.

1951

- G.M. Hopper utiliza árboles binarios para representar fórmulas algebraicas en el desarrollo del lenguaje A-1.

1952

- D.A. Huffman desarrolla un algoritmo para la codificación de información con mínima redundancia, conocido como la técnica de los códigos de Huffman.

1953

- H.P. Luhn sugiere el uso de "encadenamiento" para la búsqueda externa, lo que supone una de las primeras apariciones de las listas enlazadas.

1955

- Los conceptos de pila y cola –listas "LIFO" y "FIFO"– y su utilidad, aparecen en algunos tratados de contabilidad.

- A. Newell, J.C. Shaw y H.A. Simon usan por primera vez, en el desarrollo de IPL-III, pilas con implementaciones enlazadas y con operaciones "*push down*" y "*pop up*".

- W.W. Peterson inventa la técnica de representación de conjuntos mediante tablas de dispersión.

- R.E. Bellman empieza una serie de estudios sistemáticos sobre una nueva técnica de diseño de algoritmos conocida como programación dinámica.

1956

- A. Newell, J.C. Shaw y H.A. Simon desarrollan, en sus investigaciones sobre resolución heurística de problemas, las técnicas de estructuras enlazadas en memoria, y crean el primer lenguaje de procesamiento de listas, IPL-II.

- J.B. Kruskal propone un algoritmo para la obtención del árbol de expansión de coste mínimo, conocido como el algoritmo de Kruskal.

- S.C. Kleene propone un algoritmo para determinar la expresión regular asociada a un autómata finito, que sería la base de los algoritmos de Floyd y Warshall.

- J.H. Curtiss y C.W. Vickery proponen diversos algoritmos numéricos probabilistas para resolver problemas de álgebra lineal.

1957

- A. Newell, J.C. Shaw y H.A. Simon usan por primera vez, en el desarrollo de IPL-III, pilas con implementaciones enlazadas y con operaciones "*push down*" y "*pop up*".

- R.C. Prim publica un algoritmo para calcular el árbol de expansión de coste mínimo de un grafo, conocido como el algoritmo de Prim, aunque ya había sido propuesto 27 años antes.

- R.E. Bellman publica su primer libro sobre programación dinámica, dotando al área de una sólida base matemática.

1958

- Claude Berge escribe uno de los primeros libros sobre teoría de árboles y grafos, presentados desde un punto de vista matemático.

1959

- J.W. Carr escribe un artículo describiendo la utilidad de las estructuras enlazadas en memoria para los problemas más habituales, sin requerir subrutinas muy sofisticadas.

- E.F. Moore descubre la búsqueda primero en anchura en el contexto de un problema de caminos mínimos en un laberinto.

- E.W. Dijkstra propone un algoritmo para la obtención de los caminos más cortos en un grafo desde un nodo origen hasta todos los demás, conocido como el algoritmo de Dijkstra.

- D.L. Shell presenta el algoritmo de ordenación conocido como *shellsort*.

- E.N. Gilbert y E.F. Moore resuelven el problema de los árboles binarios de búsqueda óptimos, usando un algoritmo de programación dinámica.

1960

- A.J. Perlis y C. Thornton proponen en un artículo la representación de árboles mediante estructuras enlazadas. El artículo introduce también los conceptos de recorrido del árbol en varios órdenes, y árbol hilvanado. Los árboles usan una representación "hijo izquierdo-hermano derecho".

- E. Fredkin presenta la estructura de árboles trie, basados en la representación de prefijos.

1961

- D.T. Ross propone por primera vez en un artículo la utilización de listas con enlaces múltiples, aunque la idea había aparecido también en otros grupos.

- N.G. de Bruijn establece los métodos para el análisis asintótico de algoritmos.

1962

- Se populariza el uso de listas circulares y doblemente enlazadas, gracias a la existencia de sistemas de procesamiento de listas.
- H. Hellerman publica por primera vez un estudio sobre técnicas de direccionamiento y recorrido de arrays multidimensionales, aunque tales métodos ya eran usados previamente.
- A.A. Karatsuba e Y. Ofman proponen el algoritmo de multiplicación rápida de enteros largos, de orden $O(n^{1,59})$, usando la técnica de divide y vencerás.
- G.M. Adel'son-Vel'skiĭ y E.M. Landis proponen la estructura de árboles binarios de búsqueda balanceados, conocidos como árboles AVL.
- S. Warshall presenta un algoritmo para calcular el cierre transitivo de un grafo, en un tiempo $O(n^3)$, y basado en programación dinámica.
- R.W. Floyd diseña un algoritmo para calcular los caminos mínimos entre todos los nodos de un grafo, en un tiempo $O(n^3)$, conocido como el algoritmo de Floyd, basado en un teorema presentado por Warshall ese mismo año.
- R.W. Ford y L.R. Fulkerson publican un libro sobre problemas y algoritmos de flujo en redes, que da origen al estudio formal de problemas en este área.
- C.A.R. Hoare propone el método de ordenación rápida o *quicksort*.
- M. Held y R. Karp proponen un algoritmo para el problema del viajante en un tiempo $O(n^2 2^n)$, usando la técnica de programación dinámica.

1963

- J. Dunlap, de Digitek Corporation, desarrolla técnicas que permiten usar listas lineales de longitud variable compartiendo las mismas posiciones de memoria, mediante el desplazamiento de las mismas. La misma idea apareció de manera independiente en el diseño de un compilador COBOL en IBM.
- A.W. Holt crea la estructura de representación de árboles binarios mediante punteros "hijo izquierdo" e "hijo derecho".
- Se desarrollan varios sistemas de manipulación de cadenas, tratando la representación de información alfabética con cadenas de longitud variable.

1964

- E.S. Schwartz propone un algoritmo voraz para derivar códigos de Huffman óptimos.
- J.W. Williams presenta la estructura de datos de montículos, aplicándola en la creación de un nuevo método de ordenación conocido como *heapsort*, mejorado después por R.W. Floyd.

1965

- J.M. Cooley y J.W. Tukey descubren el algoritmo rápido para obtener la transformada de Fourier (FFT), lo que supondría una revolución en muchos campos de la ciencia. Problemas que se consideraban inatacables ahora podían ser resueltos.
- S.W. Golomb y L.D. Baumert describen por primera vez la técnica de *backtracking*.

1966

- E.L. Lawler y D.W. Wood inventan la técnica de diseño de algoritmos de ramificación y poda.

1967

- F.S. Hillier y G.J. Lieberman proponen un algoritmo de ramificación y poda para el problema de la asignación.

1968

- R. Morris publica un artículo tratando la técnica de las tablas de dispersión.
- I.J. Good desarrolla uno de los primeros programas para jugar al ajedrez.
- M. Bellmore y G. Nemhauser proponen una resolución para el problema del viajante usando ramificación y poda.
- D.E. Knuth publica uno de los primeros y más relevantes trabajos en el área de los algoritmos y las estructuras de datos. Entre otras contribuciones preconiza la importancia del análisis de algoritmos y ofrece un algoritmo de orden lineal para el cálculo del orden topológico de un grafo.

1969

- V. Strassen propone un algoritmo de multiplicación rápida de matrices, con un orden $O(n^{2,81})$, usando la técnica de divide y vencerás.

1970

- M.D. McIlroy R. Morris diseñan la estructura de representación de relaciones de equivalencia mediante punteros al padre, con balanceo del árbol y compresión de caminos.
- J.E. Hopcroft inventa la estructura de árboles 2-3, que sería un precursor de los árboles B.

1971

- A. Schonhage y V. Strassen crean un algoritmo de multiplicación de enteros largos, con un orden $O(n \log n \log \log n)$, usando divide y vencerás.
- Se propone la representación de listas múltiples, u ortogonales, para la implementación de la base de datos DBTG.
- J. Edmonds introduce y formaliza la noción de algoritmo voraz.

1972

- R. Bayer y E.M. McCreight introducen la estructura de árboles B, para la organización y mantenimiento de índices de gran tamaño.
- R.E. Tarjan y otros autores popularizan la búsqueda primero en profundidad para la resolución de problemas sobre grafos, como por ejemplo la búsqueda de puntos de articulación.

1973

- J.E. Hopcroft y R.E. Tarjan proponen el uso de listas de adyacencia para representar matrices de adyacencia en grafos *escasos*, es decir, con pocas aristas.
- S.S. Godbole diseña un algoritmo para resolver el problema de la multiplicación encadenada de matrices usando programación dinámica.
- D.S. Johnson desarrolla en su tesis doctoral uno de los primeros algoritmos de aproximación, aplicado al problema de empaquetamiento en recipientes.

1974

- S. Lin y B.W. Kernighan describen una técnica heurística eficiente para el problema del viajante.

1976

- N. Christofides da una solución para el problema del viajante métrico usando un algoritmo de aproximación.

1977

- D.E. Knuth, V.R. Pratt y J.H. Morris inventan el algoritmo de búsqueda de cadenas que lleva el nombre de algoritmo de Knuth-Morris-Pratt.
- R.L. Rivest, A. Shamir y L.M. Adleman proponen el algoritmo RSA, uno de los primeros y más consistentes métodos de encriptación conocidos actualmente.
- Por primera vez un ordenador derrota al campeón mundial de *backgammon*.

1978

- L.J. Guibas y R. Sedgewick proponen la estructura de árboles rojinegros.
- J. Vuillemin propone la idea de los montículos binomiales.
- R. Kosaraju desarrolla un algoritmo eficiente para calcular los componentes fuertemente conexos de un grafo dirigido.

1979

- J.L. Carter y M.N. Wegman introducen la noción de clases universales de funciones de dispersión.

1984

- M.L. Fredman, J. Komlós y E. Szemerédi desarrollan el esquema de dispersión para conjuntos estáticos conocido como dispersión perfecta (*perfect hashing*).

1985

- G. Brassard propone las notaciones asintóticas condicionales y otras ideas sobre el análisis de algoritmos.

1997

- Por primera vez un ordenador, Deep Blue de IBM, derrota al campeón mundial de ajedrez, Garry Kasparov.

2003

- Se completa el mapa del genoma humano, gracias a la ayuda de nuevos ordenadores y algoritmos.

1.3. Los principios básicos de la materia

Los desarrollos propios de la algoritmia y las estructuras de datos tienen lugar en un amplio y variado espectro de ámbitos de aplicación. Desde el principio, las investigaciones han ido encaminadas hacia direcciones muy distintas, lo que ha provocado finalmente una separación de la materia en diferentes ramas. Pero, sea como fuere, existe una serie de principios subyacentes que se encuentran en la base de la materia como disciplina científica. Ya estemos enfrentándonos al diseño de una estructura de datos, analizando un algoritmo o abordando la resolución de un problema computacional, los principios constituyen los pilares que guían la manera metódica de afrontar el problema. Podemos distinguir tres grandes principios: la abstracción, la formalización y la eficiencia. Desde nuestro punto de vista, estos son los grandes ejes en torno a los cuales se mueve y se moverá la disciplina de los algoritmos y las estructuras de datos.

1.3.1. Abstracción

La abstracción es el mecanismo fundamental para abordar la complejidad del mundo real. Los conceptos abstractos protegen al programador de los intrincados mecanismos internos de un ordenador, de los innumerables factores que concurren en los problemas de la vida real, y del errático flujo de ejecución que implica la ejecución de cualquier programa a bajo nivel. En muchos aspectos, el avance de la disciplina de AED se puede entender como una lucha por conseguir mayores niveles de abstracción, ya sea en los lenguajes de programación, en las técnicas de diseño de algoritmos, o en los tipos y estructuras de datos. Sólo cuando aparecieron los lenguajes de alto nivel, que abstraían a los programadores de la complejidad del código máquina, pudo tener lugar la eclosión definitiva de la disciplina.

Abstraer significa eliminar lo superfluo para quedarse con lo relevante, lo importante, la esencia de las cosas. Por ello, la abstracción es una capacidad inherente a la inteligencia humana. Pero ¿qué es lo importante? Los distintos grados de relevancia subjetiva –esto es, desde el punto de vista del humano– dan lugar a distintos niveles de abstracción: según el nivel de detalle, decimos que estamos en un nivel de abstracción mayor o menor.

En programación, las abstracciones surgen normalmente al establecer que lo importante es *lo que hace* el concepto abstracto, y lo superfluo es *cómo lo hace*. Es lo que llamamos **abstracción por especificación**. Por otro lado, las abstracciones adquieren una nueva dimensión al aplicar un mecanismo de parametrización, en función del cual el significado o comportamiento del concepto abstracto no es fijo sino que se adapta en función de unos parámetros con significado concreto. Esto se denomina **abstracción por parametrización**. La abstracción por parametrización permite la creación de entidades genéricas, es decir, que pueden ser usadas en un amplio conjunto de ámbitos distintos.

Prácticamente todas las áreas de la disciplina de los algoritmos y las estructuras de datos se basan en la aplicación de mecanismos de abstracción:

- **Tipos abstractos de datos.** Posiblemente, uno de los mayores impulsores en el ámbito de los lenguajes de programación ha sido el interés por introducir en los lenguajes mecanismos de abstracción de datos, cada vez más avanzados y próximos al concepto teórico de TAD. De esta forma se puede entender la evolución que se ha producido desde los tipos definidos por el usuario y la programación modular, hasta la programación orientada a objetos.

  Ya vimos en el apartado 1.1 la definición de TAD. Una interesante analogía de los TAD es la que los compara con los muebles de una casa, [García'03]:

  > *Un lenguaje que no permite usar TAD –o un entorno de programación para el que no disponemos de TAD– es como una casa sin muebles y sin decoración. Es posible vivir en ella y realizar tareas rutinarias, aunque tendríamos que hacerlo de manera muy rudimentaria. Por ejemplo, deberíamos dormir en el suelo o beber agua chupando del grifo. Los TAD son como los muebles de la casa, no son estrictamente necesarios para vivir pero facilitan mucho las cosas. Los muebles, que hacen el papel de tipos abstractos:*
  >
  > - *Aportan una funcionalidad extra fundamental, un valor añadido a la casa, permitiendo realizar las tareas comunes de manera más sencilla y cómoda.*
  > - *Pueden cambiarse por otros, moverse de sitio, arreglarlos en caso de rotura, etc., lo cual no es posible hacerlo –o resulta mucho más costoso– con la casa en sí.*
  > - *Los muebles y la decoración están adaptados a las necesidades, gustos y caprichos de cada uno, mientras que la casa es la misma para todo el bloque de apartamentos.*
  > - *Hay una separación clara entre la persona que fabrica el mueble y la que lo utiliza. Para usar un mueble no es necesario conocer nada de carpintería.*

  El comienzo de los TAD se puede datar con la aparición de la teoría de grafos, de árboles, de colas, etc. A pesar de los avances que han tenido lugar desde entonces, aún existe un margen para la mejora del mecanismo de TAD en los lenguajes. Por ejemplo, actualmente son muy pocos los que permiten la definición de TAD genéricos o parametrizados.

- **Abstracciones operacionales.** Históricamente, las abstracciones operacionales –funcionales y procedurales– se encuentran también entre las precursoras en la disciplina. El primer paso trascendental lo constituyó la introducción de las rutinas en los lenguajes de programación. La rutina es la abstracción de un comportamiento, del que se sabe su efecto pero no el modo de obtenerlo. Permiten factorizar código y evitan repeticiones innecesarias. Pero la rutina, en su significado originario, era un simple salto con retorno.

El avance cualitativo se produce con la aparición de los mecanismos de definición de funciones y procedimientos. Una función, o un procedimiento, abstraen un fragmento de código, más o menos largo, al cual se le asigna un nombre significativo. La especificación de una operación determina cuáles son los parámetros de entrada, los de salida y el efecto que tendrá su ejecución, pero no se indica nada sobre la implementación. Para el usuario de la operación es indiferente cómo se haya implementado, siempre que se garantice el efecto descrito en la especificación.

De la misma forma que los TAD son una generalización de los tipos predefinidos en un lenguaje, las funciones generalizan la idea de operador de un lenguaje de programación, permitiendo que el usuario se defina sus propios operadores. A su vez, los procedimientos son generalizaciones de las instrucciones del lenguaje. Prácticamente todos los lenguajes usados hoy en día, excepto el ensamblador, incluyen la posibilidad de definir funciones y procedimientos. Pocos, sin embargo, permiten la definición de operaciones genéricas.

- **Abstracciones de iteradores.** Aunque poco frecuentes, los iteradores constituyen otro tipo básico de abstracción en programación. Se pueden considerar como una generalización de los iteradores elementales de los lenguajes de programación. Por ejemplo, un iterador: "for i:= 1 to n", proporciona de forma abstracta un recorrido de todos los enteros entre 1 y n. De manera similar, podríamos tener un iterador for que recorriera todos los elementos de un conjunto, de un árbol o de una lista. Un iterador permite recorrer los elementos de una colección –un TAD que contiene elementos de otro tipo– de forma abstracta. En un iterador lo relevante es que se listan todos los elementos de una colección, independientemente de cuál sea la forma de almacenar y estructurar los valores del tipo.

  A pesar de su reconocida importancia –y a pesar de que la mayoría de los problemas de programación se reducen a un número reducido de esquemas de recorrido–, no existen normalmente en los lenguajes mecanismos explícitos para la definición y uso de iteradores. Es el programador quien, usando los posibilidades de las que dispone, debe definirse sus métodos de iteración. Existe, por lo tanto, un amplio margen de mejora en este sentido.

- **Abtracciones de librerías.** Las librerías o paquetes ofrecen un mecanismo de abstracción de nivel superior, que podríamos denominar "abstracción de funcionalidades". Una librería agrupa bajo un nombre significativo un conjunto de tipos, variables y operaciones que aportan una funcionalidad de interés en cierto ámbito de aplicación. Por ejemplo, una librería denominada "*impresora*" contendría todas las utilidades relacionadas con el manejo de impresoras. Este tipo de abstracciones aparece en las fases de análisis y diseño de aplicaciones, cuando se trata de crear sistemas de cierto tamaño.

- **Esquemas algorítmicos.** Un esquema algorítmico se puede considerar como la abstracción de la filosofía que subyace a un conjunto de algoritmos, y surge tras la aplicación de un proceso de generalización. En el esquema algorítmico lo importante es la estructura, la interpretación genérica que se hace del problema. Idealmente, el esquema sería como un "algoritmo con huecos", correspondientes a los elementos genéricos (funciones y tipos de datos). El programador simplemente debería insertar el código y los tipos adecuados en los huecos que correspondan, y ya tendría un algoritmo que resuelve el problema.

  En la práctica, esta idea de construir algoritmos "rellenando huecos de un esquema" no suele ser tan simple, por varios motivos. En primer lugar, la variedad de situaciones que nos podemos encontrar en distintos problemas es, normalmente, mucho mayor que la prevista en los esquemas de algoritmos. Pero, por otro lado, en esquemas excesivamente flexibles el problema suele ser la

ineficiencia; un algoritmo escrito desde cero para un problema particular puede incluir código optimizado para ese caso dado. De todas formas, los esquemas algorítmicos son una ayuda muy útil en la construcción de algoritmos, por lo menos para la obtención de una primera versión.

- **Diseño mediante abstracciones.** Si ya vimos que el objetivo de la disciplina de AED es la resolución de problemas computacionales, el modelado de los problemas constituye el primer paso necesario para alcanzar la solución. Evidentemente, el modelo surge por la aplicación de una abstracción sobre un problema de interés de la vida real, en la que descartamos todos aquellos aspectos irrelevantes que no influirán en el cálculo de la solución. La obtención del modelo abstracto es la primera etapa de un proceso que, de una u otra manera, se fundamenta en el uso de abstracciones. Podríamos distinguir los siguientes pasos en el proceso de diseño mediante abstracciones:

  1. Identificar los subproblemas en los que se divide el problema original.
  2. Especificar cada subproblema de forma abstracta, usando alguna notación.
  3. Resolver cada subproblema de forma separada.
  4. Unir las soluciones y verificar la solución para el problema original.

  Existe una clara analogía con el método científico –aplicado en las ciencias experimentales– compuesto por los pasos de: observación, proposición de hipótesis, experimentación y verificación de la hipótesis. El proceso se puede ver, por lo tanto, como una aplicación del método científico a la disciplina de la programación. En nuestro caso, las abstracciones de objetos del mundo real juegan el papel de hipótesis que deben ser especificadas, implementadas y verificadas.

- **Notaciones asintóticas.** También en el ámbito del análisis de algoritmos se usan mecanismos de abstracción. De hecho, las notaciones asintóticas de complejidad son lo que podríamos denominar "abstracciones por simplificación". El análisis de los recursos consumidos por un algoritmo puede producir fórmulas largas y complejas, con muchos factores que intervienen, algunos de ellos sólo para tamaños pequeños. El uso de una notación asintótica simplifica la fórmula original, dándonos una idea abstracta del crecimiento de la función de interés. Por ejemplo, dado un $t(n) = 2n^2+7{,}2n\log_3 n-21\pi n+2\sqrt{n+3}$, podemos quedarnos simplemente con que $t(n) \in \Theta(n^2)$, o también $t(n) \in o(2n^2)$.

  Aunque aplicado en un contexto completamente distinto, se trata del mismo principio de abstracción: eliminar lo irrelevante para quedarse con lo importante. En este caso lo importante es el crecimiento asintótico de las funciones, es decir, para tamaños del problema suficientemente grandes. Lo irrelevante, lo que se descarta, son los términos que sólo influyen para tamaños pequeños. Las notaciones asintóticas están ampliamente extendidas; permiten una rápida comprensión del comportamiento de los algoritmos y facilitan la comparación entre distintas alternativas.

En definitiva, la abstracción es uno de los principios básicos de la disciplina, uno de los que han dado lugar a mayores progresos y que, sin duda, inspirará en el futuro nuevas vías para abordar la complejidad creciente de los problemas y las aplicaciones.

1.3.2. Formalización

Si bien la abstracción constituye un pilar central en la materia de estudio, sus efectos no serían plenamente productivos si los modelos abstractos y el modo de abordarlos no se vieran sometidos al

rigor de una formalización matemática precisa. Por ejemplo, no se puede reclamar haber solucionado un problema de optimización por el simple hecho de haber aplicado un esquema algorítmico; garantizar que un algoritmo obtiene la solución óptima requiere un proceso más próximo a una demostración matemática que a la aplicación de la intuición. La formalización matemática puede introducir una complejidad añadida en muchos problemas, pero al mismo tiempo es el elemento que da base científica a la disciplina, y por lo tanto, la consideramos un principio fundamental.

No es casualidad que los pioneros de la disciplina de la algoritmia fueran matemáticos, en el intento de resolver problemas aritméticos y geométricos. Cualquier otra actividad artesanal podría haberse atribuido la aplicación de una serie de pasos sistemáticos, pero sólo bajo el rigor matemático fue posible alcanzar la categoría de algoritmo. Igual que con la abstracción, como principio básico que es, la formalización se encuentra en todos los ámbitos de la materia. Podemos señalar los siguientes aspectos:

- **TAD como modelos matemáticos.** Existe un conjunto pequeño y bien conocido de TAD que aparecen de forma recurrente en los problemas de programación: conjuntos, listas, colas, pilas, árboles, grafos, tablas; además de los tipos numéricos elementales. Aunque estamos acostumbrados a verlos como conceptos intuitivos, todos ellos se basan en modelos matemáticos precisos, soportados por la teoría correspondiente. La teoría de conjuntos, de árboles, de grafos, etc., fueron establecidas mucho antes del comienzo de la programación, como ya vimos en el apartado 1.2.2. Estas teorías son las que aportan el criterio para la abstracción: lo importante es el modelo formal, y lo irrelevante es la forma de implementarlo. Adicionalmente, cualquier avance teórico en los modelos formales podrá dar lugar a una mejora en los métodos algorítmicos relacionados.

- **Especificación formal de TAD y operaciones.** Las especificaciones formales constituyen una de las áreas con más tradición, y al mismo tiempo con más futuro, dentro de la disciplina de AED. Si ya hemos comentado que los TAD se basan en modelos matemáticos, estos modelos se pueden describir rigurosamente a través de especificaciones usando notaciones formales. Además de la importante mejora que supone tener una especificación precisa de las operaciones y tipos de datos, podemos señalar otras ventajas de usar una notación formal:

  - **Prototipado.** Al usar una descripción rigurosa, las especificaciones formales pueden ser procesadas por un ordenador, para simular la ejecución de un procedimiento o el uso de un TAD. De esta manera podríamos tener un prototipo de programa, antes de haberlo implementado, generado automáticamente a partir de las especificaciones.
  - **Corrección de programas.** La especificación formal se puede usar para comprobar que la ejecución de un programa cumple el comportamiento deseado. Dado un programa y una especificación del mismo, sería posible introducir comprobaciones que garanticen que cada procedimiento verifica lo que determina su especificación.
  - **Reutilización.** Una especificación formal descrita de forma genérica se puede reutilizar en distintos ámbitos, de manera que en cada problema no debemos partir de cero. Esto también es posible con la notación informal, aunque suele ser menos frecuente, debido a que las especificaciones informales normalmente están orientadas a aplicaciones concretas.

- **Corrección formal de programas.** Los trabajos en este ámbito intentan introducir métodos que garanticen, de forma rigurosa, el funcionamiento correcto de los programas. Las primeras referencias que contemplan la demostración de la corrección de los programas datan de inicios de la década de 1960, [Fernández'02]. Ya en 1961 J. McCarthy propugnaba la demostración de propiedades en lugar de depurar los programas mediante la aplicación de pruebas. En 1966, Peter

Naur insistió en la importancia de las demostraciones de programas y propuso una técnica informal para especificarlas. Un año más tarde, Robert Floyd sugirió que las técnicas de demostración proporcionan una definición formal adecuada de los lenguajes de programación.

Pero el gran hito en la corrección de programas tiene lugar en 1969, cuando C.A.R. Hoare definió un pequeño lenguaje de programación en términos de un sistema lógico de axiomas y reglas de inferencia para demostrar la corrección de propiedades de programas, aplicando un razonamiento deductivo. El proceso de deducción de la corrección de un programa consiste en partir de la precondición y, aplicando los axiomas y las reglas de inferencia asociadas a las instrucciones del lenguaje de programación, llegar a conseguir la postcondición. Hoare predecía que la demostración de programas aliviaría tres de los grandes problemas del software: la falta de fiabilidad, la inexistencia de métodos adecuados de documentación, y la compatibilidad. Aunque a día de hoy no se han cumplido las expectativas, algunas de las ideas que surgieron en esta corriente –como, por ejemplo, las aserciones y el invariante de bucle– propiciaron el desarrollo de importantes avances en la disciplina.

- **Demostraciones con esquemas algorítmicos.** Ya hemos visto que la corrección formal de programas sufrió los inevitables efectos de la complejidad inherente al problema. Sin embargo, es posible obtener resultados relevantes al reducir el ámbito de los programas a estudiar. En particular, cuando los algoritmos han sido diseñados aplicando un esquema algorítmico concreto existe la esperanza de poder realizar, con más o menos facilidad, demostraciones de que el algoritmo encuentra la solución óptima o la correcta –si de hecho lo hace–. De esta forma, si en la práctica la corrección formal de programas en general no suele usarse, la verificación de optimalidad en algoritmos basados en patrones algorítmicos sí que suelen encontrarse con más frecuencia en los trabajos de la disciplina.

  El tipo de demostración aplicable depende, normalmente, del esquema algorítmico. O, dicho de otro modo, cada esquema algorítmico establece la forma de comprobar la corrección de los algoritmos creados con el mismo. A modo de ejemplo, podemos mencionar los siguientes:

  - **Divide y vencerás.** Un algoritmo de divide y vencerás obtendrá la solución correcta si se puede demostrar la validez de las funciones básicas de su esquema, es decir, si el método de resolución directo (el aplicado en los casos base) es correcto, y si la solución de un problema se puede obtener a partir de soluciones de subproblemas del mismo.
  - **Algoritmos voraces.** La validez se realiza estudiando las funciones de selección. Las demostraciones suelen realizarse normalmente por reducción al absurdo, suponiendo que los candidatos fueran seleccionados en un orden distinto. En otros casos, como en el algoritmo de Dijkstra, la demostración se hace por análisis de casos.
  - **Programación dinámica.** Los algoritmos de programación dinámica se basan en el principio de optimalidad de Bellman, que cumple un problema si cualquier solución óptima se basa en combinar soluciones óptimas de subproblemas. Un algoritmo producirá soluciones óptimas si usa una descomposición que garantice que se cumple este principio.

  Debemos señalar en este aspecto que los algoritmos mencionados en el apartado 1.2.2 supusieron un avance significativo en cuanto que se podía demostrar formalmente su funcionamiento correcto.

- **Análisis de algoritmos.** El análisis de algoritmos constituye uno de los ámbitos de la disciplina en los que más peso adquieren las herramientas matemáticas. Dado un algoritmo o una

estructura de datos, el objetivo es obtener una función que indique el consumo de recursos en relación al tamaño del problema. Obviamente, este proceso se basa en la aplicación de técnicas matemáticas: sumatorios, series, integrales, límites, probabilidades, etc. La formalización en este caso viene impuesta por la propia naturaleza del objeto de estudio, las funciones matemáticas. En muchos casos el análisis requiere del uso de técnicas no triviales; ciertos algoritmos sólo pudieron ser analizados años después de ser propuestos, como ocurre con el algoritmo de Euler y las estructuras de representación de relaciones de equivalencia mediante punteros al padre, cuyo tiempo está relacionado con la denominada función de Ackerman.

El análisis puede realizarse también de forma experimental, y también ahí será necesario el rigor matemático. En este caso se aplican instrumentos estadísticos: representaciones gráficas, interpolación, ajustes de regresión, análisis de residuales, etc.

- **Razonamiento inductivo.** Otra de las muestras más evidentes del carácter fundamental de la formalización matemática en la disciplina es la aplicación del razonamiento inductivo. Mediante este mecanismo es posible demostrar cierta propiedad para un conjunto infinito de valores, a través de la demostración para un caso base primero, y de un caso general después haciendo uso de una hipótesis de inducción. Pero en programación la inducción no sólo desempeña el papel de ser un método de demostración, sino que es aplicada también en el diseño de algoritmos para resolver muchos tipos de problemas. Con esta técnica, un problema es resuelto en dos partes: (1) planteando su solución en ciertos casos base; (2) estableciendo una descomposición recurrente de un caso general en subproblemas más cercanos al caso base.

  La técnica de diseño de algoritmos de divide y vencerás es el ejemplo más evidente de uso del razonamiento inductivo. Pero no el único. La programación dinámica también se basa en la descomposición de un problema en subproblemas, aunque en este caso no se aplica recursividad sino una estrategia ascendente. Otras técnicas de diseño, como backtracking o ramificación y poda, pueden hacer uso de la inducción. También el análisis de algoritmos recursivos requerirá, evidentemente, un razonamiento recurrente. Por otro lado, las especificaciones formales axiomáticas suelen estar basadas en la aplicación de la inducción, estableciendo el comportamiento de una operación para los casos base, y para los casos generales.

Recientemente, algunos de los grandes pioneros de la disciplina han criticado el estado actual de la programación, incidiendo de forma muy particular en la falta de formalización. Los programas son cada vez más complejos, incluyen características totalmente superfluas, no hay estándares claros y consensuados, y los programadores usan básicamente la intuición propia como única manera de afrontar los problemas. Los métodos formales se presentan como la clave para evitar esta situación. En una charla celebrada en la "*ACM Computer Science Conference*" en 1989, Edsger W. Dijkstra anunció la siguiente predicción: "*espero, digamos en unos 50 años, que la informalidad matemática de hoy día sea definitivamente una cosa del pasado*".

1.3.3. Eficiencia

La eficiencia constituye, desde nuestro punto de vista, el tercer gran eje de la disciplina de AED. Este es el principio esencial que diferencia a la algoritmia de una ciencia meramente matemática y teórica, dándole una entidad propia. No es suficiente con encontrar algoritmos que garanticen la solución a los problemas: es imprescindible que funcionen en un tiempo razonable. En la resolución de un problema –ya sea el diseño de un algoritmo o de una estructura de datos–, muchas opciones posibles deben ser descartadas por no ser razonables. El análisis de algoritmos es la rama de la algoritmia que

se encarga del estudio de la eficiencia. Pero el principio de eficiencia no se reduce a una etapa del proceso de programación, sino que debe estar presente desde el principio en todas las fases:

- En el análisis y diseño se tiene en cuenta como un requisito subyacente al problema, ya sea de forma implícita o explícita. Los algoritmos y las estructuras de datos utilizados deben estar orientados a ofrecer mecanismos de acceso y manipulación de la información coherentes y razonables.
- En la implementación, las decisiones deben buscar el uso razonable de los recursos, ajustándose al marco establecido en el diseño.
- En la fase de verificación, se deben realizar estudios experimentales de los programas, que pueden servir para comprobar su corrección y detectar posibles fallos o situaciones imprevistas.

Ya vimos que la eficiencia se puede definir como el criterio empresarial que relaciona los resultados obtenidos en función de los recursos consumidos. En programación, los recursos críticos son normalmente el tiempo y la memoria usados. En el diseño de una estructura de datos, el uso de memoria debe ser razonable en relación al tamaño de los datos representados, y las operaciones de acceso o modificación deben ser adecuadas para el tipo de necesidades existentes. En cuanto a los algoritmos, normalmente el tiempo de ejecución es el factor más importante, aunque también el uso máximo de memoria puede ser un parámetro crítico.

En los primeros ordenadores, muy lentos y con escasa memoria, la eficiencia era entendida como una necesidad básica. Se preveía que con la mejora de los ordenadores dejara de ser un factor relevante. Sin embargo, el incremento en la potencia de cálculo ha venido acompañado con un aumento proporcional del tamaño de los problemas y las aplicaciones exigidas. Un ejemplo son las necesidades en la representación gráfica. Los primeros PC utilizaban un terminal de texto de 80x25 celdas y un modo gráfico de 320x240 píxeles. Cuando los ordenadores aumentaron su capacidad, el modo gráfico aumentó a 640x480 (4 veces más), después a 800x600 con 3 bytes de color (18,75 veces más), a 1200x1024 (48 veces más), etc.; al mismo tiempo aumentan las velocidades de *refresco*, requiriéndose una actualización de 50 imágenes por segundo. En consecuencia, la eficiencia sigue siendo hoy día un requisito sustancial.

Cuando hablamos de eficiencia no nos referimos a la optimización del tiempo que puede ser conseguida, por ejemplo, programando determinado fragmento del código en ensamblador –que, a lo sumo, podría producir una mejora en un factor constante–, sino en cuanto al orden de complejidad de los algoritmos y las estructuras de datos usados. Es decir, lo relevante es el comportamiento asintótico (para tamaños suficientemente grandes) e independiente de constantes multiplicativas.

Un claro ejemplo de la carrera por la eficiencia lo encontramos en el problema de la multiplicación de matrices.

- El algoritmo clásico de multiplicación de matrices, consistente en multiplicar filas por columnas, tiene un orden de complejidad de $O(n^3)$, para matrices cuadradas de tamaño n. Durante mucho tiempo se creyó que ésta era la complejidad del problema, y que no podían existir algoritmos más eficientes.
- En 1969, V. Strassen rompe la barrera del $O(n^3)$, e inventa un algoritmo de multiplicación de matrices con un orden de $O(n^{\log_2 7}) \approx O(n^{2,808})$, basado en una descomposición no trivial usando divide y vencerás.
- Casi una década después, Victor Pan descubre un algoritmo para el mismo problema, también basado en divide y vencerás, con un orden de complejidad de $O(n^{2,796})$.

- Aunque la propuesta de V. Pan suponía un avance muy pequeño, dio origen a la denominada *guerra de los decimales*, en el intento de conseguir reducir la constante a la que está elevada n. Se descubrieron numerosos algoritmos cada vez más eficientes asintóticamente. Para finales de 1979 se sabía que era posible multiplicar dos matrices en un tiempo con un $O(n^{2,522})$.
- El algoritmo más rápido que se conoce actualmente fue propuesto en 1986 por D. Coppersmith y S. Winograd. Su método está basada en progresiones aritméticas y es capaz de funcionar asintóticamente en un $O(n^{2,376})$.

A consecuencia de las constantes ocultas en los tiempos de ejecución, ninguno de los algoritmos posteriores al de Strassen tiene una aplicación práctica real. En la actualidad, no se conoce cuál es el límite teórico del problema de multiplicación de matrices, es decir, el tiempo del algoritmo más rápido que resuelva el problema, por lo que la carrera aún continúa.

En otros casos, el factor eficiencia ha originado la aparición de nuevas técnicas de diseño de algoritmos, que de otro modo no habrían estado justificadas. Por ejemplo, el problema del viajante –que como ya mencionamos es NP-completo– es de una gran utilidad, a pesar de su complejidad computacional implícita. Muchas técnicas han sido desarrolladas intentando resolverlo: técnicas de búsqueda local, técnicas probabilistas, algoritmos de aproximación, técnicas de enfriamiento simulado, algoritmos genéticos, computación con ADN, etc.

Finalmente, desde un punto de vista más general, podríamos considerar también en la eficiencia el consumo de recursos humanos: el tiempo y la capacidad de los propios programadores. De esta manera, aparecerían otros criterios de eficiencia relacionados con las personas: facilidad de programación y de mantenimiento, legibilidad, tamaño del código, reutilización, etc. En cualquier caso, la eficiencia implica siempre un compromiso entre resultados y recursos. El compromiso es la expresión de la naturaleza práctica de la disciplina de AED: el objetivo final es crear soluciones que funcionen, y que lo hagan de manera razonable.

1.4. El presente y el futuro de la disciplina

El estado actual de la disciplina de AED está marcado por la desmembración de su ámbito de estudio en diferentes ramas, como ya hemos mencionado previamente. Los grandes avances, conceptos revolucionarios y técnicas generales, tuvieron lugar durante las décadas de 1950, 1960 y 1970. A partir de entonces, la especialización de las investigaciones ha dado lugar a una virtual desaparición de los algoritmos y las estructuras de datos como tema de investigación independiente. Pero la disciplina subsiste en el entorno académico, como el tronco del que toman su base muchas otras ramas de la informática. Es en estas otras materias donde continúan los avances y las nuevas propuestas en AED. Sin querer ser exhaustivos, podemos señalar las siguientes grandes áreas:

- **Bases de datos y sistemas de información.** El problema de representar información de manera eficiente, tanto en el uso de memoria como en el tiempo de acceso a los datos, es una de las cuestiones básicas de este área. Muchas estructuras de datos fueron diseñadas específicamente para su uso en bases de datos, como los árboles B o la dispersión extensible, y es previsible que sigan apareciendo otras nuevas en el futuro. Es más, con el aumento de los volúmenes de información manejados por estos sistemas, la eficiencia es un factor cada vez más crítico. Un ejemplo claro son los buscadores de páginas web, que deben procesar miles de consultas por segundo entre miles de millones de páginas.
- **Redes.** El desarrollo de las redes de ordenadores aprovechó en gran medida todos los avances que la algoritmia había producido en relación al estudio de los grafos. Problemas de conectividad,

caminos mínimos, árboles de expansión, de flujo, etc., habían sido ya resueltos antes de que existieran las primeras redes de ordenadores. Con la aparición de las grandes redes surgen nuevas cuestiones, como la necesidad de aplicar los algoritmos de forma distribuida. No hay un núcleo central que conozca toda la red, sino que cada nodo debe resolver la parte del problema que le corresponde.

- **Cálculo científico.** Aún hoy en día siguen existiendo grandes problemas de computación matemática, con ingentes necesidades de procesamiento, que requieren el estudio y la búsqueda de nuevas técnicas de cálculo. Estos nuevos algoritmos aparecen normalmente relacionados con aplicaciones como la predicción meteorológica, la dinámica de fluidos, la robótica, la simulación de moléculas, etc. Estos sistemas usan cálculos numéricos conocidos. El requisito surge del interés de acelerar los cálculos. La optimización de la eficiencia en los problemas de cálculo numérico ha sido en el pasado uno de los grandes focos de la algoritmia, y lo sigue siendo actualmente.

- **Inteligencia artificial.** Los sistemas de inteligencia artificial intentan simular el comportamiento humano en ámbitos de aplicación muy restringidos. En su base parten también del tronco común de la algoritmia. Las necesidades de representación en este ámbito han dado lugar a diversas ideas novedosas, como las redes de conocimiento, los grafos bayesianos o la lógica difusa. Y las estrategias propias de obtención de soluciones son el origen de algoritmos, como los algoritmos A, A*, etc. En muchos casos, los problemas que trata la inteligencia artificial son NP-completos; para ellos se han desarrollado nuevas técnicas de diseño, como los algoritmos genéticos, heurísticos y de aproximación. A veces, incluso, las propuestas han salido del ámbito de los ordenadores, como es el caso de la computación con ADN, basada en procesos bioquímicos.

- **Resolución de juegos.** Problemas como el ajedrez siguen siendo grandes objetivos de investigación en los que podemos encontrar propuestas interesantes. Las técnicas de resolución de juegos se basan, en esencia, en recorridos en árboles. Este ámbito tiene el aliciente añadido de la competencia entre contrincantes, aunque esto es también frecuentemente el motivo de la escasa divulgación científica de las técnicas aplicadas. Del estado actual de las competiciones más importantes, podemos intuir que los enfoques basados en búsqueda exhaustiva (que requerían máquinas grandes, rápidas y costosas, como Deep Blue), han sido sustituidos por propuestas donde priman las heurísticas, cada vez más refinadas (por ejemplo, el último campeón mundial, Deep Junior, funcionaba en un PC estándar con doble procesador).

- **Gráficos y multimedia.** El área de la informática gráfica y multimedia debe hacer frente a las grandes cantidades de datos procesadas, para lo cual se usan mecanismos de compresión adecuados. Una de las claves de la revolución del multimedia fue la aparición de los estándares de compresión MPEG, basados en algoritmos previamente conocidos, como la FFT y los códigos de Huffman. Además, se han propuesto y utilizado algoritmos específicos, orientados al procesamiento de señales digitales y a la generación gráfica. Los nuevos estándares de MPEG, que incluyen flujos múltiples y metainformación, darán lugar a nuevas formas de estructurar los datos y nuevos algoritmos de análisis de señales.

- **Visión artificial.** Los problemas y los algoritmos que aparecen en visión por computador son, en buena parte, algunos de los problemas que ya había tratado la algoritmia. Un ejemplo son los acercamientos de alto nivel, basados en representaciones de grafos. Problemas NP-completos, como el de asignación y el isomorfismo de grafos, pueden ser muy útiles en el ámbito del reconocimiento de patrones. Otras técnicas y algoritmos originales han sido desarrollados en este

área y podrían resultar interesantes en otras, como por ejemplo el algoritmo EM, para el ajuste de una muestra de datos a un modelo.

- **Seguridad y criptografía.** Con la aparición de las redes a nivel mundial, la seguridad y la necesidad de codificar la información se hacen cada vez más importantes. Uno de los grandes avances lo constituyó la invención del algoritmo RSA. A partir de entonces se han sucedido numerosas propuestas, basadas fundamentalmente en propiedades matemáticas de los números grandes.

- **Programación paralela.** El estudio y diseño de algoritmos paralelos, ejecutados simultáneamente en muchas máquinas, es otro de los ámbitos donde sigue teniendo lugar el avance en la algoritmia. En muchos casos se trata de problemas de cálculo matricial, para los cuales existen algoritmos conocidos. El interés es adaptar esos algoritmos a un modelo de ejecución con múltiples procesadores.

En definitiva, el carácter central de la disciplina de AED dentro de la informática, hace que prácticamente cualquier rama pueda ser relacionada con la misma. Todas las ramas han adaptado las ideas generales a sus peculiaridades propias y, a partir de ahí, los nuevos algoritmos han sido orientados a aplicaciones específicas. En nuestra opinión, aún queda un amplio margen para el avance del núcleo de la disciplina, lo cual podrá repercutir en todas las ramas que se desprenden. Podemos señalar, a título personal, las siguientes ideas con vistas al futuro:

- **Algoritmos de inspiración biológica.** La analogía con procesos biológicos ha sido el origen de muchos algoritmos, y con toda probabilidad será una fuente de inspiración en el futuro. Desde los esquemas como divide y vencerás o avance rápido –basados en el proceder habitual de los humanos–, hasta los algoritmos genéticos –inspirados en los procesos naturales de la evolución–, podemos encontrar numerosos ejemplos de esta idea. A nivel más ordinario, la formalización del conocimiento propio es de gran utilidad en muchos problemas, es decir, pensar cómo se podría resolver un problema "a mano" y trasladarlo después a un ordenador.

- **Algoritmos con huecos.** Un esquema algorítmico establece una estructura básica, en la cual quedan indefinidos algunos componentes genéricos (componentes que pueden ser algoritmos o estructuras de datos). En esencia, definiendo esos componentes para cada problema dado obtendríamos un algoritmo que lo soluciona. De esta forma, podemos ver los componentes como "huecos que deben ser rellenados". A nuestro parecer, la idea de los esquemas algorítmicos como "algoritmos con huecos" aún no ha sido suficientemente explotada.

  El desarrollo de esta idea supondría cambiar la filosofía en el diseño de los algoritmos, centrándose más en los problemas particulares que en las estructuras generales. Además, una vez definidos los huecos sería posible, en algunos casos, rellenar con ellos diferentes esquemas algorítmicos. Por ejemplo, los esquemas de avance rápido, backtracking y ramificación y poda podrían tener, esencialmente, los mismos componentes genéricos; una vez definidos, se podría optar por usar uno u otro esquema. La programación OO ofrece un marco adecuado para la implementación de estas ideas. Las funciones diferidas, el mecanismo de herencia y la redefinición son usados ya, de hecho, en el problema relacionado de definir iteradores sobre colecciones.

- **Algoritmos para la creación de algoritmos.** Entrando dentro del campo de lo hipotético, podríamos plantearnos la cuestión, ¿es posible diseñar un algoritmo capaz de diseñar algoritmos? El supuesto *meta-algoritmo* recibiría como entrada el enunciado de un problema algorítmico

(número y tipo de posibles parámetros de entrada y valores esperados en función de los anteriores) y debería producir en la salida un algoritmo que resolviera los casos concretos de ese problema.

Construir este *meta-algoritmo* implicaría dotar al ordenador de la capacidad humana de resolución de problemas y, por lo tanto, entraría dentro del ámbito de la inteligencia artificial. Por otro lado, comentamos al principio que los algoritmos codifican el conocimiento de un humano para resolver problemas. En este caso, sería necesario codificar la inteligencia humana para construir algoritmos lo cual, sin duda, es una labor nada trivial. Es más, el problema general es, previsiblemente, no computable. No obstante, creemos que sería interesante investigar hasta dónde se podría llegar en casos simplificados, por ejemplo, limitando el lenguaje de descripción de problemas, usando esquemas algorítmicos básicos, etc. De esta manera sería posible obtener prototipos de algoritmos a partir de especificaciones. Algo parecido ha sido ya tratado con las especificaciones formales de TAD. La investigación en algoritmos es un campo aún sin explorar.

Capítulo 2

El contexto de desarrollo

"El potencial de la universidad en el futuro recaerá
menos en la información y más en la academia como comunidad,
menos en las clases magistrales y más en las tutorías,
menos en la ciber-enseñanza y más en el carisma de sus profesores."
E.M. Noam, Electronics and the dim future of the university, Science, 1995

El diseño de un proyecto docente no puede ser independiente del contexto en el que se pretende aplicar, de la misma forma que el desarrollo de una asignatura sólo puede entenderse como parte de un plan de estudios, impartido por cierta institución en determinado momento y lugar. Por ello, la primera tarea a realizar es el análisis del contexto de la asignatura "Ampliación de Algoritmos y Estructuras de Datos". De acuerdo con Zabalza [Zabalza'93], los factores influyentes en el diseño de un proyecto docente se pueden articular en torno a cuatro dimensiones, que constituyen los contextos: institucional, curricular, personal y profesional. A su vez, en cada uno de estos contextos se puede diferenciar un espacio interior y otro exterior, en relación al propio ámbito de la universidad. En la figura 2.1 se muestran gráficamente estos espacios y contextos.

El contexto **institucional** está relacionado con las tradiciones, características y circunstancias propias de la universidad, el centro y el departamento donde se desarrolla la actividad docente. En el ámbito exterior, las normativas y legislaciones dispuestas por los representantes políticos –tanto a nivel estatal como autonómico– rigen y determinan el funcionamiento de estas instituciones universitarias.

Frente al contexto puramente académico –y con frecuencia demasiado distante– se sitúa el mundo **profesional**. Un proyecto docente no puede obviar las características propias de la práctica profesional en la disciplina de que se trate. Es importante analizar los conocimientos, habilidades y actitudes que se requerirá de los futuros profesionales. Por otro lado, los profesores universitarios, como profesionales de la educación superior, forman el ámbito interno de esta dimensión.

Los alumnos, su formación previa, hábitos, intereses, objetivos y aspiraciones, son otro elemento fundamental que da lugar a la dimensión **personal**. En cierto sentido, la finalidad de su formación es la incorporación al mundo laboral, por lo que estas aspiraciones y objetivos se verán también influenciados por la situación del mundo del empleo.

Pero, posiblemente, el factor más relevante en la selección de los contenidos de una asignatura es el propio plan de estudios del que forma parte. Este factor conforma el contexto **curricular**. La propuesta docente debe basarse en los conocimientos adquiridos en asignaturas previas, y complementar la formación de los alumnos según los objetivos del currículum en cuestión. A más largo plazo, son

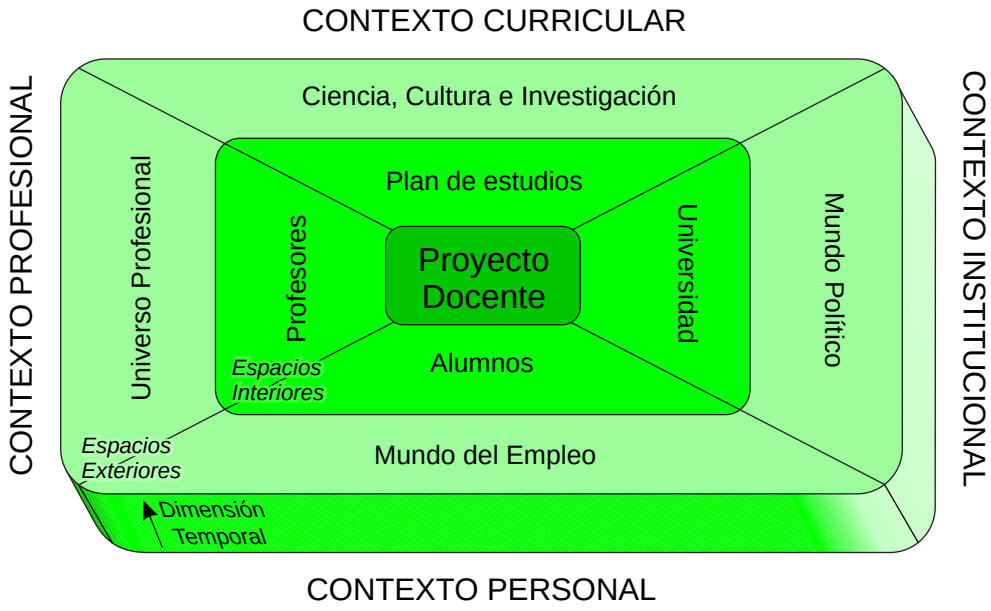


Figura 2.1: Dimensiones influyentes en el proceso de toma de decisiones curriculares, [Zabalza'93]. Se ha incorporado la dimensión temporal, presente en los cuatro contextos propuestos por Zabalza.

los avances científicos y su calado en la cultura general de la disciplina los que influirán en el diseño curricular de los planes de estudios.

En las siguientes secciones se hará un análisis detallado de cada uno de los componentes del contexto en el que se desarrollará la docencia de la materia objeto de concurso. Es evidente que este estudio es sólo un reflejo de la situación presente. Los imparables cambios culturales, sociales, legislativos y el marcado dinamismo de la disciplina de la informática podrían constituir un nuevo eje, la *dimensión temporal* –que libremente hemos incorporado al esquema de Zabalza–, a lo largo del cual se mueven las cuatro dimensiones anteriores. Este nuevo eje es sólo un recordatorio de que un proyecto docente no es algo inamovible, sino que debe estar en un continuo proceso de evolución, tendente a su actualización y mejora.

2.1. El contexto institucional

Como ya mencionamos en la introducción, la dimensión institucional del contexto en el que se desenvuelve un proyecto docente está compuesta por los organismos donde tiene lugar la docencia –Universidad, Centro, Departamento y Área de conocimiento– y la legislación que regula su funcionamiento. En nuestro caso, consideramos conveniente hacer referencia tanto al marco legal bajo el que fue convocada la plaza objeto de concurso (la Ley Orgánica de Reforma Universitaria de 1983), como al marco legislativo actual (la Ley Orgánica de Universidades de 2001). Tampoco podemos dejar de referirnos al contexto más amplio, el contexto europeo, que sin duda alguna será el motor de importantes reformas institucionales en un futuro no muy lejano.

2.1.1. Marco legal

La Ley de Reforma Universitaria

La Ley Orgánica 11/1983 de Reforma Universitaria [LRU'83] (en adelante LRU), aprobada el 25 de agosto de 1983 y publicada en el BOE número 209 de 1 de septiembre, es la primera ley reguladora del sistema universitario español tras la aprobación de la Constitución de 1978. El notable crecimiento del número de estudiantes, la integración en el ámbito universitario europeo y la necesidad de impulsar el desarrollo científico-técnico de la sociedad española eran los tres motivos que justificaban la necesidad de esta reforma.

La LRU define la Universidad como la institución a la que corresponde el servicio público de la educación superior. Tal y como proclama el preámbulo, *"la Universidad no es patrimonio de los actuales miembros de la comunidad universitaria, sino que constituye un auténtico servicio público referido a los intereses generales de toda la comunidad nacional y de sus respectivas Comunidades Autónomas"*. Este servicio se realiza a través de la docencia, el estudio y la investigación. En particular, la ley enumera las siguientes funciones de la Universidad:

a) La creación, desarrollo, transmisión y crítica de la ciencia, de la técnica y de la cultura.

b) La preparación para el ejercicio de actividades profesionales que exijan la aplicación de conocimientos y métodos científicos o para la creación artística.

c) El apoyo científico y técnico al desarrollo cultural, social y económico, tanto nacional como de las Comunidades Autónomas.

d) La extensión de la cultura universitaria.

Para que estas funciones se puedan desarrollar efectivamente, logrando altos niveles de calidad docente e investigadora, es necesario garantizar las condiciones de libertad y autonomía universitaria *"pues sólo en una Universidad libre podrá germinar el pensamiento investigador, que es el elemento dinamizador de la racionalidad moderna y de una sociedad libre"*. La libertad de la Universidad se fundamenta en el principio de la libertad académica, manifestado en las libertades de cátedra, de investigación y de estudio. Por otro lado, la autonomía universitaria comprende la autonomía estatutaria o de Gobierno, académica o de planes de estudio, financiera o de gestión de sus recursos y en la capacidad de seleccionar y promocionar a su profesorado. Según el Artículo 3º de la LRU, la autonomía universitaria, entre otras, le otorga la capacidad para:

a) La elaboración de los Estatutos y demás normas de funcionamiento interno.

b) La elección, designación y remoción de los órganos de gobierno y administración.

c) La elaboración, aprobación y gestión de sus presupuestos y la administración de sus bienes.

d) El establecimiento y modificación de sus plantillas.

e) La selección, formación y promoción del personal docente e investigador y de administración y servicios, así como la determinación de las condiciones en que han de desarrollar sus actividades.

f) La elaboración y aprobación de planes de estudio e investigación.

g) La creación de estructuras específicas que actúen como soporte de la investigación y la docencia.

h) La admisión, régimen de permanencia y verificación de conocimientos de los estudiantes.

i) La expedición de sus títulos y diplomas.

El reconocimiento de la autonomía universitaria es, por lo tanto, una de las principales aportaciones de la LRU respecto a la situación anterior a su aprobación. Esta autonomía se concreta, fundamentalmente, en la capacidad de las Universidades de elaborar sus propios **Estatutos** y toda la normativa que conforma el marco jurídico en el que se encuadra el quehacer universitario. Además del impulso de la autonomía universitaria, podemos destacar estos otros aspectos como los más importantes e innovadores de la Ley Orgánica de Reforma Universitaria:

- La creación del **Consejo Social**, definido por la LRU como el órgano de participación de la sociedad en la Universidad. A este órgano le corresponde la aprobación del presupuesto y la programación plurianual de la Universidad, a propuesta de la Junta de Gobierno, la supervisión de las actividades de carácter económico y el rendimiento de sus servicios, y promover la colaboración de la sociedad en la financiación de la Universidad. Está compuesto por una representación de la comunidad universitaria y, en un mayor porcentaje, por una representación de los intereses sociales, culturales y económicos de la Comunidad Autonómica correspondiente.

- La atribución de funciones al **Consejo de Universidades**, el cual funciona como un organismo estatal para la ordenación, coordinación, planificación, propuesta y asesoramiento en materia de enseñanza universitaria. El Consejo de Universidades se constituyó en 1985 y entre sus competencias se encuentran: proponer al Gobierno los títulos con carácter oficial y validez en el territorio nacional, acordar las directrices generales para la elaboración de los planes de estudios conducentes a estos títulos y los criterios generales en materia de convalidación de estudios cursados en centros españoles y extranjeros, y regular la homologación de títulos extranjeros. Son miembros de este organismo el Ministro de Educación, los responsables de educación universitaria de las diferentes Comunidades Autónomas, los rectores de las universidades públicas y otros quince miembros elegidos por el Congreso de los Diputados, el Senado y el Gobierno.

- La potenciación de los **Departamentos**, frente a la antigua organización facultativa. Según el Artículo 8º, *"los Departamentos son los órganos básicos encargados de organizar y desarrollar la investigación y las enseñanzas propias de su respectiva área de conocimiento en una o varias Facultades, Escuelas Técnicas Superiores o Escuelas Universitarias"*. Los Departamentos agrupan a todos los docentes e investigadores de áreas relacionadas. El máximo órgano de representación de un Departamento es su Consejo de Departamento.

- La simplificación de la **estructura jerárquica** del profesorado. Sin ahorrar calificativos, la Ley describe la situación existente hasta ese momento como un *"caos selvático e irracional"*. Se crean cuatro únicas categorías de profesorado: Catedráticos de Universidad, Profesores Titulares de Universidad, Catedráticos de Escuelas Universitarias y Profesores Titulares de Escuelas Universitarias. Además, se crean los contratos de: Profesor Asociado, Profesor Visitante, Ayudante de Escuela y Ayudante de Facultad. Asimismo, se establecen las directrices para el desarrollo de la carrera docente en la institución universitaria española.

Directrices Generales de Planes de Estudios

Como hemos indicado antes, la LRU encomendó al Consejo de Universidades la elaboración de unas Directrices Generales de Planes de Estudios, en la que se establecen las líneas básicas y comunes que deben cumplir todos los planes de estudios de las titulaciones válidas en el territorio español. Estas directrices debían ser completadas con las Directrices Generales Propias de cada titulación.

El Real Decreto 1497/1987, de 27 de noviembre (BOE de 14 de diciembre) –modificado posteriormente en los decretos adicionales RD 1267/1994, de 10 de junio (BOE de 11 de junio), RD 2347/1996, de 8 de noviembre (BOE de 23 de noviembre), RD 614/1997 (BOE de 16 de mayo) y RD 779/1998, de 30 de abril (BOE de 1 de mayo)– contiene las Directrices Generales de Planes de Estudios, estableciendo las siguientes características a las que debían ajustarse todos los planes de estudios:

- **Estructuración en ciclos.** Las enseñanzas universitarias se estructuran en ciclos, pudiendo existir hasta un máximo de tres. La superación del primer ciclo permite la obtención de un título oficial y el consiguiente acceso a la actividad profesional. Su duración es de dos o tres años. Se posibilita la continuación de los estudios en un segundo ciclo de dos años, o excepcionalmente tres. Finalmente, el tercer ciclo, de carácter mucho más especializado, tiene como objetivo la formación investigadora dentro de un área de conocimiento específica.

- **Racionalización de la carga lectiva.** Se limita la carga lectiva por curso académico al intervalo de entre veinte y treinta horas semanales. Además, se introduce el *crédito* como la unidad de valoración empleada para medir la carga lectiva y estimar la capacidad y dedicación docente del profesorado. Un crédito equivale a diez horas de clase. Las principales ventajas de su uso son la mayor apertura de los planes de estudio y la flexibilización del currículum del estudiante. Los planes de estudios de las distintas titulaciones deben establecer los créditos teóricos y prácticos que corresponden a cada materia de estudio.

- **Tipos de contenidos.** Las directrices generales establecen los tres siguientes bloques, entre los que se tendrán que distribuir los contenidos de cada plan de estudios:

  1. **Materias troncales.** Contenidos homogéneos mínimos de todos los planes de estudios de un mismo título oficial. Deben ser establecidos por las directrices generales propias de cada titulación. Su carga lectiva debe ser como mínimo un 30 % del total en primer ciclo, y un 20 % en el segundo.
  2. **Materias no troncales.** Pueden ser establecidas discrecionalmente por cada Universidad, en forma de asignaturas obligatorias –necesarias para todo titulado–, o asignaturas optativas –entre las que el alumno puede elegir al objeto de intensificar el perfil que le interese–.
  3. **Materias de libre elección.** Pueden ser escogidas por los alumnos de entre una bolsa de asignaturas ofertadas por la Universidad correspondiente, o por otra Universidad con la que se establezca un convenio a tal efecto. Permiten a los alumnos configurar libremente su propio currículum. El porcentaje de créditos de este tipo sobre la carga lectiva total no puede ser inferior al 10 %.

- **Uniformidad de contenidos.** Finalmente, intentando conseguir la necesaria coherencia en los planes de estudios conducentes a la obtención de títulos oficiales, cada titulación posee sus Directrices Generales Propias. Estas directrices propias establecen la denominación del título, su estructura cíclica, duración, carga lectiva mínima y máxima por cada ciclo, los descriptores de los contenidos troncales, sus créditos teóricos y prácticos y la correspondiente vinculación a áreas de conocimiento. Estudiaremos las Directrices Generales Propias de la titulaciones de informática en el punto 2.2.1.

Actualmente, y como consecuencia de los cambios que requerirá la adaptación de España al Espacio Europeo de Educación Superior que analizaremos en el punto 2.1.2, se espera la inminente

elaboración de unas nuevas directrices generales. Como veremos, estas modificaciones supondrán una revisión profunda de los actuales planes de estudios, situación a la que deberán adaptarse las Universidades españolas antes de finalizar la década presente.

La Ley Orgánica de Universidades

Tras dieciocho años de funcionamiento de la LRU, se emprende una nueva reforma de la legislación universitaria. Este nuevo proceso de reforma desemboca en la Ley Orgánica 6/2001 de Universidades [LOU'01] (en adelante LOU), aprobada por el Pleno del Congreso de los Diputados el 20 de diciembre de 2001. Reconociendo los profundos cambios producidos en la Universidad española en las dos últimas décadas –derivados del espectacular crecimiento de las Universidades[1] y de las tecnologías de la información y las comunicaciones–, se expone, en consecuencia, la necesidad de actualizar todos los aspectos académicos, de docencia, de investigación y de gestión, de la vida universitaria.

En términos generales, la LOU *"pretende dotar al sistema universitario de un marco normativo que estimule el dinamismo de la comunidad universitaria,* [...] *que mejore su calidad, que sirva para generar bienestar y que, en función de unos mayores niveles de excelencia, influya positivamente en todos los ámbitos de la sociedad"*. No casualmente, las referencias a la calidad y a la excelencia son una constante a lo largo del texto de la Ley. Otros de los grandes objetivos de la LOU son:

- Reorganizar las competencias entre los distintos actores, dotando a las Universidades de una mayor autonomía y profundizando las competencias de las Comunidades Autónomas en materia de enseñanza superior. A la Administración General del Estado le corresponde el impulso en la vertebración y cohesión del sistema universitario español.

- Fomentar la movilidad tanto de los estudiantes como de los profesores e investigadores, dentro del sistema español y también del europeo e internacional. Según la Ley, la movilidad supone una mayor riqueza y la apertura a una formación de mejor calidad. Además, la movilidad introduce elementos de competitividad entre las distintas Universidades.

- Responder a los nuevos retos de la educación superior en el siglo XXI. Estos retos tienen dos vertientes: por un lado, los derivados de la enseñanza no presencial, a través de las nuevas tecnologías de la información y de la comunicación, y por otro lado los relativos a la necesidad de una formación a lo largo de la vida.

- Adaptar e integrar el sistema universitario español en el nuevo espacio universitario europeo, permitiéndole desarrollar su labor competitivamente junto a los mejores centros de enseñanza superior de Europa.

Para lograr estos objetivos, se diseña una arquitectura normativa más moderna y actualizada, que modifica muchos aspectos de la antigua LRU. Entre las reformas más importantes y significativas introducidas por la LOU podemos destacar las siguientes:

- **Ampliación de las competencias** de las Universidades y de las Comunidades Autónomas. Esto implica para las primeras una mayor eficiencia en el uso de los recursos públicos y nuevas atribuciones de coordinación y gestión para las segundas. Además de las competencias actuales, las Universidades adquieren otras relacionadas con la contratación de profesorado, el reingreso en el servicio activo de sus profesores, la creación de centros de enseñanza a distancia, los

[1] La Ley destaca la triplicación del número de Universidades, *"creándose centros universitarios en casi todas las poblaciones de más de cincuenta mil habitantes"*.

procedimientos para la admisión de estudiantes y la constitución de figuras jurídicas para el desarrollo de sus fines. Por su parte, a las competencias de las Comunidades Autónomas se añaden la regulación del régimen jurídico y retributivo del profesorado contratado, la capacidad para establecer retribuciones adicionales para el profesorado, la aprobación de programas de financiación plurianual y la evaluación de la calidad de las Universidades de su ámbito de responsabilidad.

- Se mantienen los **órganos de representación** definidos por la LRU, en condiciones similares aunque con ciertos cambios. En cuanto a los órganos unipersonales, están compuestos por: Rector, Vicerrectores, Secretario general, Decanos o Directores de Escuelas Técnicas, y Directores de Departamento. Se modifica el proceso de elección del Rector, siendo ahora por elección directa y sufragio universal libre y secreto[2]. En lo referente a los órganos colegiados, continúan las figuras del Consejo Social, como órgano de participación de la sociedad en la Universidad –reforzando sus competencias y la participación de actores externos a la Universidad–, el Claustro Universitario, como máximo órgano de representación de la Comunidad Universitaria –aunque con ligeros cambios en su composición y atribuciones–, las Juntas de Facultad o Escuela y los Consejos de Departamento. En general, se potencia la participación en estos órganos de los profesores doctores frente a los no doctores.

- Se crean nuevos órganos para el gobierno de la Universidad, que sustituyen a la Junta de Gobierno. Estos órganos son: el **Consejo de Gobierno**, el **Consejo de Dirección** y la **Junta Consultiva**. El Consejo de Gobierno es el órgano de gobierno de la Universidad, encargado de establecer las líneas estratégicas y programáticas de la Universidad, así como las directrices y procedimientos para su aplicación. Está compuesto por el Rector, el Secretario general, el Gerente, un máximo de cincuenta miembros de la Comunidad Universitaria y tres miembros del Consejo Social, ajenos a la Universidad. La Junta Consultiva es el órgano ordinario de asesoramiento del Rector y del Consejo de Gobierno en materia académica. Además del Rector y del Secretario general, pueden formarlo hasta cuarenta profesores e investigadores de reconocido prestigio, con méritos docentes e investigadores acreditados. Finalmente, en el Consejo de Dirección están presentes los Vicerrectores, el Secretario general y el Gerente. Su objetivo es asistir al Rector en su actividad cotidiana al frente de la Universidad.

- El Consejo de Universidades de la LRU pasa a denominarse **Consejo de Coordinación Universitaria**. La Ley lo describe como el máximo órgano consultivo y de coordinación del sistema universitario, al que corresponden las funciones de consulta sobre política universitaria, y de coordinación, programación, informe, asesoramiento y propuesta en las materias relativas al sistema universitario. Como principal novedad, entran a formar parte del mismo los rectores de las Universidades privadas, aunque con restricciones cuando se traten cuestiones que afecten sólo a las públicas.

- Se crea una **prueba de habilitación** del profesorado, a nivel nacional, imprescindible para poder acceder a los cuerpos docentes universitarios. El funcionamiento de esta prueba es regulado por el Real Decreto 774/2002, de 26 de julio de 2002. El objetivo de esta reforma es mejorar el proceso de selección del profesorado, garantizando los principios de igualdad, mérito y capacidad de los candidatos. La habilitación consta de dos o tres pruebas, según la escala docente, en la que los candidatos deben realizar: una presentación y discusión de sus méritos y currículum docente e investigador (para todas las categorías), y de un proyecto (docente en el caso de los

[2]En lugar de la elección en el Claustro, como establecía anteriormente la LRU.

Titulares de Escuelas, y docente e investigador para los Titulares de Universidad y Catedráticos de Escuelas); la exposición y debate de un tema del programa presentado por el candidato (para todas las categorías excepto para los Catedráticos de Universidad); y la exposición y debate con la Comisión de un trabajo original de investigación (para todas las categorías excepto para los Titulares de Escuelas). Sin duda, esta fue una de las medidas que más controversia creó en su momento, principalmente entre la comunidad docente e investigadora.

- Definición de nuevas **figuras de profesorado**, redefinición de las existentes y reorganización de la carrera docente. En particular, se definen las figuras de ayudante, profesor ayudante doctor, profesor colaborador, contratado doctor, asociado, emérito y visitante. La carrera docente empieza con la figura de ayudante, contratado por un máximo de cuatro años, con el objetivo principal de completar su formación investigadora, aunque puede participar en tareas de docencia. Tras este periodo, si el candidato consigue realizar su doctorado, podría aspirar a un puesto de profesor ayudante doctor –por un máximo de cuatro años, y siempre en una Universidad distinta a aquella donde ejerció de ayudante–, o bien a un puesto de profesor colaborador –por un tiempo indefinido, y tras recibir una evaluación favorable de la Agencia Nacional de Evaluación de la Calidad y Acreditación o del órgano de evaluación externa que la Ley de la Comunidad Autónoma determine–. Por su parte, la figura de profesor contratado doctor es también una figura laboral, de duración indefinida, a la que pueden aspirar los candidatos que acrediten al menos tres años de actividad docente e investigadora, o prioritariamente investigadora.

- Creación de la **Agencia Nacional de Evaluación de la Calidad y Acreditación (ANECA)**, como un organismo externo e independiente que desarrolla una actividad evaluadora, propia de sistemas universitarios avanzados, para medir el rendimiento de las Universidades y reforzar su calidad, transparencia, cooperación y competitividad. Esta Agencia es la encargada de evaluar tanto las enseñanzas de primer, segundo y tercer ciclo, como la actividad investigadora, docente y de gestión, así como los servicios y programas de las Universidades. La influencia de esta institución es fundamental, ya que a medio plazo se requerirá que todos los títulos de las Universidades españolas estén acreditados para que se reconozca oficialmente su validez. La Ley también deja la posibilidad de que las Comunidades Autónomas creen sus propios organismos de evaluación de la calidad, con competencias en las Universidades de su ámbito respectivo.

- La implantación del **distrito universitario abierto**, con el propósito de propiciar la movilidad y la igualdad en las condiciones de acceso a los estudios universitarios. Se crean mecanismos para garantizar que los estudiantes pueden acceder a los estudios que deseen, en la Universidad de su elección, en condiciones de igualdad de oportunidades y no discriminación. Por otro lado, como ya hemos comentado, se deja a las Universidades la capacidad de establecer los procedimientos para la admisión de los estudiantes que soliciten ingresar en centros de las mismas. La consecuencia práctica de esta reforma es la eliminación de la selectividad, que es sustituida por una prueba de reválida –pendiente de implantación– necesaria para conseguir el título de bachiller. Además, los candidatos deberán realizar las pruebas de acceso que cada Universidad estime oportunas.

- Creación de la figura del **Defensor del Universitario**, de acuerdo con la Disposición adicional 14. Su funcionamiento está regido por los principios de independencia y autonomía, y tiene por misión velar por el respeto a los derechos y las libertades de los profesores, estudiantes y personal de administración y servicios, ante las actuaciones de los diferentes órganos y servicios universitarios. Su régimen de funcionamiento debe establecerlo cada Universidad en sus Estatutos.

- Finalmente, y a diferencia de la LRU, hay una detallada **regulación de las Universidades**

privadas, con el objetivo de introducir en las Universidades privadas las exigencias ya requeridas a las Universidades públicas. La Ley fija los principales aspectos sobre los requisitos para el establecimiento y funcionamiento de sus centros, la evaluación de su calidad, y la expedición y homologación de los títulos a que conducen los estudios que imparten.

2.1.2. El Espacio Europeo de Educación Superior

Durante el último lustro, el empeño por impulsar el proceso de *construcción europea* ha sido una constante en la política de los países de la Unión. A corto plazo, previsiblemente, este proceso desembocará en la aprobación de una Constitución Europea, que consolidará la Europa de los ciudadanos frente a la Europa *"del Euro, de los bancos y de la economía"*. Asistimos, por lo tanto, a una integración europea a todos los niveles, a la cual nuestro sistema Universitario no será ajeno.

La construcción del denominado **Espacio Europeo de Educación Superior (EEES)** es un proceso que está en marcha desde la declaración de La Sorbona (1998), y continuado después con la declaración de Bolonia (1999) y los comunicados de Praga (2001) y Berlín (2003). Su implantación supondrá cambios transcendentales, que abarcarán desde la reordenación de la actual organización en ciclos de nuestras titulaciones, hasta el modelo educativo que soporta nuestros métodos de enseñanza. De hecho, la LOU ya dedica su Título XIII (artículos 87, 88 y 89) a las medidas necesarias para la plena integración del sistema español en este nuevo contexto.

Las Declaraciones de La Sorbona, Bolonia, Praga y Berlín

Las raíces del proceso de construcción del EEES se establecen en la primavera de 1998, cuando los ministros de educación de Francia, Alemania, Italia y Reino Unido, reunidos en la celebración del aniversario de la Universidad de París, firman la Declaración de La Sorbona [Sorbona'98]. Admitiendo que la integración europea no puede limitarse a la dimensión económica, se reconoce la necesidad de construir una **Europa del conocimiento**, un espacio común en las dimensiones cultural, social, científica, tecnológica e intelectual. Y es a las Universidades a las que corresponde ser el motor de estos cambios.

La armonización, comparabilidad y legibilidad de los títulos, el fomento de la movilidad y la formación a lo largo de la vida son algunas de las principales líneas maestras que se perfilan en la declaración. La integración, sin embargo, no es sinónimo de uniformidad o estandarización sino que debe respetar la diversidad cultural de los diferentes países y la autonomía de sus Universidades.

Pero es la Declaración de Bolonia, [Bolonia'99], firmada el 19 de junio de 1999, la que consolida y amplía este proceso, comprometiendo a los gobiernos de los 29 países firmantes a adoptar las medidas oportunas para la integración de sus sistemas de educación superior en el EEES. En concreto, se fijan seis grandes objetivos, con el propósito de lograrlos *"antes del final de la primera década del tercer milenio"*, es decir, en 2010. Los objetivos propuestos son los siguientes:

1. Adoptar un sistema de títulos de sencilla legibilidad y comparabilidad, a través de la introducción del **suplemento europeo al título**, con tal de favorecer la *ocupabilidad* de los ciudadanos europeos y la competitividad internacional del sistema europeo de enseñanza superior.

2. Adoptar un sistema de titulaciones basado esencialmente en **dos ciclos principales**, denominados grado y postgrado. El acceso al segundo ciclo precisa de la conclusión satisfactoria de los estudios de primer ciclo, que duran un mínimo de tres años. El título de grado será utilizable como cualificación en el mercado laboral europeo. El segundo ciclo debe conducir a un título de master o doctorado.

3. Establecer un **sistema común de créditos** –como el modelo ECTS[3]– como medio de promover la movilidad de estudiantes. Los créditos también pueden adquirirse en otros contextos, como la formación permanente.

4. Promover la **movilidad** tanto de los estudiantes, como de los profesores, investigadores y del personal técnico-administrativo, mediante la eliminación de los obstáculos para el pleno ejercicio de la libre circulación. Para profesores e investigadores se propone el reconocimiento y valorización de períodos de investigación en contextos europeos relacionados con la docencia y la formación. Para los estudiantes, se propone la posibilidad de obligar o estimular la realización de al menos un semestre de sus estudios en alguna Universidad de otro país.

5. Promocionar la **colaboración europea** en relación a garantizar la calidad de los sistemas universitarios de los distintos países, con vistas al diseño de criterios y metodologías educativas comparables.

6. Promover la **dimensión europea de la educación superior** y en particular, el desarrollo curricular, la cooperación institucional, esquemas de movilidad y programas integrados de estudios, de formación y de investigación.

Los comunicados de Praga [Praga'01] en 2001 y Berlín [Berlín'03] en 2003, corresponden a sendas reuniones de seguimiento, en las que los ministros de educación de la Unión Europea y de otros países del entorno europeo analizan el cumplimiento de los objetivos fijados, proponiendo nuevas actuaciones de coordinación encaminadas a la consecución de los mismos. Además, se incorporan los siguientes objetivos adicionales:

1. Introducir en la educación superior el concepto de **aprendizaje a lo largo de la vida** como elemento esencial para alcanzar una mayor competitividad europea, para mejorar la cohesión social, la igualdad de oportunidades y la calidad de vida.

2. Potenciar la **participación activa** de las universidades, de las instituciones de educación superior y de los estudiantes en el desarrollo del proceso de convergencia. Difícilmente se lograrán los objetivos planteados si las reformas se quedan en meros cambios legislativos.

3. Promocionar el **atractivo del EEES** a los ciudadanos europeos y del resto de mundo, mediante el desarrollo de sistemas de garantía de la calidad y de mecanismos de certificación y de acreditación. Aunque la referencia no es explícita, está claro que el principal competidor en este sentido es el sistema de educación superior estadounidense.

Posteriores informes del Parlamento Europeo han expresado su apoyo incondicional a la creación de este espacio educativo común y, como ya hemos comentado, la LOU contiene tres artículos que establecen que el Gobierno adoptará las medidas necesarias para lograr los propósitos del EEES. En definitiva, la apuesta por la armonización de los sistemas de educación superior europeos es firme y no tiene marcha atrás.

[3] *European Credit Transferm System*, en español, Sistema Europeo de Transferencia de Créditos. Se trata de un sistema de cómputo que nace y se desarrolla con los programas de movilidad de estudiantes, y es utilizado en la actualidad en los planes de estudios de muchos países de la Unión Europea.

La integración de España en el EEES

En febrero de 2003, el Ministerio de Educación, Cultura y Deporte remite al Consejo de Coordinación Universitaria el Documento-Marco sobre "La integración del sistema universitario español en el Espacio Europeo de Enseñanza Superior" [MECD'03]. Este documento pretende ser un punto de partida para la reflexión y el estudio de las medidas concretas que se deben aplicar en el sistema universitario español, con el fin de alcanzar los objetivos establecidos en las declaraciones de La Sorbona, Bolonia y Praga. En particular, las medidas se refieren al sistema europeo de créditos, la estructura de las titulaciones, el Suplemento Europeo al Título (SET) y la garantía de la calidad.

- **Implantación del sistema de créditos europeos.** Recordemos que el crédito es la unidad de medida utilizada en la estructuración y organización de los planes de estudios. Homogeneizar esta unidad es esencial para facilitar el reconocimiento del trabajo realizado por los estudiantes en Universidades de diferentes países. En España, actualmente, el crédito mide el número de horas lectivas impartidas, ya sean de teoría o de prácticas. Es, por lo tanto, una medida centrada en la labor docente de los profesores. Frente a esta perspectiva, el crédito europeo propone una medida centrada en el aprendizaje de los estudiantes. El sistema conocido como ETCS es un ejemplo de esta filosofía, y el modelo usado para la definición del crédito europeo[4].

  Recientemente ha sido regulada la implantación del crédito europeo en España, por el Real Decreto 1125/2003, de 5 de septiembre (BOE de 18 de septiembre). Adoptando el modelo ETCS, se establece en 60 el número de créditos correspondientes a un año de trabajo del alumno a tiempo completo, siendo 30 el número de créditos de un semestre y 20 el de un cuatrimestre. Se considera que un año lectivo tiene una duración de entre 36 y 40 semanas. El tiempo medido por un crédito europeo incluye todo el trabajo del alumno: las clases docentes teóricas, prácticas y seminarios; la preparación y realización de exámenes; y las horas de estudio y los trabajos o proyectos que los alumnos deban realizar. El Decreto fija que *"el número mínimo de horas, por crédito, será de 25, y el número máximo, de 30"*[5]. De modo análogo, el reconocimiento de la labor docente de los profesores deberá incluir no sólo las horas dedicadas a impartir su docencia, sino también las dedicadas a prepararla y a organizar, orientar y supervisar el trabajo de los alumnos.

  Además, el Real Decreto citado regula también el sistema de calificaciones de los resultados obtenidos por los alumnos. En el nuevo modelo de calificación, las notas se deberán expresar con una escala numérica de 0 a 10, incluyendo un decimal. Otra novedad sustancial es la que trata de *poner en contexto* la calificación obtenida por el alumno en cada asignatura. O, en otras palabras, no es lo mismo un 7,5 en una asignatura *fácil* que en una *difícil*. De acuerdo con el Decreto, en el expediente académico del alumno, junto con sus notas deberá figurar *"el porcentaje de distribución de estas calificaciones sobre el total de alumnos"* en esa universidad, asignatura y curso particular. Por ejemplo, la calificación podría decir algo como: "7,5 superando al 12 %" en la asignatura *fácil*; y "7,5 superando al 74 %" en la *difícil*.

  Pero, si este nuevo sistema de cómputo supondrá variaciones sustanciales en las cargas de trabajo de los actuales planes de estudios, más importantes pueden llegar a ser los cambios que se vislumbran en la afirmación del Documento-Marco en el sentido de que *"esta nueva unidad de medida debe comportar* ***un nuevo modelo educativo*** *basado en el trabajo del estudiante y*

[4] De hecho, en algunos documentos oficiales ambos conceptos son usados como sinónimos.

[5] Frente a las 10 horas del crédito que actualmente usamos. Pero recordemos que nuestro sistema no cuenta el trabajo *extra* del alumno. A modo orientativo, se estima que 1 hora de clase requiere de media entre 1,5 y 2 horas adicionales de preparación y estudio. En cierto sentido, tenemos dos factores que se *compensan*: se requieren más horas por cada crédito europeo, pero se cuentan más horas de trabajo por cada clase.

no en las horas de clase, o, dicho de otro modo, centrado en el aprendizaje de los estudiantes, no en la docencia de los profesores". En definitiva, se trata de potenciar los modelos de enseñanza centrados en prácticas, trabajos y tutorías, frente a los métodos basados en las tradicionales lecciones magistrales.

- **Adaptación de las enseñanzas y títulos oficiales universitarios.** Otra de las reformas de mayor calado que requerirá la implantación del EEES, son las derivadas de la armonización de la estructura cíclica de las enseñanzas con el esquema propugnado en la Declaración de Bolonia; y particularmente en España, donde la organización actual está muy alejada de la propuesta. Este nuevo esquema, utilizado ya por muchos países europeos, está compuesto por un **primer nivel de grado** que dará lugar a la obtención de un título con cualificación profesional en el mercado laboral europeo, y un **segundo nivel de postgrado**, para cuyo acceso será necesario haber superado el primero, y que podrá dar lugar a la obtención del título de Master y/o Doctorado. Obviamente, esto implica una obligada revisión de nuestra estructuración de los estudios en tres ciclos. En la figura 2.2 se muestra gráficamente la situación actual y la propuesta.

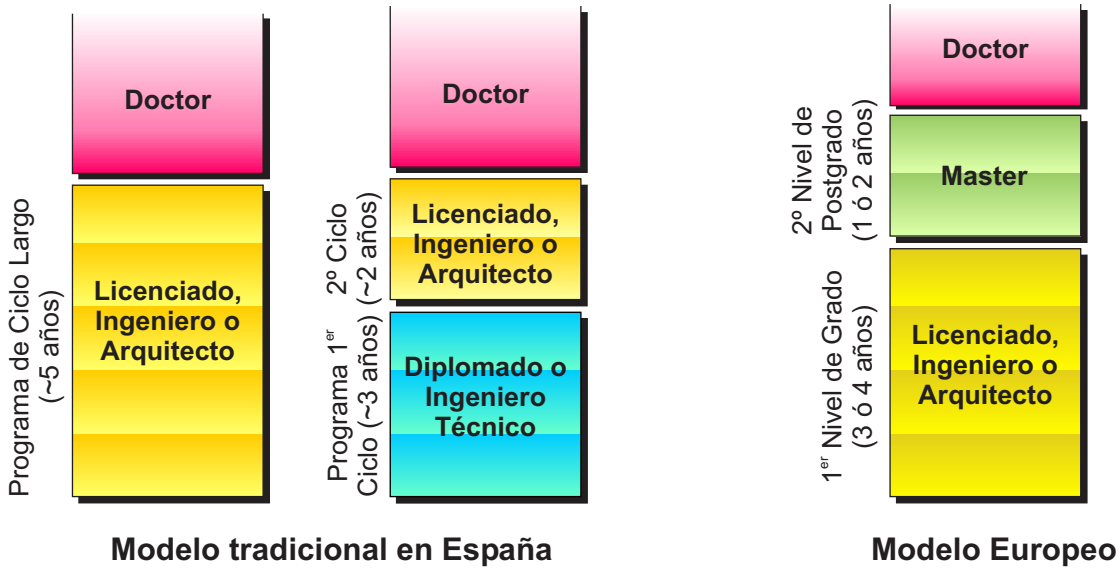


Figura 2.2: Estructuración cíclica de las titulaciones. Izquierda: situación actual en España. Derecha: esquema de niveles propuesto en la Declaración de Bolonia (1999).

Los estudios de primer nivel podrán constar de entre 180 y 240 créditos europeos, es decir, una duración de 3 ó 4 años. Con ellos el alumno podrá obtener el título oficial de Licenciado, Ingeniero o Arquitecto. El Documento-Marco plantea, además, una revisión del catálogo de titulaciones *"propiciando una disminución mediante las fusiones o agrupaciones necesarias para racionalizar el conjunto tanto desde el punto de vista nacional como europeo"*[6]. Para regular estas reformas, el Gobierno deberá desarrollar las nuevas directrices generales de planes de estudios, así como las directrices generales propias de cada titulación.

En cuanto al segundo nivel, deberá perseguir objetivos formativos más específicos, orientados a

[6]En este sentido, y en el caso concreto de las titulaciones de informática, parece que toman fuerza las perspectivas de que las tres titulaciones actuales (las dos técnicas y la superior) vuelvan a integrarse en una única titulación de Ingeniería en Informática.

una profundización intelectual, desarrollo académico disciplinar e interdisciplinar, especialización científica, orientación a la investigación o formación profesional avanzada. Estos estudios conducirán a los títulos de Master y/o Doctor. Pero no está clara la secuenciación de los estudios, existiendo varias alternativas: primero Master y luego Doctorado; Master y Doctorado independientes; u obligar para el Doctorado tener cierta cantidad de créditos de un Master. Respecto a la situación actual, los títulos de Master pasan a ser oficiales (aunque permitiendo también los no oficiales) y su duración deberá ser entre uno y dos años. En cuanto al Doctorado, se busca una revalorización de estos estudios[7] y un incremento en sus niveles de excelencia. Como hasta ahora, la obtención del título requerirá la elaboración y defensa de una tesis doctoral que deberá contener resultados originales de investigación. Se introduce como novedad la revisión externa de las tesis, previa a su defensa en un tribunal.

- **Suplemento Europeo al Título (SET).** En palabras del Documento-Marco, el SET *"constituye un elemento de transparencia ya que su objetivo fundamental es hacer comprensibles y comparables los títulos universitarios en Europa por medio de una información académica y profesional relevante para la sociedad, la universidad y los empleadores"*. En la práctica, el suplemento es un documento añadido al título recibido por el alumno, en el que se incluye información para comparar fácilmente los resultados de diversas universidades, y se resume el aprendizaje, las competencias y las capacidades profesionales adquiridas por el estudiante a lo largo de su vida en las distintas instituciones del EEES. Además de las calificaciones obtenidas por el alumno, se incluyen datos sobre la institución donde ha desarrollado sus estudios, la titulación cursada, los requisitos de acceso, la orientación profesional de los estudios, etc.

  En el Real Decreto 1044/2003, de 1 de agosto (BOE de 11 de septiembre) se establece el procedimiento para la expedición por las Universidades del Suplemento Europeo al Título. El formato de este documento se ajusta al modelo elaborado por la Comisión Europea y contiene ocho apartados: datos del estudiante, información de la titulación, del nivel de la titulación, del contenido y los resultados, de la función de la titulación, información adicional, certificación del suplemento y, por último, información sobre el sistema nacional de educación superior. Entre los subapartados en los que se desglosa, llama la atención el denominado "Sistema de calificación" que debe contener *"la distribución de las calificaciones en el conjunto de las asignaturas [de ese curso y esa Universidad] en los últimos dos años"*.

  Se prevé la implantación del SET en dos fases. La primera es una fase experimental, en la que se expedirá el suplemento para las titulaciones actuales, mencionando expresamente su carácter experimental. La segunda fase será la implantación definitiva del SET, y tendrá lugar una vez incorporado al sistema universitario español el sistema de créditos europeos. La expedición del SET es responsabilidad de las Universidades, aunque el Consejo de Coordinación Universitaria deberá establecer algunas directrices.

- **Acreditación académica y calidad.** La mejora de la calidad –en todos los sentidos, no sólo en el docente– es uno de los objetivos primordiales establecidos en todas las declaraciones que pusieron en marcha el proceso de construcción del EEES. Crear un clima de confianza mutuo entre los distintos países y sistemas universitarios del EEES es un requisito indispensable para conseguir objetivos como la movilidad y el reconocimiento de títulos. Y esta confianza sólo se

[7] Nuevamente refiriéndonos al caso de los estudios de informática, esta *revalorización* puede venir del hecho de reducir el ciclo largo (5 años) a unos estudios de 3 ó 4 años. En otras palabras, un alumno que se plantea hoy realizar una carrera de 5 años, verá el equivalente en el futuro a realizar un primer nivel de 3 ó 4 años, seguido de un Master y/o Doctorado de 1 ó 2 años.

puede lograr estableciendo los adecuados sistemas de garantía de la calidad, a través de los organismos externos de evaluación y acreditación de cada país. La creación de la ANECA en la LOU, tal y como vimos en la página 42, es la respuesta del Gobierno español a esta necesidad. La ANECA, junto con los órganos de evaluación que puedan crearse en las Comunidades Autónomas, serán las responsables de llevar a cabo las políticas de evaluación, certificación y acreditación. Con el fin de lograr la convergencia en los sistemas de garantía de calidad de los distintos estados, se deberá desarrollar al máximo la colaboración entre la ANECA y las agencias de otros países de la Unión Europea.

Todas estas modificaciones legislativas deberán tenerse en cuenta para el diseño de los nuevos planes de estudios que se propongan en adelante. Y, en todo caso, todas las titulaciones existentes actualmente deberán ser reformadas y adaptadas antes del 2010.

2.1.3. La Universidad de Murcia

Hasta hace bien poco, la Universidad de Murcia (UM) era la única universidad existente en la Comunidad Autónoma de la Región de Murcia, la cual tiene transferidas las competencias en materia universitaria desde el curso 1995/96. Pero esta situación cambió en el año 1998, por un lado, por la creación de la Universidad Politécnica de Cartagena (UPCT) –como fruto de una escisión de lo que hasta entonces había sido el Campus de Cartagena de la UM–, y por otro lado por la creación de la Universidad Católica San Antonio (UCAM), de carácter privado, ese mismo año.

Las titulaciones de la UPCT tienen, evidentemente, un carácter más tecnológico, por lo que no entran en conflicto con las impartidas en la UM. A pesar de ello, la competencia ha aparecido recientemente en las solicitudes de implantación de nuevos títulos relacionados con las nuevas tecnologías de la información y las comunicaciones, en el que ambas universidades buscan ámbitos de expansión. En cuanto a la UCAM, además de otras titulaciones no impartidas por la UM, también ofrece la Ingeniería Técnica en Informática de Sistemas, debido a su alta demanda. La competencia en este caso con la UM es más clara, y aún más si tenemos en cuenta que su sede está situada a poco menos de 4 kilómetros del Campus de Espinardo de la UM[8].

Por otro lado, la Región de Murcia cuenta también con un Centro Asociado de la Universidad Nacional de Educación a Distancia (UNED), situado en Cartagena. Este centro oferta los estudios de Ingeniería Técnica en Informática de Sistemas, Ingeniería Técnica en Informática de Gestión y el segundo ciclo de Ingeniería Informática. Debido al carácter no presencial de estos estudios, no consideramos que exista una competencia o conflicto con los estudios ofrecidos por la UM.

En esta sección vamos a centrarnos, obviamente, en la Universidad de Murcia, empezando por un repaso de su historia, la situación presente, organización, estudios ofertados, y las perspectivas de futuro. La mayor parte de la información contenida en esta sección ha sido extraída del libro "Universidad de Murcia. Pasado, presente y futuro" [HistoriaUM'98], de la Guía de la UM [GuíaUM'03] y de la Guía de Titulaciones de la UM [TitulUM'03].

Antecedentes históricos

En el léxico medieval, *Universidad* equivale a pluralidad o conjunto de personas. La creación de las primeras universidades en España data del siglo XIII, siendo la de Palencia, en 1208, la primera de ellas. Pero estas instituciones pioneras de la educación superior no son creadas por disposiciones

[8] No obstante, preferimos pensar en la relación entre ambas instituciones en términos de coexistencia y respeto mutuo; como veremos más adelante, ya han tenido lugar algunas actuaciones conjuntas que –si bien no de excesivo calado académico– dan muestra del buen clima existente.

políticas, sino que *"surgen de la misma entraña social"* [Gonzalo'62]. Esto ocurre con la Universidad de Murcia, cuya primera fundación se fija en el 6 de abril de 1272. Esta fecha –que figura en el escudo de la Universidad de Murcia, como se muestra en la figura 2.3– corresponde realmente con un documento expedido por el rey Alfonso X el Sabio, por el que concede a los Dominicos unas casas y huerta en la partida de la Arrixaca.

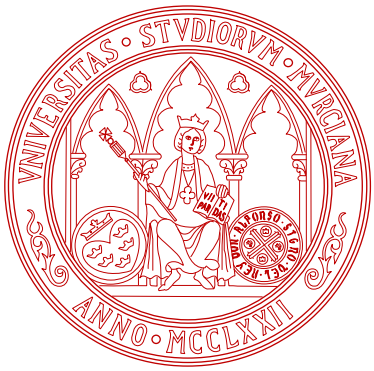


Figura 2.3: Escudo de la Universidad de Murcia, con la leyenda: UNIVERSITAS STUDIORUM MURCIANA - ANNO MCCLXXII.

Sea como fuere, la fundación de la primera Universidad de Murcia se ha vinculado tradicionalmente a Alfonso X el Sabio. En mayo de 1243, después del tratado con Ibn Hud, el entonces Infante de Castilla don Alfonso hizo su entrada en Murcia, cuyo reino tardaría aún dos años en pacificar. Una de sus primeras preocupaciones –y en la que pondría todo su entusiasmo– fue mantener y aprovechar el alto nivel cultural existente en el Sureste de la península. Durante los últimos años de la dominación musulmana, Murcia había dominado sobre los demás reinos de taifas, trasladándose a vivir en ella numerosos sabios. Entre ellos se encontraba Mohamed ibn Ahmed Abubequer Al-Ricotí, en quien don Alfonso encontró un aliado para sus propósitos. Le concedió numerosos privilegios y le construyó un estudio, donde durante muchos años impartió sus enseñanzas de Medicina, Geometría, Lógica y Filosofía, utilizando indistintamente árabe, latín y romance. El espíritu de convivencia de aquella institución, en la que todos contribuían a la difusión del conocimiento sin distinción de raza y religión, es aún hoy en día –o "especialmente" hoy en día– un ejemplo digno de mención.

Pero la situación no tardaría en cambiar. Tras la sublevación de los mudéjares, en 1266, se impuso la dominación castellana y la pérdida de los privilegios para los que antes entonces habían sido los promotores de la cultura en Murcia. Muchos marcharon a tierras granadinas, Al-Ricotí entre ellos.

La primera fundación cristiana de enseñanzas superiores corresponde a la Orden de Predicadores, que se instalaron en Murcia poco después de la reconquista. Sobre los años 1252 ó 1253 establecieron en Murcia un Estudio o Escuela de Artes y Filosofía, donde se cursaban grados inferiores de Artes, y estudios de gramática, retórica y lógica. Entre las figuras más destacadas al frente de esta institución se encontraba el padre dominico Ramón Martí, discípulo del gran científico cristiano San Alberto Magno[9]. Desafortunadamente, el esplendor cultural que vivió Murcia en aquellos años fue decayendo poco a poco, debido a la escasez de pobladores y al aislamiento que sufrió la región.

Durante los siglos siguientes, los centros de enseñanza existentes estaban asociados a las diferentes órdenes religiosas asentadas en la región, franciscanos, jesuitas y dominicos. Destaca el Seminario

[9]Que, dicho sea de paso, es el patrón de nuestra Facultad de Informática, celebrándose su festividad el día 15 de noviembre.

de San Fulgencio, que llegaría a lograr el privilegio para otorgar títulos superiores. Pero la falta de medios y la proximidad de la Universidad de Orihuela eran factores que dificultaban la creación de una universidad en Murcia.

No es hasta 1840 cuando vuelve a surgir el propósito de crear una universidad en Murcia. Ese año se funda la Universidad Literaria de Murcia, con cargo a los bienes del Instituto de Segunda Enseñanza instalado en el Colegio de Teólogos de San Isidoro (actual sede del Instituto Licenciado Cascales), que al mismo tiempo suprimía. Pero su breve vida iba a ser de apenas unos meses, los comprendidos entre su inauguración el 22 de octubre de 1840 y su supresión el 15 de mayo de 1841. La falta de medios económicos era clara, y el gobierno liberal de la época prefería utilizar los fondos en una buena red de centros de enseñanza secundaria, que compitiese con los centros eclesiásticos.

El siguiente intento de creación de una universidad no tardó mucho tiempo en producirse, aunque teniendo también una corta existencia. Tras la Revolución de 1868, se establece la libertad de enseñanza que autorizaba a Diputaciones y Ayuntamientos a crear universidades, siempre que las financiasen con cargo a sus fondos. Al amparo de esta situación, se crea la Universidad Libre de Murcia el año siguiente, inaugurándose el curso el 11 de noviembre de 1869. Esta universidad ofrecía los estudios de Derecho, con accesoria de Notariado, Ciencias y Filosofía y Letras. Pero el bajo número de estudiantes (en el curso 1870/71 habían en total 254) y las dificultades económicas volvieron a dar al traste con esta experiencia. Tras importantes reducciones en los estudios y la acumulación de numerosas deudas, se suprime la Universidad Libre de Murcia en julio de 1874.

La actual fundación de la Universidad de Murcia

Tendrían que pasar más de cuarenta años para que Murcia pudiese contar de nuevo con una Universidad; inicialmente siguiendo el patrón de las universidades libres, para luego ser absorbida por el Estado y convertirse en una universidad oficial. Teniendo en cuenta la fecha de su fundación, 1915, la actual Universidad de Murcia es la undécima de las españolas. Gracias a una considerable campaña de presión política, periodística e incluso popular, liderada por el diario "El Liberal", y a las gestiones de los ministros murcianos García Alix y Juan de la Cierva Peñafiel, y su hermano Isidoro, el 17 de diciembre de 1914 se aprueba la disposición legal por la que se crea la Universidad de Murcia. El 7 de octubre de 1915 tiene lugar, en medio de una gran solemnidad, la inauguración del curso académico 1915/16, que es al mismo tiempo la inauguración de la propia Universidad. Ubicada inicialmente en el edificio del actual Instituto Licenciado Cascales, ofrecía los estudios de Licenciatura de Derecho y unos cursos preparatorios para las facultades de Filosofía y Letras, y Medicina y Farmacia.

El Rector de nuestra Universidad durante esta primera etapa fue don José Loustau y Gómez de Membrillera, que ocuparía el cargo hasta el final de la Guerra Civil, en 1939. Su etapa coincide con importantes eventos para la institución, marcados por los intentos de supresión, la precariedad económica y la guerra. Pero gracias a su imaginativa y acertada gestión, el Rector Loustau consiguió consolidar la Universidad de Murcia, quedando su nombre ligado a la Universidad. Entre 1920 y 1935, la UM cambió de localización, siendo situada en la sede actual del Instituto del Carmen. Y al inaugurarse el curso 1935/36, la Universidad, tras complejas gestiones de compra, se traslada al edificio contiguo a la iglesia de la Merced, que actualmente constituye el Campus de la Merced.

La andadura de nuestra Universidad empieza con un número relativamente alto de estudiantes, cifrado en 1213 en el curso 1916/17, aunque sólo 100 de ellos eran oficiales y el resto libres. Pero este crecimiento no se mantiene, y desde 1939 hasta 1975 la UM sufre un periodo de estancamiento, frente al crecimiento de otras universidades españolas. La falta de prestigio de nuestra institución provoca el traslado de muchos de sus catedráticos a otras universidades, que les ofrecían mejores condiciones para su labor. El número de alumnos matriculados es muy bajo y no aparecen nuevas enseñanzas,

salvo la creación de la Facultad de Medicina en 1968. Las cifras de alumnos matriculados son muy significativas: 1036 alumnos en 1940; prácticamente la misma que en 1961, de 1117 alumnos. Durante esta etapa se suceden dos Rectores, Jesús Mérida, desde 1939 hasta 1944, y Manuel Batlle Vázquez, desde 1944 hasta 1975.

Con la llegada de la democracia a España, en 1975, se abre una nueva etapa para la Universidad de Murcia, marcada por un gran dinamismo y expansión a todos los niveles: se multiplica el número de alumnos, de titulaciones, especialidades, centros de investigación, sedes, infraestructuras, equipamientos, etcétera. Ante este aumento, las dependencias del Campus de la Merced se muestran claramente insuficientes, y a finales de los setenta la Universidad adquiere terrenos en Espinardo (pedanía murciana situada a unos 3 kilómetros de la capital), que se acabarían convirtiendo en el actual Campus de Espinardo. Los órganos de gobierno de nuestra universidad tampoco han sido ajenos a este mayor dinamismo, sucediéndose los rectores: Francisco Sabater García (1975-1980), José Antonio Lozano Teruel (1980-1984), Antonio Soler Andrés (1984-1990), Juan Roca Guillamón (1990-1994), Juan Monreal Martínez (1994-1998) y José Ballesta desde 1998 hasta la actualidad.

El presente de la Universidad

La Universidad de Murcia es, hoy por hoy, el principal centro de difusión de la cultura, la ciencia y la tecnología en la Región de Murcia, y uno de los mejores motores de su desarrollo social, científico y económico. El espectacular crecimiento que ha experimentado en los últimos quince años es un reflejo de su renovado prestigio: el número de escuelas y facultades se ha multiplicado por cinco, mientras que el número de estudiantes lo ha hecho por cuatro, pasando de 8.000 a más de 32.000 en el curso 1998/99. En proporción similar se han potenciado los recursos académicos e investigadores de la comunidad universitaria; así como otros aspectos no académicos pero necesarios para el desarrollo personal de los integrantes de una Universidad moderna: actividades culturales, instalaciones deportivas, Colegios Mayores, voluntariado, etc.

En la actualidad[10] la Universidad de Murcia cuenta con 15 facultades, 4 escuelas universitarias y un centro asociado. Estos centros, a excepción de los adscritos, están localizados en los dos campus de la Universidad: el Campus de Espinardo y el Campus de La Merced. La distribución de los centros por campus es la siguiente:

- **Campus de La Merced.**
  - Facultad de Derecho
  - Facultad de Letras
- **Campus de Espinardo.**
  - E.U. de Enfermería de Murcia
  - E.U. de Trabajo Social
  - Facultad de Bellas Artes
  - Facultad de Biología
  - Facultad de Ciencias del Trabajo
  - Facultad de Ciencias de la Documentación

[10]Datos referidos al comienzo del curso 2003/04. Como ya hemos mencionado, estos datos han sido extraídos de la Guía de la Universidad de Murcia, [GuíaUM'03].

- Facultad de Economía y Empresa
- Facultad de Derecho
- Facultad de Educación
- Facultad de Filosofía
- Facultad de Informática
- Facultad de Letras
- Facultad de Matemáticas
- Facultad de Medicina
- Facultad de Psicología
- Facultad de Química
- Facultad de Veterinaria

- **Centros asociados.**
  - E.U. de Enfermería de Cartagena
  - E.U. de Turismo
  - Centro Pérez de Lema de Enseñanza Universitaria de Cartagena

En estos centros desempeñan su labor 1874 profesores, repartidos entre 74 departamentos y 287 grupos de investigación, además de 1008 miembros del cuerpo de personal de administración y servicios. Por otro lado, durante el curso 2002/03, el número de alumnos de la Universidad fue de 28.177, notándose un ligero descenso respecto a otros años y respecto al máximo de 32.443 alcanzado el curso 1998/99. Este descenso puede ser debido a los dos hechos mencionados al principio de esta sección: la creación de la UPCT y de la UCAM el año 1998.

La oferta de titulaciones ofertadas por la Universidad de Murcia abarca las áreas de: Ciencias de la Salud, Ciencias Experimentales, Humanidades, Ciencias Sociales y Jurídicas, y Enseñanzas Técnicas. La oferta está compuesta por 19 estudios de primer ciclo (Diplomaturas e Ingenierías Técnicas), 11 estudios de segundo ciclo (Licenciaturas e Ingenierías) y 25 estudios de primer y segundo ciclo. Los títulos agrupados por área de conocimiento son los siguientes:

- **Titulaciones de Ciencias de la Salud**
  - Diplomaturas:
    - Diplomado en Enfermería
    - Diplomado en Fisioterapia
  - Licenciaturas de 1er y 2^{o} ciclo:
    - Licenciado en Medicina
    - Licenciado en Odontología
    - Licenciado en Veterinaria

- **Titulaciones de Ciencias Experimentales**
  - Diplomaturas:
    - Diplomado Óptica y Optometría

- Licenciaturas de 1er y 2º ciclo:
  - Licenciado en Biología
  - Licenciado en Ciencias Ambientales
  - Licenciado en Física
  - Licenciado en Matemáticas
  - Licenciado en Química
- Licenciaturas de 2º ciclo:
  - Licenciado en Bioquímica
  - Licenciado en Ciencia y Tecnología de los Alimentos

▪ **Titulaciones de Humanidades**

- Licenciaturas de 1er y 2º ciclo:
  - Licenciado en Bellas Artes
  - Licenciado en Filología Clásica
  - Licenciado en Filología Francesa
  - Licenciado en Filología Hispánica
  - Licenciado en Filología Inglesa
  - Licenciado en Filosofía
  - Licenciado en Geografía
  - Licenciado en Historia
  - Licenciado en Historia del Arte

▪ **Titulaciones de Ciencias Sociales y Jurídicas**

- Diplomaturas:
  - Diplomado en Biblioteconomía y Documentación
  - Diplomado en Ciencia Empresariales
  - Diplomado en Educación Social
  - Diplomado en Gestión y Administración Pública
  - Diplomado en Relaciones Laborales
  - Diplomado en Trabajo Social
  - Diplomado en Turismo
  - Maestro. Especialidad Educación Especial
  - Maestro. Especialidad Educación Física
  - Maestro. Especialidad Educación Infantil
  - Maestro. Especialidad Educación Musical
  - Maestro. Especialidad Educación Primaria
  - Maestro. Especialidad Lengua Extranjera (Francés)
  - Maestro. Especialidad Lengua Extranjera (Inglés)
- Licenciaturas de 1er y 2º ciclo:
  - Licenciado en Administración y Dirección de Empresas
  - Licenciado en Derecho

    - Licenciado en Economía
    - Licenciado en Pedagogía
    - Licenciado en Psicología
    - Programa de Estudios Simultáneos de Licenciado en Administración y Dirección de Empresas y de Licenciado en Derecho
  - Licenciaturas de 2º ciclo:
    - Licenciado en Ciencias Políticas y de las Administración
    - Licenciado en Ciencias del Trabajo
    - Licenciado en Comunicación Audiovisual
    - Licenciado en Documentación
    - Licenciado en Investigación y Técnicas de Mercado
    - Licenciado en Periodismo
    - Licenciado en Psicopedagogía
    - Licenciado en Publicidad y Relaciones Públicas
    - Licenciado en Sociología

- **Titulaciones de Enseñanzas Técnicas**
  - Ingenierías Técnicas:
    - Ingeniero Técnico en Informática de Gestión
    - Ingeniero Técnico de Informática de Sistemas
  - Ingenierías Superiores:
    - Ingeniero en Informática
    - Ingeniero Químico

Se puede observar que la oferta se concentra principalmente en el área de Ciencias Sociales y Jurídicas, la cual cuenta con en torno al 58 % de los alumnos de nuestra universidad. En cuanto a las titulaciones técnicas, sólo se imparten las tres titulaciones existentes de informática y la ingeniería química. Hay que tener en cuenta que en el mapa de titulaciones a nivel regional, la mayoría de las carreras de este área corresponden a la Universidad Politécnica de Cartagena.

Además de las facultades y escuelas universitarias, la UM dispone de otros centros para la investigación y la promoción profesional de sus alumnos, cuya creación está regulada en la legislación vigente. Estos centros son de tres tipos:

- **Institutos Universitarios de Investigación.** Se trata de centros dedicados a la investigación científica y técnica o a la creación artística, con capacidad para organizar y desarrollar programas y estudios de doctorado y de postgrado según los procedimientos previstos en los Estatutos. La UM dispone actualmente de los siguientes Institutos de Investigación:
  - Instituto de Ciencias de la Educación (ICE).
  - Instituto Universitario del Agua y del Medio Ambiente (INUAMA).
  - Instituto del Próximo Oriente Antiguo (IPOA).
- **Institutos Universitarios Propios.** Estos Institutos Propios son estructuras internas de la Universidad, que nacen con la finalidad de fomentar actividades de investigación de tipo interdisciplinar o de alta especialización científica, técnica o artística, en campos que sobrepasan el ámbito de los Departamentos. Los Institutos Propios con los que cuenta la UM son los siguientes:

  - Instituto de Ciencias del Deporte.
  - Instituto del Envejecimiento.
  - Instituto Universitario Propio de Estudios Fiscales y Financieros (IUEFF).
  - Instituto Universitario Propio de Cooperación al Desarrollo.

- **Escuelas Profesionales.** Son centros de especialización profesional, cuyo objetivo es complementar la labor formativa de los centros oficiales –en colaboración con las asociaciones profesionales correspondientes–, proporcionando a los titulados interesados en ampliar su formación un conocimiento suficiente para el desempeño de las diversas actividades implícitas en el ejercicio profesional específico. Las Escuelas Profesionales que existen actualmente en la UM son:

  - Escuela de Práctica Enfermera.
  - Escuela de Práctica Jurídica.
  - Escuela de Práctica Laboral.
  - Escuela de Práctica Psicológica.
  - Escuela de Práctica Social.
  - Escuela de Práctica Tecnológica.

Destacamos, por su especial relación con los estudios ofertados en la Facultad de Informática, la **Escuela de Práctica Tecnológica**. Su misión es ofrecer una formación avanzada y complementaria a la recibida en los estudios oficiales, para ingenieros en cualquiera de las ramas de las tecnologías de la información y las comunicaciones (TIC) y para profesionales afines. Esta Escuela se crea el 16 de mayo de 2000, gracias a un convenio entre la UM y la Asociación de Empresas en Tecnologías de la Información de la Región de Murcia (TIMUR). Entre las actividades que contemplan sus estatutos están: la realización de cursos de formación de postgrado, perfeccionamiento y reciclaje profesional; cursos de formación extracurricular para alumnos; seminarios, jornadas, reuniones y coloquios; y fomento de las relaciones con instituciones, organizaciones, empresas y profesionales de las TIC. Sin embargo, hay que reconocer que la implantación actual de la Escuela de Práctica Tecnológica es muy escasa, y de las actividades formativas planificadas para el curso 2002/03, ninguna fue llevada a cabo por no haber alcanzado el número necesario de asistentes para su realización[11].

Finalmente, para ayudar a los miembros de la comunidad universitaria en la realización de sus tareas, existe una serie de servicios de apoyo. Estos servicios se pueden clasificar en tres tipos: servicios a la comunidad universitaria, servicios de gestión y administración general, y servicios de apoyo a la investigación.

- **Servicios a la comunidad universitaria**: Servicio de Información Universitario (SIU), Centro de Orientación e Información al Empleo (COIE), Servicio de Actividades Deportivas, Servicio de Asesoramiento y Orientación Personal, Servicio Universitario de Voluntariado, Servicio de Actividades Culturales, Unidad para la Calidad, Promoción Educativa y Enseñanza Extracurricular, Servicio de Informática, Servicio de Comedores y Cafeterías, Servicio de Publicaciones, Servicio de Relaciones Internacionales y Defensor del Universitario.

- **Servicios de gestión y administración general**: Asesoría Jurídica, Área de Recursos Humanos y Servicios Generales, Inspección de Servicios, Área de Control Interno, Servicio de

[11] Podemos, no obstante, extraer la conclusión positiva de que esta situación es debida a la alta *empleabilidad* de nuestros titulados en informática, que les hace innecesaria la realización de estos cursos para encontrar empleo.

Planificación, Infraestructuras y Mantenimiento, Servicios Sociales y Asistenciales, Centro de Recursos Audiovisuales, Servicio de Gestión Académica, Área de Gestión Económica, Servicio de Relaciones Institucionales, Unidad de Comunicación e Imagen, Gabinete de Prensa, Área de Contratación, Patrimonio y Servicios, y Servicio de Prevención.

- **Servicios de apoyo a la investigación**: Servicio de Apoyo a las Ciencias Experimentales (SACE), Biblioteca Universitaria, Oficina para la Transferencia de Resultados de Investigación (OTRI) y Servicio de Gestión de Investigación.

Marco legal, organizativo y estatutario

El documento que rige el funcionamiento de una Universidad son sus Estatutos, donde se desarrollan y completan todos los aspectos de la ley de nivel superior (la LRU hasta ahora, o la LOU desde su aprobación). Los Estatutos son elaborados por la misma Universidad, como consecuencia de su autonomía de gobierno, y deben ser aprobados por el Claustro Universitario y por el Gobierno de la Comunidad Autónoma correspondiente. Los actuales Estatutos de la UM citan entre sus fines principales la creación, transmisión y difusión del conocimiento científico, técnico y cultural, *"con atención singular a las demandas particulares de la Región de Murcia"*; se concretan cuestiones como los porcentajes en la composición de los órganos colegiados, sus funciones y proceso de elección; se regulan aspectos como la gestión de los servicios universitarios y los mecanismos de evaluación del profesorado; y también se crean nuevos órganos, como el Secretario de Departamento, encargado de la gestión económica, y la Junta de Departamento, encargada de ejecutar los acuerdos y realizar las tareas que le asigne el Consejo de Departamento.

La estructura organizativa de la UM, de acuerdo con sus actuales Estatutos, está compuesta por todos los órganos colegiados y unipersonales regulados por la LRU y los que la propia Universidad añade:

- **Colegiados:** Consejo Social, Claustro Universitario, Junta de Gobierno, Juntas de Facultad o Escuela, Consejos y Juntas de Departamento, y Consejos de Instituto Universitario.
- **Unipersonales:** Rector, Vicerrectores, Secretario General, Gerente, Vicegerentes, Decanos de Facultad y Directores de Escuela, Vicedecanos de Facultad y Subdirectores de Escuela, Secretario de Centro, Directores y Secretarios de Departamento, y Directores de Instituto Universitario.

Los vicerrectorados con los que cuenta la UM son mostrados en la tabla 2.1.

En este momento, al comienzo del año 2004, la Universidad de Murcia está inmersa en el proceso de reforma de sus actuales Estatutos, con el objetivo de adaptarlos al nuevo contexto creado por la LOU. En noviembre de 2003 se publicó un borrador de los nuevos Estatutos de la UM[12], para propiciar la discusión, debate y propuesta de modificaciones por parte de la comunidad universitaria. Entre las principales reformas, respecto a los actuales Estatutos, están aquellas derivadas de la definición de nuevos órganos de gobierno: Consejo de Gobierno, Junta Consultiva, Consejo de Dirección y Defensor del Universitario. También hay algunos cambios menores, como la elección del Secretario del Departamento por parte del Director en lugar de en el Consejo de Departamento, y otros de mayor trascendencia que provocan algunas opiniones enfrentadas: la modificación de los porcentajes de composición de los órganos de gobierno, la posible limitación temporal de los cargos unipersonales, y la supresión de las convocatorias de diciembre. Igualmente, está en discusión la inclusión entre las funciones de los Centros y los Departamentos de: *"velar por la calidad de la docencia"* impartida por el Centro o el Departamento correspondiente.

[12] Accesible en: `http://www.um.es/estructura/organos/claustro/p-estatutos`

Vicerrectorado	Función
Calidad y Convergencia Europea	Encargado de la gestión de la calidad y de la integración de la UM en el Espacio Europeo
Estudiantes y Empleo	Promociona la participación de los estudiantes y el empleo
Estudios y Postgrado	Atiende la oferta docente oficial y de títulos propios
Extensión Cultural y Proyección Universitaria	Proyecta la Universidad de Murcia en su entorno social, tanto en el ámbito local como en el internacional
Investigación y Nuevas Tecnologías	Apoyo, fomento y difusión de la labor investigadora desarrollada por los grupos de investigación
Planificación e Infraestructuras	Gestiona las actividades relacionadas con el equipamiento y patrimonio de la Universidad
Profesorado y Formación	Dedicado a la gestión del profesorado y su formación

Tabla 2.1: Vicerrectorados existentes en la Universidad de Murcia y sus funciones.

2.1.4. La Facultad de Informática

A pesar de su relativamente reciente creación, la Facultad de Informática de la Universidad de Murcia (FIUM) es, por número de alumnos matriculados en el curso 2002/03, la sexta mayor facultad de las 15 con las que cuenta la UM. El número de alumnos, 1725, supera al de otras facultades de mayor tradición, como Medicina o Biología, y se acerca mucho al de la Facultad de Química. Dentro de su ámbito de influencia, primero a través de la Escuela Universitaria y después con la Facultad, la UM ha participado en la revolución que han provocado los ordenadores en nuestra sociedad. En sus clases se han formado muchos de los especialistas que han contribuido al desarrollo de la sociedad de la información en la Comunidad Autónoma de la Región de Murcia.

Para la elaboración de este apartado se ha utilizado, fundamentalmente, la Guía de la Facultad de Informática, curso 2003/04, [GuíaFIUM'03][13].

Breve historia de la Facultad de Informática

La aparición de las primeras facultades de informática en España tiene lugar en el año 1976, con la publicación del Decreto 593/1976, de 4 marzo, por el que se crean las facultades de informática en Barcelona, Madrid y San Sebastián. La Universidad de Murcia no se quedó atrás, y mostró pronto un gran interés por los temas relativos a las tecnologías de la información, previendo el papel clave de estas tecnologías para el desarrollo de la región y la integración de nuestro país en la estructura científica de la Comunidad Económica Europea.

En consecuencia, la Universidad de Murcia propuso la creación de los estudios de la Diplomatura de Informática, y en junio 1982 se crea –por el Real Decreto 1469/1982– la **Escuela Universitaria de Informática de Murcia**. Ese mismo año, el Ministerio de Educación y Ciencia autoriza el inicio de actividades para el curso académico 1983/84, con la impartición de la titulación "Diplomado en Informática". La Universidad de Murcia fue, por lo tanto, una de las primeras universidades españolas en impartir estudios de informática, demostrando una buena visión de futuro e interés por una disciplina que estaba en sus inicios. Los poco más de veinte años de historia de la Escuela, y después Facultad, son un reflejo de la rápida y continua evolución de la disciplina de la informática en ese mismo periodo de tiempo. En la figura 2.4 se muestra gráficamente y de forma resumida la historia

[13] Este curso, por primera vez, la Guía no se publica en papel, sino exclusivamente en formato electrónico.

de la FIUM.

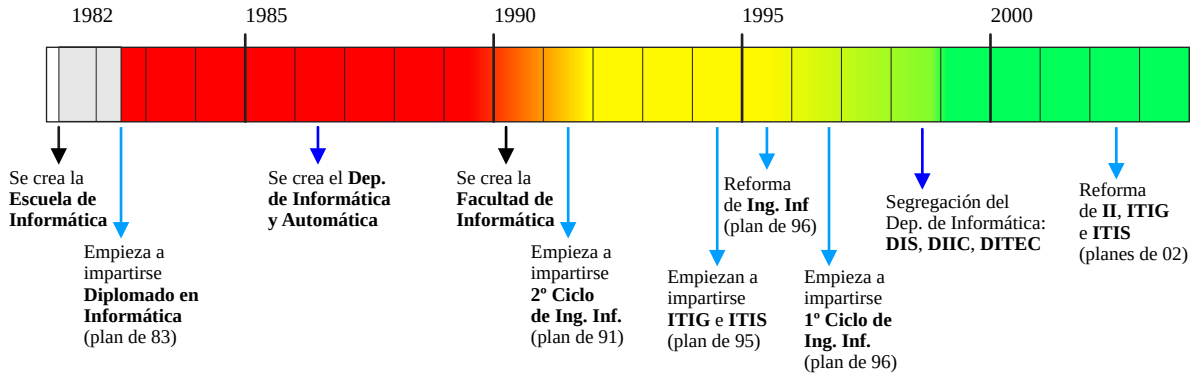


Figura 2.4: Los principales acontecimientos en la historia de la Facultad de Informática de la Universidad de Murcia.

En los primeros años tras su creación, muchos estudiantes de nuestra universidad decidían continuar sus estudios de segundo ciclo en otras universidades –principalmente Málaga, Granada y la Politécnica de Valencia–. Esto hizo evidente la necesidad de impartir este título en la UM. En respuesta a esta situación, se crea en 1990 la **Facultad de Informática de Murcia** –por el Real Decreto 1049/1990, de 27 de julio (BOE de 10 de agosto)– siendo autorizado el inicio de actividades para el curso 1991/92. Ese año se empieza a impartir el segundo ciclo de "Ingeniero en Informática", de dos años. En mayo de 1993, la Junta de Gobierno de la UM aprueba la integración de la Escuela Universitaria en la Facultad de Informática, que sería la encargada de impartir, a partir de ese momento, todas las titulaciones relacionadas con la informática.

En noviembre de 1990, de acuerdo con el desarrollo normativo de la LRU –como vimos en la página 39 del apartado 2.1.1– se aprueban las directrices generales propias sobre planes de estudios de las titulaciones de informática, en los Reales Decretos 1459/1990, 1460/1990 y 1461/1990, de 26 de octubre. Se establecen los títulos oficiales de "Ingeniero en Informática" (II), "Ingeniero Técnico en Informática de Gestión" (ITIG) e "Ingeniero Técnico en Informática de Sistemas" (ITIS), que sustituyen a las dos titulaciones existentes de Diplomatura y Licenciatura en Informática. En aplicación de estas directivas, la FIUM desarrolla los nuevos planes de estudios, y en el curso 1994/95 se sustituyen por las titulaciones antes referidas: las dos de grado medio, ITIG e ITIS; y la de grado superior, II. Aunque la titulación de II ya se impartía desde el curso 1991/92, la aplicación de las nuevas directrices requirió de la elaboración de unos nuevos planes de estudios para II, que empezaron a impartirse en el curso 1995/96.

Ante las buenas expectativas de crecimiento, y tras varios años de rodaje, la FIUM consideró la posibilidad de incorporar a su oferta el primer ciclo de II. De este modo, la Junta de Gobierno de la UM acordó solicitar el primer ciclo de esta titulación que, una vez concedido, empezó a impartirse en el curso 1996/97. Por tanto, la FIUM imparte actualmente las tres titulaciones de informática incluidas en el catálogo nacional de títulos universitarios.

Durante el curso 2001/02 la FIUM emprende otro proceso de reforma de planes de estudios[14], motivado por una serie de deficiencias encontradas en los planes anteriores, planteándose como objetivos: la eliminación de la organización semestre/trimestre, recuperación de las asignaturas anuales, aumento de la optatividad, renovación de contenidos, etc. El proceso culmina con la aprobación de los

[14] Analizaremos estas reformas en profundidad como parte del análisis del contexto curricular, sección 2.2.

nuevos planes para las tres titulaciones, II, ITIG e ITIS, y su implantación a partir del curso 2002/03, para los primeros cursos. Durante el curso 2003/04 se están impartiendo los planes nuevos en II, y en los dos primeros cursos de ITIG e ITIS; y finalmente en el curso 2004/05 se impartirá el nuevo plan para el tercer curso de las titulaciones técnicas. La tabla 2.2 muestra de forma resumida todas las implantaciones y modificaciones de los planes de estudios de la FIUM.

	Diplomado en Informática
BOE 14-9-1983	Se hace público el plan de estudios, que comienza a impartirse en el curso 1983/84
	Ingeniero Técnico en Informática de Gestión
BOE 14-2-1995	Se hace público el plan de estudios, que comienza a impartirse en el curso 1994/95
BOE 6-4-1996	Modificación del plan de estudios
BOE 25-11-1997	Modificación del plan de estudios
BOE 30-1-2003	Se hace público el plan de estudios, que comienza a impartirse en el curso 2002/03
	Ingeniero Técnico en Informática de Sistemas
BOE 14-2-1995	Se hace público el plan de estudios, que comienza a impartirse en el curso 1994/95
BOE 6-4-1996	Modificación del plan de estudios
BOE 27-6-1998	Modificación del plan de estudios
BOE 14-12-2001	Modificación del plan de estudios
BOE 30-1-2003	Se hace público el plan de estudios, que comienza a impartirse en el curso 2002/03
	Ingeniero en Informática
BOE 30-4-1993	Se hace público el plan II (2º ciclo), que se imparte desde el curso 1991/92
BOE 17-4-1997	Se hace público el plan II (2º ciclo), que se imparte desde el curso 1995/96
BOE 26-3-1998	Se público el plan de estudios de II (1er ciclo), que comienza a impartirse en el curso 1996/97
BOE 12-2-1999	Modificación de II (2º ciclo)
BOE 23-2-2002	Se unifican los dos ciclos en la titulación de Ingeniero en Informática
BOE 30-1-2003	Se hace público el plan de estudios, que comienza a impartirse en el curso 2002/03

Tabla 2.2: Implantación y reforma de los planes de estudios impartidos en la FIUM.

Como vimos en la sección 2.1.2, no pasará mucho tiempo hasta que la Facultad deba abordar otra nueva reforma de sus planes de estudios, derivados de la integración de España en el Espacio Europeo de Educación Superior. Para el 2010, como máximo, deberán estar adaptadas todas las titulaciones que, posiblemente, no serán las mismas (ya comentamos que el Ministerio ha mostrado su interés por que se agrupen titulaciones relacionadas) ni tendrán la misma duración ni número de créditos.

Situación actual de la Facultad

En la actualidad, los alumnos que entran en la Facultad de Informática de la UM pueden optar entre realizar una Ingeniería Técnica, de tres años de duración, o la Ingeniería Informática, de cinco años. Son también muchos los estudiantes que tras completar la ITIG o ITIS, deciden continuar sus estudios con el segundo ciclo de II.

De acuerdo con la Memoria Académica del curso 2002/03 de la UM [MemoriaUM'02], durante ese curso el número de alumnos matriculados en las carreras de la FIUM fue de 1710. Estos alumnos se reparten de forma más o menos equitativa entre las tres titulaciones: 642 en II, 532 en ITIG, y 536 en ITIS. De estos alumnos, el 25 % eran de nuevo ingreso, distribuyéndose de forma similar entre las tres carreras. También el número de egresados por año es relativamente homogéneo, y suele ser de unos 30 a 40 alumnos por curso y titulación. En concreto, en el curso 2002/03 fueron expedidos 41 títulos de la Ingeniería Informática y 69 de la Ingenierías Técnicas. En la dimensión temporal, el número de alumnos ha crecido de manera más o menos uniforme durante los últimos cinco años, en unos porcentajes promedios en torno al 5 % anual, si bien el presente curso ha habido un ligero descenso.

En cuanto a la parte docente, son un total de doce departamentos los que imparten docencia en la FIUM. Los departamentos, de mayor a menor número de profesores asociados a la FIUM, son los siguientes:

- Informática y Sistemas (DIS).
- Ingeniería de la Información y las Comunicaciones (DIIC).
- Ingeniería y Tecnología de Computadores (DITEC).
- Matemática Aplicada (DMA).
- Física (DFIS).
- Estadística e Investigación Operativa (DESTIO).
- Organización de Empresas (DOE).
- Economía Financiera y Contabilidad (DEFC).
- Información y Documentación (DINFODOC).
- Derecho Civil (DDC).
- Comercialización e Investigación de Mercados (DCIM).
- Filosofía (DFIL).

Aunque son los tres primeros, DIS, DIIC y DITEC, los más relacionados con la informática, los que tienen adscritos un mayor número de profesores y se encargan de la mayor parte de la docencia. En concreto, estos tres departamentos suman el 74 % del más de un centenar de profesores de la FIUM. La distribución de los profesores por departamento y por categoría profesional, al finalizar el curso 2002/03, es mostrada en la tabla 2.3.

Analizando los datos de la tabla 2.3, podemos ver que hay un cierto equilibrio entre el número de profesores de los Cuerpos Docentes (CU, TU, CEU y TEU) que constituyen un 51 %, y el número de profesores contratados (AY y ASOC) con un 49 %. No obstante, hay que tener en cuenta que estas

Departamento	CU	TU	CEU	TEU	AY	ASOC	TOTAL
DIS		6	1	13	4	7	**31**
DIIC	2	10		3	5	11	**31**
DITEC		1		4	9	11	**25**
DMA	1	5		2	3	3	**14**
DFIS		3	1	1			**5**
DESTIO		3		1			**4**
DOE		1		1			**2**
DEFC					1	1	**2**
DINFODOC		1					**1**
DDC						1	**1**
DCIM						1	**1**
TOTAL	**3**	**30**	**2**	**25**	**22**	**35**	**117**
Porcentaje	**2,5%**	**25,6%**	**1,7%**	**21,4%**	**18,8%**	**29,9%**	**100%**

Tabla 2.3: Profesores con docencia en la Facultad de Informática de la UM durante el curso 2002/03, por departamento y categoría profesional. Categorías: Catedrático de Universidad (CU), Titular de Universidad (TU), Catedrático de Escuelas Universitarias (CEU), Titular de Escuelas Universitarias (TEU), Ayudante (AY) y Asociado (ASOC).

cifras son muy variables, debido a las constantes demandas de los departamentos de nuevo profesorado, a la promoción de los profesores y a los cambios de adscripción en las áreas minoritarias.

En cuanto al personal de administración y servicios, la FIUM cuenta con un total de 37 empleados, entre conserjes, administrativos y encargados del centro de cálculo.

Además de las numerosas tareas relacionadas con la impartición de los títulos oficiales –y todas las necesarias para la consecución de los recursos e infraestructuras que ello implica (ordenadores y software para laboratorios de prácticas, espacio para los profesores y grupos de investigación, elaboración de planes de estudios, coordinación entre los departamentos, etc.)– la Facultad de Informática se ha mostrado muy activa en la toma de iniciativas orientadas a las otras misiones básicas de todo centro docente: extensión de la cultura a la sociedad, mejora de la formación profesional de sus estudiantes, fomento de la relación con las empresas y con su entorno. Entre las actuaciones más destacadas en este sentido, podemos mencionar las siguientes:

- Participación activa en la **Escuela de Práctica Tecnológica**. Como vimos en el apartado 2.1.3, esta Escuela fue creada gracias a un convenio de la UM con TIMUR (la asociación de empresas murcianas en tecnologías de la información). Los profesores de la FIUM son una parte importante con la que cuenta esta institución para el desarrollo de sus actividades.

- Organización de **cursos de promoción educativa**, también en colaboración con TIMUR, con el objetivo de complementar la formación recibida por los alumnos en la carrera. Estos cursos tratan de métodos, técnicas o tecnologías que quedan fuera de los planes de estudios por ser muy especializados. Son impartidos por profesionales de las empresas informáticas, y versan

sobre temas como la plataforma .NET, datawarehouse, OLAP, ERP y comercio electrónico. La Facultad organiza además otros cursos orientados a potenciar las habilidades personales de los alumnos, en aspectos en los que muestran carencias con más frecuencia: cursos de inglés, y de expresión oral y escrita. Desafortunadamente, en los años anteriores la acogida de estos cursos entre los estudiantes ha sido muy inferior a lo esperado.

- Colaboración con la ***junior*-empresa INFOMUN**. Una *junior*-empresa es una asociación de carácter no lucrativo, constituida y gestionada por estudiantes, que opera en el propio centro universitario, desde el que ofrecen sus servicios a las empresas en materias relacionadas con sus estudios. INFOMUN es la *junior*-empresa constituida por alumnos de la FIUM. Al ser no lucrativa, sus beneficios están destinados a otras actividades que complementan la formación de sus asociados.

- Tramitación de la construcción de la **nueva Facultad de Informática**. La falta de espacio físico empezó a hacerse evidente desde la implantación del segundo ciclo de II, en 1990. Los trámites desde entonces para la construcción del nuevo edificio han sido muy largos. El proyecto definitivo se aprobó a finales del 2001, pero sólo recientemente se ha colocado la primera piedra del nuevo edificio, estando comprometida su construcción en el plazo de 18 meses.

- Participación en el **Concurso Internacional de Programación**[15] para estudiantes universitarios, patrocinado por IBM y la ACM (*Association for Computer Machinery*). En el curso 2002/03 la Universidad de Murcia participó por primera vez en este concurso, que tiene como fin propiciar la relación y la competición entre alumnos de universidades de todo el mundo. Con motivo de este concurso, la Facultad organizó un curso de promoción educativa, orientado a la preparación de los estudiantes para su participación en el concurso. Entre los profesores organizadores del concurso en la FIUM, se decidió realizar una fase local del concurso (denominada "Olimpiadas Murcianas de Programación") extendiendo su ámbito a toda la Región de Murcia. De esta manera, se invitó a las otras dos universidades murcianas, la UPCT y la UCAM, fomentando el espíritu competitivo interuniversitario.

  Gracias a la financiación de la FIUM y la Fundación Séneca, los dos primeros equipos clasificados en la Olimpiada Murciana (ambos de la UM) participaron en París, en noviembre de 2003, en la fase del ACM Contest de la región del suroeste de Europa, SWERC'2003[16]. Es interés de la Facultad seguir organizando este concurso, y el curso de preparación, en los años sucesivos. Por la filosofía del concurso, su formato y el tipo de problemas propuestos, existe una relación muy estrecha con los fines de la asignatura de Algoritmos y Estructuras de Datos. Por esta razón, creemos muy conveniente fomentar la participación de los alumnos en esta actividad extracurricular.

Estructura organizativa actual de la FIUM

Para poder llevar a cabo sus tareas cotidianas de coordinación, gestión y gobierno, la Facultad de Informática de la UM cuenta con una estructura organizativa compuesta básicamente por: órganos de gobierno, órganos consultivos y órganos administrativos. Las funciones y composición actual de estos órganos son las siguientes:

[15] También conocido como *ACM Contest*. Se puede encontrar información del concurso en: `http://icpc.baylor.edu` (fase mundial), `http://www.polytechnique.edu/icpc2003` (fase del suroeste de Europa); y `http://dis.um.es/contest` (fase murciana).

[16] Diremos, a modo anecdótico, que los equipos murcianos quedaron clasificados en las posiciones 26 y 33, de las 52 universidades europeas participantes.

- **Órganos de Gobierno.**
  - **Junta de Facultad.** Es el máximo órgano representativo del Centro, encargado del gobierno y control de los fines de la Facultad de Informática. Está constituida por representantes de los profesores pertenecientes a los Cuerpos Docentes (65 %), de ayudantes, becarios y alumnos del tercer ciclo (5 %) y de alumnos matriculados en algún curso (30 %), a los que se unen 3 miembros del personal de administración y servicios[17]. Son miembros natos el Decano, que la preside, los Vicedecanos y el Secretario de Centro. En la actualidad, la Junta de Facultad de la FIUM consta de 59 miembros.
  - **Órganos unipersonales.** Los cargos existentes en la FIUM son: Decano, Vicedecanos y Secretario de Centro. La composición del equipo decanal a comienzo del curso 2003/04 era la siguiente[18]:
    - **Decano:** Jesús Joaquín García Molina.
    - **Vicedecano de Docencia y Alumnos:** Isidro Verdú Conesa.
    - **Vicedecano de Relaciones Externas:** Luis Daniel Hernández Molinero.
    - **Vicedecano de Infraestructura:** Antonio Flores Gil.
    - **Secretario:** Domingo Giménez Cánovas.
- **Órganos Consultivos o Comisiones.** Son grupos reducidos de personas, constituidos en la Junta de Facultad, y con autorización de la misma para resolver asuntos de trámite o realizar informes consultivos a petición de la Junta. La FIUM tiene actualmente las siguientes comisiones: Permanente, de Convalidaciones, de Biblioteca, de Coordinación Académica, de Actividades Culturales, de Edificio e Infraestructura, y de Actividades Extracurriculares.
- **Órganos Administrativos.** Son los encargados de llevar a cabo las labores administrativas propias de la Facultad de Informática. Se distinguen: Secretaría de la Facultad de Informática, Secretaría del Decanato y Conserjería de la Facultad.

2.1.5. El Departamento de Informática y Sistemas

De acuerdo con la LRU[19], los Departamentos son los órganos encargados de coordinar las enseñanzas de las áreas de conocimiento de que consta, de apoyar las actividades e iniciativas docentes e investigadoras del profesorado, así como de las otras funciones que sean determinadas en los Estatutos. El Departamento de Informática y Sistemas (DIS), convocante de la plaza objeto de concurso, es el resultado de un intenso proceso de reformas y segregaciones que tuvieron lugar durante la última década [Montoya'01]. Vamos a hacer un breve repaso de este proceso, para centrarnos después en su composición y situación actual.

Antecedentes del Departamento de Informática y Sistemas

Como analizamos en el apartado 2.1.1, la LRU de 1983 establecía la organización de las universidades en Departamentos y Áreas de conocimiento, frente a la antigua organización facultativa.

[17] En los nuevos estatutos de la Universidad de Murcia –aún en trámites de elaboración cuando se redactó este proyecto docente– se modifican ligeramente los porcentajes.

[18] La elección de un nuevo equipo decanal está prevista para el 26 de marzo del presente año.

[19] Y al estilo de lo dispuesto en la LOU, así como en los Estatutos de la UM y en el Reglamento del Departamento de Informática y Sistemas.

En aplicación de esta legislación, la UM crea en 1986 el **Departamento de Informática y Automática**. Este Departamento impartía docencia principalmente en la Diplomatura en Informática, en el Campus de Espinardo, y en las Escuelas de Ingeniería del Campus de Cartagena –actualmente integradas en la Universidad Politécnica de Cartagena–. La separación física entre ambos centros y la clara distinción entre los ámbitos de trabajo de sus miembros aconsejaba una separación, que se produce en el año 1993. Como resultado, surgen dos nuevos Departamentos:

- Departamento de Automática, Electricidad y Electrónica Industrial, con sede en el entonces Campus de Cartagena.
- **Departamento de Informática y Sistemas**, con sede en el Campus de Espinardo, y compuesto por las cuatro siguientes áreas de conocimiento:
  - Lenguajes y Sistemas Informáticos,
  - Ciencia de la Computación e Inteligencia Artificial,
  - Arquitectura y Tecnología de los Computadores,
  - Electrónica.

En 1998, después del gran crecimiento de todas las áreas que tuvo lugar tras la implantación en la FIUM de las tres titulaciones de informática *"y tras algunas desavenencias"* [Fernández'02], el antiguo Departamento de Informática y Sistemas vuelve a dividirse en otros tres:

- **Departamento de Informática: Lenguajes y Sistemas Informáticos**, constituido únicamente por el área de conocimiento "Lenguajes y Sistemas Informáticos".
- Departamento de Informática, Inteligencia Artificial y Electrónica, constituido por las áreas de conocimiento "Ciencia de la Computación e Inteligencia Artificial" y "Electrónica". Posteriormente, este departamento conseguiría la adscripción de nuevas áreas, y pasaría a denominarse, en el año 2000, Departamento de Ingeniería de la Información y las Comunicaciones.
- Departamento de Ingeniería y Tecnología de los Computadores, constituido por el área "Arquitectura y Tecnología de los Computadores".

En 1999, el Departamento de Informática: Lenguajes y Sistemas Informáticos, solicita y consigue la adscripción del área de conocimiento "Ingeniería de Sistemas y Automática". Finalmente, en el año 2000 el Consejo de Departamento decide el cambio de denominación, pasando a llamarse **Departamento de Informática y Sistemas**, tras la correspondiente aprobación de la Junta de Gobierno el 21 de julio de 2000.

Composición y estado actual del Departamento

Al comienzo del curso académico 2003/04, el Departamento de Informática y Sistemas de la UM contaba con 31 profesores –repartidos desigualmente entre sus dos áreas de conocimiento– y una administrativa, encargada de las tareas de gestión del Departamento. Nueve de estos profesores son doctores, lo cual representa un 29 % del total. La distribución de los profesores por categorías es mostrada en la tabla 2.4. Recordamos, no obstante, la alta variabilidad de estos datos.

La mayor parte de la docencia impartida por los profesores de DIS tiene lugar en las tres titulaciones impartidas en la Facultad de Informática: ITIG, ITIS e II. Además, el Departamento imparte docencia en las siguientes titulaciones: Diplomado en Biblioteconomía y Documentación, Licenciado

Área de conocimiento	CU	TU	CEU	TEU	AY	ASOC	Total
Lenguajes y Sistemas Informáticos		5	1	12	4	7	29
Ingeniería de Sistemas y Automática		1		1			2
Total	0	6	1	13	4	7	31

Tabla 2.4: Distribución de los profesores de DIS, por área y categoría profesional. Las categorías son las mismas que las mostradas en la tabla 2.3.

en Documentación, Diplomado en Ciencias Empresariales, Diplomado en Gestión y Administración Pública, Diplomado en Trabajo Social, Licenciado en Física, Licenciado en Ciencias Ambientales, Licenciado en Bellas Artes y Licenciado en Publicidad y Relaciones Públicas. En la mayoría de los casos se trata de asignaturas de introducción a la informática, conocimientos que por su amplia utilidad son necesarios para un número muy grande de titulados.

En cuanto a la docencia en tercer ciclo, el Departamento ha realizado un considerable esfuerzo por impartir y mantener en su oferta un programa de doctorado propio, a pesar del escaso número de alumnos[20]. El programa que imparte actualmente se denomina "Matemáticas e Informática Aplicadas en Ciencias e Ingenierías"[21]; se trata de un programa interuniversitario, en el que junto con DIS participan también el Departamento de Matemática Aplicada de la UM y el Departamento de Matemática Aplicada y Estadística de la UPCT. Este programa está dirigido especialmente a Licenciados en matemáticas, física, química e ingenieros superiores. Las líneas de investigación propuestas por DIS son: programación paralela, ingeniería de requisitos, orientación a objetos, control y robótica, verificación formal de sistemas de software y visión artificial.

Finalmente, en lo relativo a la actividad investigadora, existen oficialmente tres grupos de investigación en el Departamento:

- **Computación Científica:** Percepción Artificial, Procesamiento Paralelo y Computación Gráfica, COCI (E083-02).
- Informática Industrial (E083-03).
- Ingeniería del Software (E083-01).

Es en el primero de ellos en el que desarrolla su actividad investigadora el candidato de la presente plaza, desde su incorporación a la UM, trabajando dentro de líneas relacionadas con la percepción artificial, visión por computador, reconocimiento de patrones, interfaces perceptuales y biométricas. Considerando el importante componente algorítmico de los problemas tratados, la interacción con la docencia de la asignatura Algoritmos y Estructuras de Datos puede resultar interesante en algunos casos; por ejemplo, para la ejemplificación de algunas técnicas de diseño estudiadas en la asignatura, aunque sin profundizar más de lo estrictamente necesario.

2.2. El contexto curricular

Desde la selección de los contenidos a gran escala, hasta la concreción de los apartados que componen cada tema, uno de los factores más influyentes en la planificación de una asignatura es el

[20] Por ejemplo, el programa del bienio 2004-2006 consiguió "salvarse" *in extremis*, al encontrarse en el último momento un alumno que completara el cupo mínimo de diez, necesario para que se pudiera impartir el programa.

[21] Se puede obtener más información en: `http://www.um.es/mataplic/doctorado`

contexto curricular en el que esa asignatura tiene lugar, es decir, en el plan de estudios del que forma parte. Ninguna asignatura es una isla, sino que se fundamenta en los conocimientos adquiridos por los estudiantes en las asignaturas anteriores del plan, se imparte al mismo tiempo que otras asignaturas a las que los alumnos dedicarán sus esfuerzos simultáneamente, y se espera de ella que aporte la base para las materias con las que continuará su plan de estudios.

Como ya vimos en la introducción, la asignatura objeto de concurso, "Ampliación de Algoritmos y Estructura de Datos", aparece en el 2º curso de la titulación Ingeniero en Informática en los planes de estudios de 1996. En los planes actualmente vigentes en la FIUM, esta asignatura es sustituida por "Algoritmos y Estructuras de Datos". En esta sección analizaremos ambos planes y las relaciones de la asignatura "Ampliación de Algoritmos y Estructura de Datos" –y su *heredera* en el plan nuevo– con el resto de las asignaturas del plan. Por otro lado, no podemos dejar de hacer algunos comentarios sobre las Ingenierías Técnicas en Informática, las cuales comparten buena parte de sus contenidos con la Ingeniería Superior, y especialmente en lo que se refiere al contexto de la asignatura en estudio. Haremos también referencia al ámbito más externo del plan de estudios, formado por las directrices generales propias así como por las recomendaciones curriculares internacionales.

2.2.1. Directrices generales propias de las titulaciones de informática

Para poder ser homologados oficialmente, los planes de estudios de las titulaciones universitarias deben ajustarse a unas directrices generales –que ya estudiamos en la página 39 del apartado 2.1.1–, comunes para todas las titulaciones, y a las directrices generales propias establecidas por el Ministerio para cada titulación. Estas directrices contienen las materias troncales del plan de estudios, una breve descripción de los contenidos asociados[22], la carga mínima de cada materia y las áreas de conocimiento que están oficialmente capacitadas para impartirlas.

Las directrices generales propias de los estudios de informática aparecen en el BOE número 278 del 20 de noviembre de 1990, Reales Decretos 1459, 1460 y 1461/1990. En concreto, el RD 1459/1990 es el que se refiere a los estudios de Ingeniero en Informática. En las tablas 2.5 y 2.6 se resumen las directrices generales propias de II, ITIG e ITIS.

Debemos recordar tres aspectos en relación al carácter de estas directrices.

- Las materias no se corresponden necesariamente con asignaturas de un plan de estudios, sino que sus contenidos pueden estar repartidos entre distintas asignaturas y, a su vez, una asignatura puede tener contenidos de diferentes materias.
- Cuando no se indica el número de créditos teóricos o prácticos de forma explícita (se indica sólo el total), se entiende que el plan de estudios debe destinar a enseñanzas prácticas, bien por materias, bien como prácticas integradas, entre el 40 % y el 50 % del número total de créditos.
- Las directrices establecidas dejan un amplio margen a las Universidades para incluir en sus planes de estudios otras materias de su propia elección, como materias de carácter obligatorio u optativo. Este margen está en torno al 60 % de la carga total del plan.

Se pueden apreciar notables coincidencias en las materias troncales de las tres titulaciones: el primer ciclo de II e ITIG comparten el 76 % de los créditos, mientras que II e ITIS tienen en común el 97 %. La peculiaridad de ITIG es la introducción de materias como Ingeniería del Software y Gestión Empresarial, que ponen énfasis en una formación que permita al alumno el desarrollo de aplicaciones para las empresas y que favorezca su integración en los departamentos de Informática de cualquier

[22] Conocida normalmente como "los descriptores" de esa materia.

INGENIERO EN INFORMÁTICA, PRIMER CICLO				
Troncal y descripción	**Créditos**			**Áreas de conocimiento**
	T	**P**	**Total**	
Estadística Estadística descriptiva. Probabilidades. Métodos estadísticos Aplicados.	-	-	**6**	CCIA, EIO, MA
Estructura de Datos y de la Información Tipos Abstractos de Datos. Estructura de Datos y Algoritmo de Manipulación. Estructura de Información: Ficheros, Bases de Datos.	-	-	**12**	CCIA, LSI
Estructura y Tecnología de Computadores Unidades Funcionales: Memoria, Procesador, Periferia, Lenguajes Máquina y Ensamblador, Esquema de Funcionamiento. Electrónica. Sistemas Digitales. Periféricos.	-	-	**15**	ATC, ELEC, ISA, TELEC
Fundamentos Físicos de la Informática Electromagnetismo. Estado sólido. Circuitos.	-	-	**6**	ELEC, MA, ISA, IELEC, TELEC
Fundamentos Matemáticos de la Informática Algebra. Análisis Matemático. Matemática Discreta. Métodos numéricos.	-	-	**18**	ALG, AMAT, CCIA, MA
Metodología y Tecnología de la Programación Diseño de Algoritmos. Análisis de Algoritmos. Lenguajes de programación. Diseño de Programas: Descomposición Modular y Documentación. Técnicas de Verificación y Pruebas de programas.	-	-	**15**	CCIA, LSI
Sistemas Operativos Organización, estructura y servicios de los sistemas operativos. Gestión y administración de memoria, procesos y entrada/salida. Sistemas de Ficheros.	-	-	**6**	ATC, CCIA, LSI
Teoría de Autómatas y Lenguajes Formales Máquinas secuenciales y Autómatas finitos. Máquinas de Turing. Funciones Recursivas. Gramáticas y Lenguajes Formales. Redes neuronales.	-	-	**9**	ALG, CCIA, ISA, LSI, MA
INGENIERO EN INFORMÁTICA, SEGUNDO CICLO				
Arquitectura e Ingeniería de Computadores Arquitecturas paralelas. Arquitecturas orientadas a aplicaciones y lenguajes.	-	-	**9**	ATC, ELEC, ISA, TELEC
Ingeniería del Software Análisis y definición de requisitos. Diseño, propiedades y mantenimiento del software. Gestión de configuraciones. Planificación y gestión de proyectos informáticos. Análisis de aplicaciones.	-	-	**18**	CCIA, LSI
Inteligencia Artificial e Ingeniería del Conocimiento Heurística. Sistemas basados en el conocimiento. Aprendizaje. Percepción.	-	-	**9**	CCIA, ISA, LSI
Procesadores de Lenguaje Compiladores, Traductores e Intérpretes. Fases de Compilación. Optimización de código. Macroprocesadores.	-	-	**9**	CCIA, LSI
Redes Arquitectura de Redes. Comunicaciones.	-	-	**9**	ATC, CCIA, ISA, ITEL, LSI
Sistemas Informáticos Metodología de análisis. Configuración, diseño, gestión y evaluación de sistemas informáticos. Entornos de sistemas informáticos. Tecnologías avanzadas de sistemas de información, bases de datos y sistemas operativos. Proyectos de sistemas informáticos.	-	**15**	**15**	ATC, CCIA, EIO, ISA, ITEL, LSI, OE

ALG: Álgebra; AMAT: Análisis Matemático; ATC: Arquitectura y Tecnología de Computadores; CCIA: Ciencia de la Computación e Inteligencia Artificial; EFC: Economía Financiera y Contabilidad; EIO: Estadística e Investigación Operativa; ELEC: Electrónica; EM: Electromagnetismo; FISA: Física Aplicada; IELEC: Ingeniería Eléctrica; ISA: Ingeniería de Sistemas y Automática; ITEL: Ingeniería Telemática; LSI: Lenguajes y Sistemas Informáticos; MA: Matemática Aplicada; OE: Organización de Empresas; TELEC: Tecnología Electrónica

Tabla 2.5: Directrices de la titulación Ingeniero en Informática, RD 1459/1990.

Troncal y descripción	Créditos			Áreas de conocimiento
	T	P	Total	
INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN				
Estadística Estadística descriptiva. Probabilidades. Métodos estadísticos Aplicados.	-	-	**6**	CCIA, EIO, MA
Estructura de Datos y de la Información Tipos Abstractos de Datos. Estructura de Datos y Algoritmo de Manipulación. Estructura de Información: Ficheros, Bases de Datos.	-	-	**12**	CCIA, LSI
Estructura y Tecnología de Computadores Unidades Funcionales: Memoria, Procesador, Periferia, L. Máq. y Ensamblador, Esquema de Funcionamiento. Electrónica. Sistemas Digitales. Periféricos.	-	-	**9**	ATC, ELEC, ISA, TELEC
Fundamentos Matemáticos de la Informática Algebra Análisis Matemático. Matemática Discreta. Métodos numéricos.	-	-	**18**	ALG, AMAT, CCIA, MA
Ingeniería del Software de Gestión Diseño, propiedades y mantenimiento del software de gestión. Planificación y gestión de proyectos informáticos. Análisis de aplicaciones de gestión.	-	-	**12**	CCIA, LSI
Metodología y Tecnología de la Programación Diseño de Algoritmos. Análisis de Algoritmos. Lenguajes de programación. Diseño de Programas: Descomposición Modular y Documentación. Técnicas de Verificación y Pruebas de programas.	-	-	**15**	CCIA, LSI
Sistemas Operativos Organización, estructura y servicios de los sistemas operativos. Gestión y administración de memoria, procesos y entrada/salida. Sistemas de Ficheros.	-	-	**6**	ATC, CCIA, LSI
Técnicas de Organización y Gestión Empresarial El sistema económico y la empresa. Técnicas de administración y contables.	-	-	**12**	EFC, EO
INGENIERO TÉCNICO EN INFORMÁTICA DE SISTEMAS				
Estadística Estadística descriptiva. Probabilidades. Métodos estadísticos Aplicados.	-	-	**6**	CCIA, EIO, MA
Estructura de Datos y de la Información Tipos Abstractos de Datos. Estructura de Datos y Algoritmo de Manipulación. Estructura de Información: Ficheros, Bases de Datos.	-	-	**12**	CCIA, LSI
Estructura y Tecnología de Computadores Unidades Funcionales: Memoria, Procesador, Periferia, L. Máq. y Ensamblador, Esquema de Funcionamiento. Electrónica. Sistemas Digitales. Periféricos.	-	-	**15**	ATC, ELEC, ISA, TELEC
Fundamentos Físicos de la Informática Electromagnetismo. Estado sólido. Circuitos.	-	-	**6**	ELEC, MA, ISA, IELEC, TELEC
Fundamentos Matemáticos de la Informática Algebra Análisis Matemático. Matemática Discreta. Métodos numéricos.	-	-	**18**	ALG, AMAT, CCIA, MA
Metodología y Tecnología de la Programación Diseño de Algoritmos. Análisis de Algoritmos. Lenguajes de programación. Diseño de Programas: Descomposición Modular y Documentación. Técnicas de Verificación y Pruebas de programas.	-	-	**12**	CCIA, LSI
Redes Arquitectura de Redes. Comunicaciones.	-	-	**6**	ATC, CCIA, ISA, ITEL, LSI
Sistemas Operativos Organización, estructura y servicios de los sistemas operativos. Gestión y administración de memoria, procesos y entrada/salida. Sistemas de Ficheros.	-	-	**6**	ATC, CCIA, LSI
Teoría de Autómatas y Lenguajes Formales Máquinas secuenciales y Autómatas finitos. Máquinas de Turing. Funciones Recursivas. Gramáticas y Lenguajes Formales. Redes neuronales.	-	-	**9**	ALG, CCIA, ISA, LSI, MA

Tabla 2.6: Directrices de I.T.I. de Gestión e I.T.I. de Sistemas, RD 1460 y 1461/1990.

organización. Por su parte, las directrices para II e ITIS sugieren una formación orientada al desarrollo de aplicaciones de sistemas, esto es, aplicaciones que requieren un mayor conocimiento del hardware y de los pilares teóricos de la informática.

Centrándonos en la materia objeto de estudio, los algoritmos y las estructuras de datos reparten sus contenidos entre dos troncalidades: "Estructura de Datos y de la Información", y "Metodología y Tecnología de la Programación", constituyendo uno de los ejes básicos del primer ciclo de la II (ambas suman por encima del 30 % del total de créditos troncales). Esta situación es idéntica en las tres titulaciones –salvo por una diferencia en el número de créditos en ITIS–. Evidentemente, las directrices son lo suficientemente generales como para no entrar en detalle sobre las técnicas concretas que se deben estudiar. Todos los descriptores se refieren a grandes campos de la informática, de entre los cuales se debe seleccionar lo más adecuado para incluirlo en los programas de una o varias asignaturas. Reagrupando los descriptores de las dos troncalidades por áreas dentro de la informática, podemos distinguir las cuatro siguientes:

- **Estructuras de datos:** tipos abstractos de datos, estructura de datos y algoritmos de manipulación.
- **Algorítmica:** diseño de algoritmos y análisis de algoritmos.
- **Ingeniería del software:** lenguajes de programación, diseño de programas, descomposición modular y documentación, técnicas de verificación y pruebas de programas.
- **Bases de datos:** estructura de información, ficheros, bases de datos.

Las dos primeras áreas anteriores son las que determinan el ámbito específico de la materia "Ampliación de Algoritmos y Estructura de Datos". Pero no resulta posible, ni adecuado, eliminar la referencia a los aspectos contenidos en las otras dos áreas relacionadas. Por un lado, porque la materia en estudio tiene como base la programación, y esta debe seguir un proceso metódico propio de una ingeniería: análisis, diseño, verificación, documentación, etc. Por otro lado, porque se debe dar una buena base para el estudio de esas materias en cursos posteriores.

Debemos aclarar, no obstante, que la asignación de descriptores a asignaturas no es responsabilidad de un proyecto docente, ni de las directrices generales propias, sino del plan de estudios.

Inciso sobre la vigencia de las directrices

Cabe hacer un comentario sobre la vigencia actual de las directrices generales propias. Desde su aprobación hasta el momento presente han pasado casi catorce años, lo cual es mucho tiempo, y especialmente en una disciplina como la informática. En estos años se han producido numerosos avances que, evidentemente, las directrices generales propias no contemplan: la programación orientada a objetos, la aparición de los servicios basados en web, el desarrollo de la tecnología multimedia, etc. Esto nos lleva a tratar las directrices con la debida cautela, no restringiendo el ámbito de estudio para las nuevas tecnologías que se consideren de interés.

Dos argumentos podrían, no obstante, atenuar la referida obsolescencia de las directrices propias. Primero, que las directrices fueron establecidas por expertos, que en previsión del problema seleccionaron el núcleo básico y más estable de la disciplina, la base teórica que se encuentra por debajo de todo desarrollo tecnológico. Segundo, que la ambigüedad de los descriptores permite una interpretación suficientemente flexible, en la que caben aspectos que en principio no se tenían en mente; por ejemplo, la orientación a objetos podría verse reflejada en los descriptores: lenguajes de programación, diseño de programas y tipos abstractos de datos.

2.2.2. Las recomendaciones del informe Computing Curricula 2001

Desde sus primeras versiones, allá por el final de la década de 1960, las recomendaciones curriculares para los estudios de informática realizadas por la ACM (*Association for Computing Machinery*) y la IEEE-CS (*Computer Society* de IEEE) –primero de forma separada y a partir de 1991 uniendo sus esfuerzos en una labor conjunta– han constituido un verdadero referente internacional en el diseño de los planes de estudios. De hecho, la publicación de la última versión de estas recomendaciones, denominadas Computing Curricula 2001 (en adelante CC2001) [CC'2001], supuso uno de los alicientes para la reforma de los planes de estudios de la FIUM y una de las referencias básicas, como veremos en el apartado 2.2.3. En el documento final han estado involucradas directamente más de 150 personas, repartidas en diferentes grupos de trabajo; además de los numerosos miembros de la comunidad académica y de la industria, que realizaron sus aportaciones en tres borradores previos y varias reuniones a lo largo de todo el mundo.

Las recomendaciones son, por lo tanto, un interesantísimo documento de trabajo, cuyo nivel de detalle permite su utilización no sólo en el diseño de planes de estudios sino también en la planificación de cada una de las asignaturas. En este apartado analizaremos las características generales de las propuestas del CC2001. Volveremos a hacer uso de este documento en la determinación de los objetivos de la materia y en la selección de los contenidos de la asignatura.

Antecedentes y motivaciones

La elaboración de las recomendaciones CC2001 no partió desde cero sino que, como hemos comentado, se disponía de una larga tradición en el diseño de modelos curriculares. Desde el primer informe de ACM, que data de 1968, se encontraba ya la intención de homogeneizar a escala global los conocimientos mínimos que debía poseer todo titulado en informática. Se ofrecían programas detallados con las materias a cubrir, así como su distribución en cursos y extensas relaciones de referencias bibliográficas para cada materia de estudio. Pero este documento quedó pronto desfasado, realizándose una revisión importante en el año 1978, en la que se aprovechó la mayor experiencia y madurez alcanzada dentro de la disciplina. Este informe se convirtió en un indiscutible estándar *de facto*, un modelo que sería usado por la mayoría de universidades del mundo en cuanto a la organización de muchas asignaturas.

De forma independiente, un grupo de trabajo de IEEE-CS elaboró una primera propuesta curricular en 1977. Este informe proponía una visión más ingenieril de la informática, tratando de llenar el hueco existente entre la visión del informático como mero programador y la vertiente más tradicional de las ingenierías. En 1983 tiene lugar una revisión y actualización de este informe.

A finales de la década de 1980, las dos organizaciones deciden unir sus esfuerzos, intentando así evitar el conflicto que se encontraban las universidades en la implantación de ambas recomendaciones. De esta forma, en 1991 un grupo de trabajo formado por miembros de ambas asociaciones y dirigido por Allan B. Tucker, elabora una propuesta conjunta [CC'1991], también de reconocida influencia a lo largo de la década de los noventa. Esta propuesta tenía la virtud de equilibrar las tendencias de marcado carácter hardware (IEEE-CS) y software (ACM) de las recomendaciones previas por separado. Como novedad, el informe se estructuraba por unidades de conocimiento, frente a la organización por asignaturas de los informes previos. Esta fue una de las grandes críticas que recibió, ya que las universidades encontraban más cómoda la estructuración por asignaturas propuesta en la vieja propuesta de la ACM de 1978.

Inevitablemente, tras diez años de funcionamiento se hacía necesaria una revisión del CC'1991. La aparición de nuevas tecnologías, muchas de las cuales quedan obsoletas *"casi tan pronto como lo que tardan en aparecer"*, tiene profundos efectos en la enseñanza de la informática, tanto en los contenidos

como en los métodos de enseñanza. En consecuencia, los grupos de trabajo conjuntos de ACM e IEEE-CS inician la elaboración de las nuevas recomendaciones curriculares, que culminan en diciembre de 2001 con la publicación del informe final [CC'2001]. El documento clasifica los cambios producidos durante el periodo comprendido entre ambas recomendaciones en torno a los ejes tecnológico y cultural:

- **Cambios tecnológicos.** El incremento exponencial de la potencia de cálculo –siguiendo la ley de Moore, según la cual la densidad de los procesadores se duplica cada 18 meses–, ha permitido abordar problemas cuya resolución estaba fuera de alcance hace tan sólo unos pocos años. Y si estos cambios suponen una evolución, los avances en redes de ordenadores son calificados de revolución. Ambos factores han hecho necesario introducir o potenciar aspectos del currículum tales como: tecnologías web y de redes, particularmente las basadas en TCP/IP; gráficos y multimedia; sistemas embebidos; bases de datos relacionales; interoperatividad; programación orientada a objetos; uso de API sofisticadas; interacción hombre-máquina; seguridad y criptografía; y dominios de aplicación. Obviamente, la potenciación de estas áreas requiere la reducción de otras, ya que los contenidos no pueden aumentar de manera indefinida.

- **Cambios culturales.** La enseñanza también se ve influida por los cambios culturales y sociales que ha provocado la propia tecnología informática. Las nuevas tecnologías permiten la renovación de los métodos pedagógicos, como la incorporación de presentaciones, demostraciones o la posibilidad de la enseñanza a distancia. La mayor disponibilidad de ordenadores hace que los alumnos entren a las universidades habiendo usado ya ordenadores, lo cual puede aumentar las diferencias entre los conocimientos iniciales de los mismos. La influencia de la informática en la economía, el reconocimiento pleno de la informática como una disciplina académica propia y la expansión del ámbito de la disciplina, son otros factores que deben ser tenidos en cuenta.

En conclusión, estaba plenamente justificada una revisión de las recomendaciones curriculares, que evitara los problemas de informes anteriores. El nuevo marco, aunque sigue estructurándose en unidades de conocimiento al estilo del CC'1991, propone implementaciones concretas, definiendo asignaturas con sus programas de estudio incluidos. Antes de desarrollarlo en detalle, se marcan una serie de principios que deberían ser tenidos en cuenta en la aplicación del CC2001, entre los que destacamos los siguientes:

- La informática es multidisciplinar en sus bases, por lo que los estudiantes deben saber integrar los conocimientos de diferentes ámbitos. Además, *"todos los estudiantes de informática deben aprender a integrar teoría y práctica, reconocer la importancia de la abstracción y apreciar el valor de un buen diseño de ingeniería"*.

- La rápida evolución de la informática requiere un proceso continuo de revisión de los planes de estudios, no limitado al plazo de diez años que se marca para la próxima actualización del CC2001.

- El dinamismo de la disciplina también requiere que los alumnos aprendan a desenvolverse en este ámbito, de manera que *"la educación en informática debe tratar de preparar a los estudiantes para un aprendizaje que dure toda la vida, permitiéndoles ir más allá de la tecnología actual hacia los desafíos del futuro"*.

- Las recomendaciones identifican el núcleo básico de conocimientos, que es mantenido tan reducido como sea posible, a pesar del amplio crecimiento de la disciplina. Los planes de estudios deben incluir otros contenidos elegidos por las universidades.

- La elaboración del CC2001 busca la participación de miembros de todos los sectores, y sus resultados deberían tener un ámbito de aplicación internacional.
- Los programas deberían incluir prácticas profesionales, en las que se puedan desarrollar habilidades de comunicación oral y escrita, ética y valores, trabajo en equipo y autoformación. Se establece que *"un dominio de la disciplina incluye no sólo entender los contenidos de las materias básicas, sino también entender la aplicabilidad de los conceptos a problemas del mundo real"*.

El cuerpo de conocimiento de la informática

Siguiendo la estructuración de recomendaciones previas, el CC2001 distingue entre los conocimientos que deben ser enseñados y la organización de los mismos en asignaturas. El cuerpo de conocimiento de la disciplina es descrito de forma jerárquica en tres niveles: áreas, unidades y tópicos. Las **áreas** representan los grandes campos dentro de la disciplina de la informática, como las redes, los sistemas operativos, o los algoritmos y complejidad. Estas áreas son descompuestas en **unidades**, que representan módulos temáticos dentro de cada área. A las áreas se les asocia un código identificativo de dos letras, mientras que las unidades tienen el código del área correspondiente seguido de un número. En el nivel más bajo, cada unidad es dividida en un conjunto de **tópicos**[23]. Estos tópicos son de una gran utilidad, ya que sirven para concretar lo que se entiende incluido o no dentro de cada unidad[24], indicando métodos, técnicas y algoritmos específicos.

El informe distingue las siguientes catorce áreas. Destacamos en negrita las que se relacionan de manera más clara y directa con la materia "Ampliación de Algoritmos y Estructura de Datos".

- Estructuras Discretas (DS)
- **Fundamentos de Programación (PF)**
- **Algoritmos y Complejidad (AL)**
- Arquitectura y Organización (AR)
- Sistemas Operativos (SO)
- Computación Orientada a Redes (NC)
- Lenguajes de Programación (PL)
- Interacción Hombre-Máquina (HU)
- Gráficos y Computación Visual (GR)
- Sistemas Inteligentes (IS)
- Gestión de la Información (IM)
- Aspectos Sociales y Profesionales (SP)
- Ingeniería del Software (SE)
- Ciencia Computacional y Métodos Numéricos (CS)

[23] Usamos la traducción literal del inglés por coherencia con la organización del documento original.

[24] Pues en otro caso nos encontraríamos ante una ambigüedad e imprecisión similar a la de las directrices generales propias.

Dentro de las 132 unidades en las que se dividen estas áreas, se identifica un conjunto de 64 como el **núcleo** básico y mínimo de la disciplina, compuesto por aquellas unidades para las cuales existe un amplio consenso en cuanto a su carácter esencial para cualquier titulado de informática. El resto de unidades son electivas. Por sí solas, las unidades del núcleo no pueden constituir un plan de estudios, sino que deben ser completadas con unidades electivas a criterio de cada universidad. Por otro lado, para orientar sobre la carga de trabajo se da una estimación del número de horas mínimo necesario para cada una de las unidades del núcleo. Debemos tomar estas medidas de forma orientativa, ya que se supone que por cada hora de clase el alumno debe hacer una preparación previa de tres horas. Hacer esta suposición en nuestro sistema educativo actual es altamente irrealista.

Las asignaturas se forman agrupando unidades, del núcleo y/o electivas, posiblemente de áreas muy distintas. Para satisfacer las recomendaciones del CC2001 entre todas las asignaturas se debería cubrir la carga asignada al núcleo de conocimientos.

Refiriéndonos, en concreto, a las áreas relacionadas más estrechamente con AAED (PF y AL) se distinguen las siguientes unidades, con las horas especificadas en las que forman parte del núcleo.

- **PF. Fundamentos de Programación (núcleo de 38 horas)**
  - PF1. Construcciones fundamentales de programación (9)
  - PF2. Algoritmos y resolución de problemas (6)
  - PF3. Estructuras de datos fundamentales (14)
  - PF4. Recursividad (5)
  - PF5. Programación dirigida por eventos (4)
- **AL. Algoritmos y Complejidad (núcleo de 31 horas)**
  - AL1. Análisis de algoritmos básico (4)
  - AL2. Estrategias algorítmicas (6)
  - AL3. Algoritmos fundamentales en computación (12)
  - AL4. Algoritmos distribuidos (3)
  - AL5. Computabilidad básica (6)
  - AL6. Las clases de complejidad P y NP
  - AL7. Teoría de autómatas
  - AL8. Análisis de algoritmos avanzado
  - AL9. Algoritmos criptográficos
  - AL10. Algoritmos geométricos
  - AL11. Algoritmos paralelos

También resulta muy interesante y reveladora la justificación y descripción general que se hace de cada una de estas dos áreas, en el apéndice donde se detalla el cuerpo de conocimientos. En referencia al área de Fundamentos de Programación (PF) se expone lo siguiente:

> *La fluidez en un lenguaje de programación es un prerrequisito para el estudio de la mayor parte de la ciencia informática. En el informe CC'1991, el conocimiento de un lenguaje de programación –aunque identificado como esencial– recibía poco énfasis en el currículum. El área "Introducción a un Lenguaje de Programación" representa sólo 12 horas*

de clase y es identificado como opcional, bajo la suposición optimista de que "un creciente número de estudiantes ... consiguen esa experiencia en la enseñanza secundaria". Creemos que los programas de las titulaciones de informática deben enseñar a los estudiantes cómo usar al menos un lenguaje de programación correctamente; es más, recomendamos que los programas de informática deberían enseñar a los estudiantes a ser competentes en lenguajes que hagan uso de ***al menos dos paradigmas de programación****. Alcanzar esta meta requiere considerablemente más de 12 horas.*

Este área de conocimiento consiste en aquellas ***habilidades y conceptos*** *que son esenciales en la práctica* ***de la programación independientemente del paradigma*** *subyacente. Como resultado, este área incluye unidades sobre conceptos fundamentales de programación, estructuras de datos básicas, y procesos algorítmicos. Estas unidades, sin embargo, no cubren completamente el rango de conocimientos de programación que un graduado en informática debe poseer. Muchas de las otras áreas –en especial Lenguajes de Programación (PL) e Ingeniería del Software (SE)– también contienen unidades relacionadas con la programación que son parte del núcleo de conocimientos. En muchos casos, estas unidades podrían igualmente haber sido asignadas a Fundamentos de Programación o al área más avanzada correspondiente.*

La elección del lenguaje de programación es un aspecto esencial en una asignatura como AAED. Pero la docencia no puede centrarse en la enseñanza de ese lenguaje concreto, sino que existen una serie de habilidades propias de la programación, que son las que deberían ser la base de la asignatura. Tal y como advierte el informe, *"Los cursos de programación a menudo se centran en la sintaxis y las características particulares del lenguaje [...] en lugar de en las habilidades de programación subyacentes"*. Es más, como se ha destacado antes, estas habilidades son también independientes del paradigma de programación; de esta manera los alumnos no deben identificar los algoritmos y las estructuras de datos como algo exclusivo de un paradigma imperativo. Todo esto nos conduce a la posibilidad de utilizar en clase un pseudocódigo, haciendo ver a los alumnos que los algoritmos y las técnicas estudiadas pueden ser aplicados en cualquier contexto de programación. No obstante, la elección de un paradigma y un lenguaje concretos será inevitable en la planificación de las prácticas. Abordaremos estas cuestiones en el Capítulo 4 (Selección y organización de contenidos).

En lo relativo al área de Algoritmos y Complejidad (AL), el informe CC2001 menciona lo siguiente:

Los algoritmos son fundamentales en la ciencia informática y en la ingeniería del software. El rendimiento real de cualquier sistema software depende sólo de dos cosas: (1) los algoritmos elegidos y (2) la adecuación y eficiencia de las distintas capas de implementación. Un ***buen diseño de los algoritmos*** *es, por lo tanto, crucial en el rendimiento de todos los sistemas informáticos. Es más, el estudio de los algoritmos ofrece un mayor entendimiento en la naturaleza intrínseca de los problemas así como posibles técnicas de solución independientes del lenguaje de programación, el paradigma de programación, el hardware del ordenador, o cualquier otro aspecto de implementación.*

Una parte importante de la computación es la habilidad de seleccionar los algoritmos apropiados para los propósitos particulares y saber aplicarlos a los mismos, reconociendo la posibilidad de que pueda no existir ningún algoritmo adecuado. Esta capacidad requiere el entendimiento de la variedad de algoritmos que abordan un conjunto de problemas bien definidos, reconociendo sus ventajas e inconvenientes, y su adecuación en contextos particulares. La eficiencia es un tema recurrente a lo largo de todo este área.

Nuevamente se insiste en la separación entre los conceptos teóricos subyacentes y el lenguaje y características particulares de implementación. Lo fundamental es la capacidad de diseño de buenos algoritmos, y la habilidad de escoger, entre todas las técnicas estudiadas, aquellas que son más útiles para los problemas particulares que se plantean. Los alumnos deben comprender las limitaciones de cada técnica, y considerar en todo momento las cuestiones relativas a la eficiencia, tanto en tiempo como en uso de memoria.

Organización y estructuración del currículum

Obviamente, el cuerpo de conocimiento descrito en el punto anterior no constituye, por sí mismo, un plan de estudios sino que debe ser implementado a través de distintas asignaturas con una organización temporal coherente. Como ya mencionamos, las asignaturas se pueden entender como una combinación de una serie de unidades de una o varias áreas distintas. En consecuencia, para ayudar en el diseño de planes de estudios el CC2001 incluye también recomendaciones detalladas sobre asignaturas y estrategias de composición de las mismas para formar un currículum completo.

Las asignaturas son clasificadas en tres categorías, según su nivel y su situación en el plan de estudios: introductorias, intermedias y avanzadas. Las asignaturas **introductorias** corresponden al primer y segundo año, y parten de los conocimientos de entrada de los alumnos. Las asignaturas **intermedias** son normalmente de segundo y tercer año, y sientan las bases para un posterior estudio en el área. Las asignaturas denominadas **avanzadas** tienen lugar en los últimos años, y requieren una preparación previa en las anteriores asignaturas del plan. Las asignaturas introductorias suelen tener normalmente más unidades incluidas en el núcleo, aunque en general no existe una relación directa entre asignatura introductoria/avanzada y unidad núcleo/electiva.

Esta distinción entre niveles ofrece una mayor flexibilidad en cuanto a las estrategias de implementación aplicadas en cada bloque de asignaturas. En particular, el informe CC2001 describe seis posibles estrategias para los cursos introductorios, y cuatro acercamientos temáticos para los intermedios. En cuanto al nivel avanzado, se proponen casi un centenar de asignaturas repartidas entre las distintas áreas. La organización de las diferentes estrategias de implementación es mostrada en la figura 2.5.

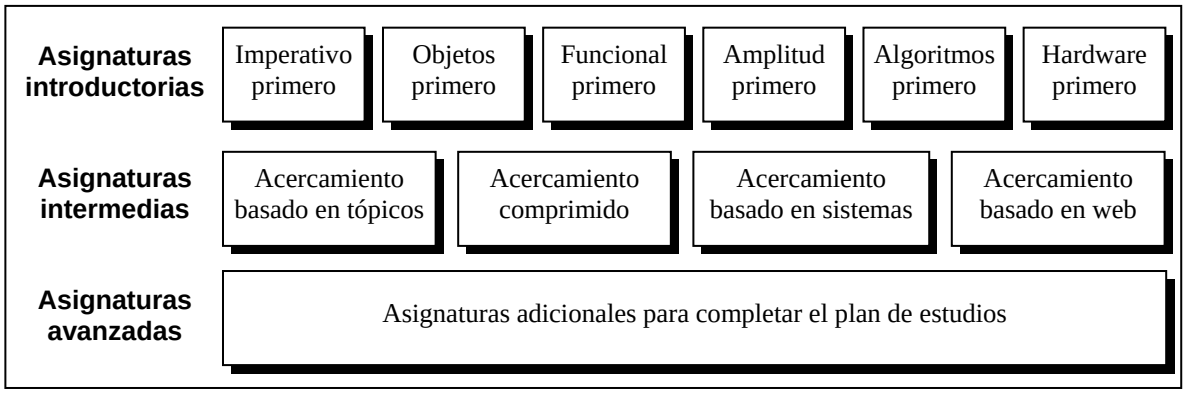


Figura 2.5: Estrategias de implementación propuestas en el informe CC2001 para cada nivel, [CC'2001].

La materia objeto de concurso se considera como un contenido de nivel introductorio, por lo que nos centraremos en las distintas estrategias propuestas para este nivel. De las seis estrategias, las tres primeras (imperativo, objetos o funcional primero) siguen una filosofía **"programación primero"**.

Este es el acercamiento más tradicional y el más aplicado en la mayoría de las universidades. Hunde sus raíces en la historia de muchas de las instituciones, pero tiene algunos inconvenientes que deben ser evitados: evitar la percepción errónea de que *"dominar la informática es igual a saber programar"*; no retrasar los aspectos teóricos y conceptuales; no enfocar las asignaturas en los aspectos de sintaxis y características del lenguaje, que hacen que los estudiantes se concentren en cosas irrelevantes; no eliminar completamente las etapas del proceso de desarrollo de programas, análisis, diseño y depuración; no desanimar o colocar en desventaja a los alumnos que llegan sin saber programar. A pesar de ello el modelo programación primero es claramente el dominante y se espera que lo siga siendo en el futuro. Ya hemos visto que son tres las estrategias de implementación que usan esta filosofía:

- **Imperativo primero.** Es sin duda el enfoque más tradicional de todos los propuestos. Está basado en comenzar el plan de estudios introduciendo los principales conceptos de los lenguajes imperativos: expresiones, estructuras de control, procedimientos y funciones. No obstante, se defiende que las ideas subyacentes al paradigma orientado a objetos no deben retrasarse mucho, incluyéndose en los primeros cursos con posterioridad al imperativo. Se plantea la posibilidad de usar desde el principio un lenguaje orientado a objetos, aunque la explicación se centre en los aspectos imperativos.

- **Objetos primero.** Este enfoque también se centra en la programación, pero haciendo un énfasis especial en los principios de la programación y el diseño orientados a objetos desde el principio. Se empieza describiendo las nociones básicas de objeto y herencia, experimentando con programas interactivos simples. Después el curso continúa introduciendo las estructuras de control más tradicionales, algoritmos, estructuras de datos fundamentales y conceptos de ingeniería del software. Este modelo tiene la ventaja de exponer antes a los alumnos a la programación OO, de indudable importancia tanto en la academia como en la industria. Pero hay que llevar cuidado con no abrumar a los alumnos con la excesiva complejidad de muchos lenguajes habitualmente usados, como C++ o Java.

- **Funcional primero.** Este acercamiento, menos común aunque también ha sido implantado con éxito, se caracteriza por introducir inicialmente un lenguaje funcional sencillo, como Scheme. La sintaxis de los lenguajes de este tipo es extremadamente simple, lo que permite que la enseñanza se enfoque en los conceptos teóricos fundamentales. Trabajar con este modelo supone un mayor esfuerzo de abstracción, de manera que algunas ideas importantes, como la recursividad, surgen desde el comienzo de forma bastante natural. Pero el uso de estos lenguajes, que raramente aparecen fuera del ámbito académico, puede desanimar a muchos estudiantes. Por eso la secuencia debe ser completada con una asignatura que trate el paradigma OO o el imperativo.

En términos generales, ya sea siguiendo uno u otro enfoque, hay una fuerte apuesta por la enseñanza de los fundamentos del paradigma OO desde los primeros cursos. La diferencia no está tanto en los contenidos de las asignaturas introductorias, como en su organización y disposición temporal.

Por otro lado, el CC2001 propone otros tres enfoques no basados en programación primero, y que han demostrado su viabilidad en, por lo menos, más de una institución de enseñanza superior. Los modelos expuestos son los siguientes:

- **Amplitud primero.** Con este modelo se intenta evitar la visión limitada de la disciplina que ofrecen los enfoques basados en programación primero. Los alumnos reciben desde el principio una amplia visión global de todas las áreas de la informática, de manera que pueden conocer antes dónde se encuentran sus intereses. La programación sigue siendo una parte fundamental, aunque no la única. En la práctica, las instituciones que usan este acercamiento deciden incluir una asignatura de este estilo, seguida por una secuencia centrada en alguno de los otros enfoques.

- **Algoritmos primero.** En este acercamiento, los conceptos básicos de la programación son introducidos usando pseudocódigo. De esta manera la docencia se puede centrar en los conceptos algorítmicos, evitando la preocupación por los detalles sintácticos del lenguaje. Los estudiantes razonan sobre los algoritmos a mano y mentalmente, pudiendo ampliar el abanico de estructuras de datos y de control presentadas. Al eliminarse la enseñanza prematura del lenguaje de programación, este enfoque deja más tiempo para profundizar en otros aspectos como el análisis de la complejidad y los principios de la computabilidad. Como contrapartida, se corre el riesgo de que los alumnos se sientan frustrados, al no tener la satisfacción de conseguir que *"el ordenador haga algo realmente"*. Por eso, es necesario coordinar la asignatura con las prácticas de laboratorio adecuadas.

- **Hardware primero.** Este acercamiento se basa en una construcción del conocimiento paso a paso, desde el nivel de máquina hasta los conceptos más abstractos de la informática. La filosofía consistiría en empezar por las puertas lógicas, después los registros y unidades aritméticas, y sólo después de establecer las bases del hardware se consideran los lenguajes de programación de alto nivel. Este acercamiento choca con la tendencia actual de la informática, en la que el software es cada vez más sofisticado. A los alumnos les puede resultar más difícil comprender los conceptos que van más allá de la implementación. Por estas razones, se considera que este acercamiento es más adecuado para ingenieros en computadores.

Estos tres enfoques son completados con asignaturas que introducen los conceptos de programación bajo un enfoque OO. En conjunto, los cursos introductorios requieren normalmente dos o tres asignaturas.

En definitiva, se definen una serie de conceptos que deberían ser cubiertos por las asignaturas de nivel introductorio, sea cual sea el enfoque que se aplique. Los conceptos están agrupados en tres categorías: pensamiento algorítmico, fundamentos de programación y entornos de computación. Los dos primeros son los que guardan una relación más directa con AAED, y contienen las siguientes descripciones:

- **Pensamiento algorítmico.**
  - **Computación algorítmica.** Algoritmos como modelos de los procesos computacionales; ejemplos de algoritmos importantes.
  - **Eficiencia algorítmica y utilización de los recursos.** Análisis simple de la complejidad algorítmica; consideraciones en la evaluación de compromisos; técnicas para la estimación y medida.
- **Fundamentos de programación.**
  - **Modelos de datos.** Estructuras estándar para representar datos; descripciones abstractas (descritas por un modelo) y concretas (descritas por una implementación).
  - **Estructuras de control.** Efecto de aplicar operaciones a objetos de un programa; qué hace una operación (descrita por una especificación); cómo lo hace la operación (descrita por una implementación).
  - **Flujo de ejecución.** Estructuras de control estándar: secuencia, selección, iteración; llamadas a funciones y paso de parámetros.
  - **Encapsulación.** Agrupación indivisible de entidades relacionadas; vista del cliente basada en la abstracción y en la ocultación de la información; vista del implementador basada en los detalles internos.

- **Relaciones entre componentes encapsulados.** El papel de las interfaces en el intercambio de información; responsabilidades de los componentes encapsulados hacia sus clientes; el valor de la herencia.
- **Prueba y depuración.** La importancia de la prueba; estrategias de depuración.

Los planes de estudios de la FIUM frente al informe CC2001

En este punto haremos una breve reflexión sobre la situación de los planes de estudios de las titulaciones de informática de la FIUM, en relación a las propuestas del CC2001. Más específicamente, estamos interesados en el enfoque aplicado en las asignaturas de nivel introductorio.

Debemos entender que la elección de una u otra estrategia no es –no puede ser– responsabilidad de la planificación de cada asignatura por separado, sino que es una decisión consensuada en el proceso de elaboración de los planes de estudios. De esta manera, no es nuestra intención hacer aquí una selección de la estrategia a usar, sino más bien analizar e identificar la filosofía implícita a los planes de nuestra facultad, en los que se enmarca la asignatura objeto de concurso.

En primer lugar, debemos tener en cuenta el contexto educativo diferenciado. Las recomendaciones del CC2001, próximas al sistema educativo estadounidense, suponen un reducido número de asignaturas simultáneas, y especialmente para los primeros años. Frente a esto, el sistema español permite hasta seis asignaturas en un mismo cuatrimestre, lo que evita tener que utilizar un único enfoque de manera secuencial. En particular, no hay una incompatibilidad entre una secuencia de programación y otra de hardware aplicadas simultáneamente. De esta forma, los planes de nuestra facultad incluyen en los primeros cursos asignaturas propias de programación y otras de organización y arquitectura de ordenadores; en consecuencia los enfoques hardware o amplitud primero carecen de sentido.

En lo relativo a la programación, los planes de estudios –tanto los nuevos como los antiguos, y en las titulaciones técnicas y en la superior– proponen claramente un **enfoque imperativo primero**. Como veremos más adelante, podemos encontrar en los planes: asignaturas de introducción a la programación en primero; asignaturas de algoritmos y estructuras de datos en segundo; y una asignatura de programación orientada a objetos –optativa en los planes antiguos y obligatoria en los nuevos– en tercer curso. Pero recordemos que incluso en el enfoque imperativo primero, el informe CC2001 sugiere una introducción temprana del paradigma OO, a partir del segundo semestre. Los planes de la FIUM suponen, en cierto sentido, una versión extrema que atrasa aún más la introducción de este tema, hasta el tercer año. No haremos aquí valoraciones personales sobre esta situación, sino que las dejaremos para el Capítulo 4 (Selección y organización de contenidos).

Afinando un poco más en el enfoque, en los proyectos docentes [Fernández'02] y [Montoya'01] se reconoce en nuestras titulaciones un **enfoque mixto**, que combina elementos de imperativo primero con otros de algoritmos primero. Esta afirmación se justifica, casi de forma exclusiva, en la planificación de las asignaturas de primer curso. Ambos proponen una enseñanza basada inicialmente en el uso de un pseudolenguaje, fácil de leer y de entender, y que evite a los alumnos centrarse en los aspectos de sintaxis. Su utilización previene algunas de las debilidades detectadas por el CC2001 en el modelo imperativo primero. Pero las propuestas que realizan ambos tampoco siguen plenamente el acercamiento algoritmos primero, ya que se continúa con las construcciones básicas de los lenguajes y se estudia un lenguaje de programación concreto al finalizar el primer cuatrimestre.

Hay otro aspecto destacado que merece la pena mencionar, relativo al estudio de la recursividad. Mientras que el enfoque imperativo primero del CC2001 retrasa el estudio de la recursividad hasta el tercer semestre –la última asignatura de la secuencia introductoria–, tradicionalmente este tema ha sido abordado en la FIUM desde las asignaturas de primer año. De esta manera se posibilita una

interesante interacción con las asignaturas de matemáticas, donde los alumnos estudian funciones recursivas y demostraciones por inducción.

Otras recomendaciones internacionales

Aparte del informe CC2001, existen otras recomendaciones internacionales disponibles, aunque indudablemente de menor repercusión. Por un lado podemos señalar la propuesta elaborada por el consorcio **Career Space**, también en el año 2001. Este consorcio está formado por algunas de las principales empresas europeas en tecnologías de la información y las comunicaciones, que en su informe [CareerSpace'01] establecen algunas guías generales sobre los conocimientos y habilidades que deberían mostrar los titulados en informática y la estructuración de los planes de estudios a gran escala. El nivel de detalle es claramente mucho más general que el del CC2001, sobre todo en cuanto a contenidos. Por su marcada orientación al estudio de habilidades y actitudes profesionales, revisamos este documento en el apartado 2.4.3 (Contexto profesional).

Otro grupo de recomendaciones de relativa importancia son las propuestas por la **UNESCO**, [UNESCO'94]. No entraremos en detalle sobre ellas, ya que están dirigidas fundamentalmente a países en vías de desarrollo; aunque sí analizaremos las propuestas de contenidos para la materia objeto de estudio, en el apartado 4.2.2. Estas recomendaciones pretenden aprovechar la experiencia de los países más avanzados en la elaboración de planes de estudios en informática –principalmente de las universidades americanas y europeas– y ofrecerla a los países menos favorecidos, pero tratando de adaptarlas a las peculiaridades de cada uno de ellos. El organismo encargado de su elaboración fue el IFIP (*International Federation for Information Processing*, Federación Internacional para el Procesamiento de la Información). Estas recomendaciones se remontan a 1984, siendo la última revisión del año 1994.

Por último, merece la pena mencionar aquí la importante influencia del currículum ofrecido por la Universidad de **Carnegie-Mellon** [CMU'02], una de las pioneras en la oferta de titulaciones universitarias en informática. Sus currículos son siempre un punto de referencia a la hora de elaborar planes de estudios para la disciplina. De hecho, las propuestas del CC2001 contienen numerosas referencias a las experiencias obtenidas por los sucesivos modelos curriculares impartidos en la misma.

2.2.3. Planes de estudio de las titulaciones de informática en la FIUM

Los planes de estudios determinan las materias que estudiarán los aprendices y su organización temporal. Por lo tanto, como bien apunta Zabalza [Zabalza'93], un plan de estudios debe conseguir un adecuado equilibrio entre la estabilidad y la actualización de sus contenidos. Por extensión, también los programas de las asignaturas que lo forman deben conseguir la justa proporción entre una base de conocimientos estable y el estudio de técnicas novedosas en la materia.

La evolución de los planes de estudios es un proceso en continuo funcionamiento, necesario en una disciplina tan activa como es la informática. En la tabla 2.2 se resume el proceso de implantación de los planes de estudios de Ingeniería Informática en la FIUM, y las distintas reformas que han tenido lugar hasta la fecha. El segundo ciclo de esta titulación se imparte en la UM desde el curso 1991/92. Posteriormente sería reformado en el curso 1995/96 para adaptarse a las directrices generales propias. El siguiente curso se implantó el primer ciclo de II, produciéndose la unificación de los dos ciclos en el año 2000. Es en este plan –y exclusivamente en este– donde aparece la asignatura "Ampliación de Algoritmos y Estructura de Datos". Este plan es frecuentemente denominado "Plan de estudios de 1996"[25], y será el primero que analicemos.

[25] Ver: `http://www.fi.um.es/estudios`. La información contenida en esta sección ha sido extraída fundamentalmente de esta página.

Pero, debemos recordar que AAED es una asignatura ya extinguida, impartida por última vez el curso 2002/03. Por esto hemos creído conveniente incluir el análisis del nuevo plan de estudios, de 2002 –implantado en segundo de II a partir de este mismo curso 2003/04–, en el que se desarrollará la docencia objeto de concurso. En este plan, la clara sucesora de "Ampliación de Algoritmos y Estructura de Datos" es la asignatura "Algoritmos y Estructuras de Datos".

Plan de estudios de Ingeniería Informática de 1996

El plan de estudios de la titulación "Ingeniero en Informática" impartido en la FIUM desde el año 1996 es mostrado en las tablas 2.7, 2.8, 2.9, 2.10, 2.11 y 2.12, que contienen los planes de primero, segundo, tercero, cuarto, quinto y las optativas, respectivamente. Se destaca en negrita la asignatura objeto de concurso, así como otras asignaturas con una relación más clara y directa con la primera.

En las tablas se ha utilizado la siguiente leyenda:

- **Tipo**. Tipo de asignatura: **T** – Troncal; **B** – Obligatoria; **O** – Optativa.
- **Créd.** Número total de créditos de la asignatura, teóricos y prácticos.
- **Dur.** Duración: **A** – Anual; **C** – Cuatrimestral.
- **Cuatr.** Cuatrimestre: **1** – Primero; **2** – Segundo; **-** – No aplicable.

PRIMER CURSO

Asignaturas Troncales y Obligatorias

Código	Asignatura	Tipo	Créd.	Dur.	Cuatr.
01T	Métodos numéricos	T	4,5	C	1
03T	Lógica computacional	B	5	C	2
88S	Análisis matemático	T	9	A	-
89S	Fundamentos físicos de la informática	T	6	C	1
90S	Fundamentos de computadores	T	9	C	1
91S	Álgebra y matemática discreta	T	9	C	1
92S	Programación	T	9	C	1
93S	Estructura de computadores	T	6	C	2
94S	Algoritmos y estructura de datos	T	6	C	2

Tabla 2.7: Plan de estudios de Ingeniería Informática, plan de 1996, curso Primero.

Además de las asignaturas indicadas en las tablas, los alumnos debían realizar a lo largo de toda la carrera un total de 37,5 créditos de libre configuración, lo que hace una media de 7,5 créditos por curso. En total, la carga de trabajo del plan de estudios es de 375 créditos.

Este plan introducía algunas características novedosas respecto a los anteriormente vigentes, algunas de las cuales resultaron exitosas y otras no tanto. A continuación destacamos algunas de las principales características del plan de 1996.

- En cuanto a la organización temporal, se eliminan casi todas las asignaturas anuales, partiéndolas en cuatrimestrales. Sólo permanece una anual, "Análisis Matemático". Es más, en el primer ciclo

SEGUNDO CURSO

Asignaturas Troncales y Obligatorias

Código	Asignatura	Tipo	Créd.	Dur.	Cuatr.
02T	Sistemas digitales	B	5	C	1
04T	Optimización	B	6	C	1
05T	Teoría de señales y sistemas	B	5	C	2
06T	Técnicas de inferencia	B	4	C	2
07T	Ampliación de sistemas operativos	B	5	C	2
08T	**Laboratorio de programación**	**B**	**6**	**C**	**2**
09T	Programación concurrente	B	6	C	2
95S	Estadística	T	7	C	1
96S	**Ampliación de algoritmos y estructura de datos**	**T**	**8**	**C**	**1**
97S	Sistemas operativos	T	7	C	1
98S	Teoría de autómatas y lenguajes formales	T	10	C	1

Tabla 2.8: Plan de estudios de Ingeniería Informática, plan de 1996, curso Segundo.

TERCER CURSO

Asignaturas Troncales y Obligatorias

Código	Asignatura	Tipo	Créd.	Dur.	Cuatr.
10T	Ampliación de estructura de computadores	B	8	C	1
11T	Fundamentos de ingeniería del software	B	9	C	1
12T	Resolución de problemas	B	7	C	2
13T	Traductores	B	7	C	1
14T	Redes y sistemas distribuidos	B	8	C	1
15T	Laboratorio de arquitectura, sistemas operativos y redes	B	6	C	2
16T	Diseño de programas	B	7	C	1
17T	Laboratorio de traductores y teoría autómatas	B	4,5	C	2
18T	Laboratorio de sistemas de información	B	4,5	C	2
99S	Bases de datos	T	9	C	1

Tabla 2.9: Plan de estudios de Ingeniería Informática, plan de 1996, curso Tercero.

la estructura no es exactamente cuatrimestral sino **semestral-trimestral**. La filosofía subyacente a este cambio era colocar en el semestre gran parte de la carga teórica de las asignaturas, para después centrar en los trimestres los contenidos prácticos. La justificación es que, de algún modo, las prácticas sirven para aplicar un conocimiento teórico aprendido previamente.

- La optatividad es reducida de forma drástica. De hecho, no hay **ninguna asignatura optativa** en el primer ciclo. En el segundo ciclo, los créditos optativos representan sólo un 24 % del total. Esto da poco margen a los alumnos para elegir distintos itinerarios según sus preferencias.

CUARTO CURSO

Asignaturas Troncales y Obligatorias

Código	Asignatura	Tipo	Créd.	Dur.	Cuatr.
V08	Arquitectura e ingeniería de computadores	T	9	C	2
V09	Ingeniería de software	T	6	C	1
V10	Arquitectura del software	T	6	C	2
V11	Inteligencia artificial	T	6	C	1
V12	Modelos de inteligencia artificial	T	4,5	C	1
V13	Procesadores de lenguaje	T	6	C	1
V14	Ampliación de procesadores de lenguaje	T	4,5	C	2
V15	Redes	T	9	C	2
V16	Programación y dirección de la producción	B	4,5	C	1

El alumno deberá cursar 12 créditos optativos.

Tabla 2.10: Plan de estudios de Ingeniería Informática, plan de 1996, curso Cuarto.

QUINTO CURSO

Asignaturas Troncales y Obligatorias

Código	Asignatura	Tipo	Créd.	Dur.	Cuatr.
V24	Gestión y planificación de proyectos informáticos	T	6	C	1
V25	Soportes tecnológicos de sistemas informáticos	T	9	C	1
V26	Proyecto informático	T	7,5	C	2
V27	Bases de datos avanzadas	B	6	C	1
V28	Diseño de arquitecturas de computadores	B	4,5	C	1
V29	Ingeniería del conocimiento	B	6	C	1
V30	Programación para la comunicación	B	6	C	2

El alumno deberá cursar 22,5 créditos optativos.

Tabla 2.11: Plan de estudios de Ingeniería Informática, plan de 1996, curso Quinto.

- Una de las innovaciones más exitosas es la creación de la asignatura de quinto **"Soportes Tecnológicos de Sistemas Informáticos"**, que nace como fruto de un esfuerzo de la Facultad por introducir a los alumnos en el trabajo en equipo y en el mundo de las empresas. En grupos de 6 a 8 personas, los alumnos debían investigar y realizar los trámites para constituirse en empresa, para después enfrentarse a problemas reales propuestos por empresas de desarrollo de software de la región, debiendo realizar el análisis del problema y el diseño de la solución. La colaboración de las empresas de la región es, por lo tanto, imprescindible en esta asignatura.

- El plan consolida la asignatura **"Proyecto Informático"**, que tiene carácter obligatorio, en la convicción de que un ingeniero en informática debe haber realizar un proyecto fin de carrera a lo largo de sus estudios. Esto no ocurre con las ingenierías técnicas, donde el proyecto es una asignatura optativa.

Asignaturas Optativas

Código	Asignatura	Tipo	Créd.	Dur.	Cuatr.
V18	Configuración y admon. de sistemas operativos	O	6	C	2
V20	Diseño de sistemas basados en microprocesadores	O	6	C	1
V21	Infografía	O	6	C	2
V22	Sistemas de ayuda a la decisión	O	6	C	2
V23	Control por ordenador	O	6	C	1
90W	Sistemas de percepción	O	6	C	2
V31	Ingeniería de sistemas informáticos	O	4,5	C	2
V32	Metodologías de desarrollo de software	O	4,5	C	1
V33	Monitorización y diagnostico inteligente	O	4,5	C	2
V34	Programación paralela	O	4,5	C	2
V35	Robótica	O	4,5	C	2
V37	Adquisición de conocimiento	O	4,5	C	2
V38	Ampliación de algoritmia	O	4,5	C	2
V41	Informática y legislación	O	4,5	C	2
V42	Modelado de computadores	O	4,5	C	2
V43	Sistemas multiprocesadores	O	4,5	C	1
V44	Sistemas tolerantes a fallos	O	4,5	C	2
V45	Técnicas de control y planificación inteligente	O	4,5	C	2
V46	Técnicas formales en ingeniería del software	O	4,5	C	1
91W	Visión por computador	O	4,5	C	2
92W	Aprendizaje computacional	O	4,5	C	1

Tabla 2.12: Plan de estudios de Ingeniería Informática, plan de 1996, asignaturas optativas.

La asignatura "Ampliación de Algoritmos y Estructura de Datos" (AAED) es un claro ejemplo de la *filosofía "semestre-cuatrimestre"*. Partiendo de una asignatura anual anterior[26], se descompone en una semestral, AAED, que acapara todo el contenido teórico de la materia, y una trimestral, "Laboratorio de Programación", a la que corresponde casi la totalidad de la carga práctica. De esta forma, de los 8 créditos de AAED, 6 son de teoría y sólo 2 de prácticas. En cambio, los 6 créditos de la trimestral son todos prácticos.

Por lo tanto, en el plan de 1996 se entiende que "Laboratorio de Programación" es el desarrollo práctico de AAED, de manera que los 2 créditos de AAED se dejan para la realización de prácticas de pizarra, clases de problemas, boletines de ejercicios, etc. Aun cuando se decidiera disponer de ellos como prácticas de programación, no sería suficiente para alcanzar los objetivos últimos de la materia y, además, podría resultar redundante con lo que se espera de la asignatura "Laboratorio de Programación". A estos inconvenientes se sumaban los problemas de coordinación entre los profesores involucrados en las distintas asignaturas, debido a la distancia temporal entre que se estudia la materia y se aplica la teoría, a la diferencia de criterios y al hecho de que ambas asignaturas eran impartidas por departamentos distintos.

En definitiva, el diseño de la asignatura AAED debía resolver el difícil problema de planificar exclusivamente mediante créditos teóricos la enseñanza de una materia eminentemente práctica. Posi-

[26] El precedente aquí es la asignatura "Algoritmos y Estructuras de Datos" de la antigua Diplomatura en Informática.

blemente no haya una solución satisfactoria para este problema, ya que la mejor opción pasa por organizar las asignaturas de forma muy diferente. Afortunadamente, esta cuestión fue tenida en cuenta al reformar los planes de estudios y, como veremos más adelante, se ha optado por volver a la estructuración anual.

Por otro lado, en referencia obligada a las Ingenierías Técnicas, tampoco los planes de 1996 habían mantenido la asignatura anual primitiva, sino que establecían dos cuatrimestrales. Pero, a diferencia de lo que ocurría con la II, la división es en este caso por contenidos, creándose la asignatura "Algorítmica" en el primer cuatrimestre de 2º curso, y "Estructura de Datos" en el segundo cuatrimestre. Y tampoco en este caso consideramos que la división sea razonable. Las dos guardan una relación muy estrecha, existiendo numerosos temas comunes: abtracciones, medición de la eficiencia –tanto en tiempo como en uso de memoria–, técnicas algorítmicas aplicadas sobre estructuras de datos no triviales, etc.; aparte de las ventajas de una asignatura anual frente a dos cuatrimestrales –entre las mayores, la posibilidad de hacer un seguimiento del aprendizaje de los alumnos a lo largo de todo el curso, frente al carácter eliminatorio de una cuatrimestral–. Finalmente, también en las Ingenierías Técnicas se ha vuelto al formato anual.

Plan de estudios de Ingeniería Informática de 2002

Entre los meses de noviembre y diciembre del año 2000, la comisión académica de la FIUM, a iniciativa del equipo decanal, empezó a plantearse la necesidad de reformar los planes de estudios de las tres titulaciones ofertadas en la Facultad. Tras un periodo de cinco años de aplicación, los planes de 1996 no sólo habían quedado obsoletos en algunos aspectos sino que la existencia de algunos problemas acuciantes aconsejaba la elaboración de unos nuevos planes. Ya hemos comentado algunos de estos inconvenientes en el punto anterior: escasa optatividad, desajustes producidos por la organización semestre-trimestre, poca diferenciación entre las titulaciones, etc.

En respuesta a esta situación, durante el primer trimestre del curso 2001/02 los departamentos implicados empiezan a trabajar en el considerable esfuerzo[27] de elaborar y consensuar los nuevos planes de estudios. Tras un largo proceso, la Junta de Facultad aprueba los nuevos planes en marzo de 2002 –siendo publicados en BOE el 30 de enero de 2003–, y se ponen en marcha en el mismo curso 2002/03. En la actualidad se están impartiendo todas las asignaturas del nuevo plan, excepto las de tercer curso de las ingenierías técnicas que comenzarán en el curso 2004/05. Tal y como fija la ley, las asignaturas del plan de 1996 quedan a extinguir, estando sin docencia pero guardándose el derecho a seis convocatorias de examen.

Podemos destacar los siguientes acuerdos, objetivos comunes a todos los departamentos implicados, como los que impulsaron el proceso de reforma de manera más importante:

- **Adecuación a las recomendaciones curriculares del CC2001**. Reconociendo el estado de obsolescencia de las directrices generales propias, de 1990, se adoptan estas recomendaciones internacionales como base de partida del proceso de reforma. La recomendaciones del CC2001 están respaldadas por importantes organizaciones, como IEEE y la ACM, así como por numerosas instituciones académicas en todo el mundo. Estas recomendaciones han sido ya analizadas en profundidad en el apartado 2.2.2.

- **Mayor diferenciación entre las tres titulaciones.** Se intenta conseguir un grado de distinción que no había quedado suficientemente marcado con los planes de 1996. Esto se hace a

[27] Debido a la necesidad de conciliar los intereses de los tres principales departamentos involucrados en la docencia de informática (DIS, DIIC y DITEC), tras la escisión de 1998; frente al único departamento existente cuando se elaboraron los planes de 1996.

través de la diferenciación tanto en asignaturas obligatorias como en optativas. La ITIG debe favorecer un perfil más orientado a ingeniería del software y gestión empresarial, la ITIS tiene un corte más centrado en el desarrollo de sistemas y la informática teórica, y la II ofrece una formación superior para trabajos de mayor cualificación, como la integración y administración de sistemas, la gestión de proyectos, el análisis y diseño de sistemas informáticos, la gestión de redes o la evaluación de arquitecturas, entre otros.

- **Potenciación y uniformidad en las optativas.** Frente a la escasa optatividad de los planes de 1996, se aumenta considerablemente el número de asignaturas optativas. Además, se iguala la carga lectiva de todas ellas a 6 créditos, la mitad de los cuales son teóricos y la otra mitad prácticos. Con este cambio se facilita la confección personalizada del currículum para los estudiantes, y también la ordenación docente para los departamentos.

- **Creación de las intensificaciones.** Además del aumento antes comentado, se crean grupos de optativas relacionadas, o intensificaciones, que permiten a los alumnos poder elegir entre distintas especializaciones. Las intensificaciones disponibles son: "Desarrollo de Aplicaciones Software y Negocio Electrónico" para la ITI de Gestión; "Redes y Sistemas", "Informática Gráfica y Multimedia", e "Integración de Sistemas y Aplicaciones" para ITI de Sistemas; e "Informática Industrial", "Sistemas Inteligentes y del Conocimiento", "Tecnología del Software", "Telemática", y "Arquitectura de Computadores y Sistemas Operativos" para la Ingeniería Superior. Para obtener alguna de estas intensificaciones, los alumnos deben completar al menos 24 créditos en las asignaturas del grupo correspondiente (lo cual equivale a 4 asignaturas).

- **Reordenación de la organización temporal.** Se pretende paliar los problemas encontrados con los planes antiguos, derivados de la supresión de las asignaturas anuales y el formato semestral-trimestral. La coexistencia de este formato (en el primer ciclo de II) con el cuatrimestral (en el segundo ciclo de II), daba lugar a periodos de exámenes que se extendían desde finales de enero hasta finales de marzo, con el consiguiente absentismo. Unido a la fragmentación de las asignaturas, eran algunos de los principales motivos de los elevados índices de fracaso encontrados. En los planes nuevos se vuelve a adoptar la temporización anual para las materias que dan pie a ello, en total, 11 en la II, y 5 en las Ingenierías Técnicas. Por otro lado, se pone fin al *experimento* semestre-trimestre, estableciendo cuatrimestres para las asignaturas no anuales.

- **Nuevas asignaturas obligatorias.** En parte como conclusión de las recomendaciones curriculares del CC2001, antes referidas, se acuerda la inclusión de nuevas asignaturas con carácter obligatorio, por su carácter esencial en la disciplina. En particular, se trata de "Programación Orientada a Objetos" y "Servicios Telemáticos Avanzados", que antes figuraban como optativas. Además, se aumenta la carga del proyecto fin de carrera, que pasa a tener 9 créditos en II. En las Ingenierías Técnicas el proyecto sigue siendo optativo, pero se reduce de 9 a 6 créditos.

El resultado de todos estos acuerdos son los nuevos planes de estudios de 2002. Las tablas 2.13, 2.14 y 2.15 muestran los planes nuevos del primer ciclo de Ingeniería Informática, mientras que los planes del segundo ciclo aparecen en las tablas 2.16, 2.17 y 2.18.

La leyenda utilizada en estas tablas es la siguiente:

- **Dur.** Duración de la asignatura: **An** – Anual; **1** – Primer cuatrimestre; **2** – Segundo cuatrimestre.

- **Tipo**. Tipo de asignatura: **T** – Troncal; **B** – Obligatoria.

- **Créd.** Número total de créditos de la asignatura, teóricos y prácticos.

PRIMER CURSO

Asignaturas Troncales y Obligatorias

Dur.	Tipo	Asignatura	Créd.	CT	CP
An	T	Álgebra y Matemática Discreta	12	9	3
An	T	Cálculo	12	9	3
An	T	Estructura y Tecnología de Computadores	15	9	6
An	T	Metodología y Tecnología de la Programación	15	9	6
1	T	Fundamentos Físicos de la Informática	6	4.5	1.5
2	B	Sistemas Lógicos Computacionales	4.5	3	1.5
2	B	Tecnología y Sistemas Electrónicos	4.5	3	1.5

Tabla 2.13: Plan de estudios de Ingeniería Informática, plan de 2002, curso Primero.

SEGUNDO CURSO

Asignaturas Troncales y Obligatorias

Dur.	Tipo	Asignatura	Créd.	CT	CP
An	**T**	**Algoritmos y Estructuras de Datos**	**12**	**6**	**6**
An	T	Sistemas Operativos	12	6	6
1	T	Autómatas y Lenguajes Formales	9	6	3
1	T	Estadística	7.5	4.5	3
1	B	Programación Concurrente	6	3	3
2	T	Computabilidad	4.5	3	1.5
2	B	Sistemas Digitales	6	3	3
2	B	Sistemas Inteligentes	6	3	3
2	B	Traductores	7.5	4.5	3

Tabla 2.14: Plan de estudios de Ingeniería Informática, plan de 2002, curso Segundo.

- **CT**. Número de créditos teóricos.
- **CP**. Número de créditos prácticos.

Igual que en los planes anteriores, los alumnos deben completar un total de 37,5 créditos de libre configuración. En conjunto, la carga total de trabajo del plan de estudios es de 375 créditos.

Se puede apreciar en las tablas el abundante número de asignaturas de carácter anual existentes en todos los cursos: "Álgebra y Matemática Discreta", "Cálculo", "Estructura y Tecnología de Computadores" y "Metodología y Tecnología de la Programación" en primero; "Algoritmos y Estructuras de Datos" y "Sistemas Operativos" en segundo; "Bases de Datos" y "Servicios Telemáticos Avanzados" en tercero; "Inteligencia Artificial e Ingeniería del Conocimiento" y "Redes" en cuarto; y "Proyecto Informático" en quinto. Por otro lado, se mantiene la satisfactoria experiencia iniciada con la asignatura "Soportes Tecnológicos de los Sistemas Informáticos", de quinto curso, denominada ahora "Ingeniería de Sistemas de Información". Recordemos que esta asignatura supone una implicación de las empresas informáticas murcianas, que se encargan de proponer proyectos para que sean realizados por grupos de alumnos.

En lo referente a la optatividad, hay un claro aumento en la oferta de asignaturas optativas y en

TERCER CURSO

Asignaturas Troncales y Obligatorias

Dur.	Tipo	Asignatura	Créd.	CT	CP
An	T	Bases de Datos	12	6	6
An	B	Servicios Telemáticos Avanzados	12	9	3
1	B	Fundamentos de Ingeniería del Software	9	6	3
1	B	Programación Orientada a Objetos	6	3	3
1	B	Redes y Sistemas Distribuidos	9	4.5	4.5
2	B	Administración de Sistemas Operativos	6	3	3
2	B	Ampliación de Estructura de Computadores	9	6	3

Asignatura Optativas	Créd.	CT	CP
Cálculo Numérico	6	3	3
Compresión de la Información	6	3	3
Comunicaciones Multimedia	6	3	3
Configuración y Ensamblaje de Equipos	6	3	3
Introducción a la Informática Gráfica	6	3	3
Investigación Operativa	6	3	3
Optimización	6	3	3
Procesamiento Audiovisual	6	3	3

El alumno debe realizar un total de 12 créditos optativos

Tabla 2.15: Plan de estudios de Ingeniería Informática, plan de 2002, curso Tercero.

CUARTO CURSO

Asignaturas Troncales y Obligatorias

Dur.	Tipo	Asignatura	Créd.	CT	CP
An	T	Inteligencia Artificial e Ingeniería del Conoc.	12	9	3
An	T	Redes	12	9	3
1	T	Ingeniería de Requisitos	6	3	3
1	T	Procesadores de Lenguaje	9	6	3
1	B	Señales y Sistemas	6	4.5	1.5
2	T	Análisis y Diseño de Software	6	3	3
2	T	Arquitectura e Ingeniería de Computadores	9	6	3
2	T	Gestión de Proyectos Informáticos	6	3	3

Tabla 2.16: Plan de estudios de Ingeniería Informática, plan de 2002, curso Cuarto.

el número de créditos que los alumnos deben realizar, pasando de 34,5 en los planes de 1996, a 54 en los planes de 2002. Además, como ya hemos mencionado, las optativas se agrupan en intensificaciones, siendo la de "Tecnología del Software" la más relacionada con el Área de Lenguajes y Sistemas Informáticos de DIS. Ocurre un curioso fenómeno de *vacío de optatividad* en cuarto curso, limitándose la oferta de optativas al segundo cuatrimestre de tercero y a quinto curso.

La materia objeto de estudio es una de las más afectadas por la reforma, aunque a nuestro

QUINTO CURSO

Asignaturas Troncales y Obligatorias

Dur.	Tipo	Asignatura	Créd.	CT	CP
An	T	Proyecto Informático	9	0	9
1	T	Ingeniería de Sistemas de Información	6	0	6

Tabla 2.17: Plan de estudios de Ingeniería Informática, plan de 2002, curso Quinto.

Asignaturas Optativas

Intensificación: Telemática	**Intensificación: Sistemas Inteligentes y de Conocimiento**
Arquitectura para Comunicaciones Móviles	Ingeniería del Conocimiento y de los Sistemas Inteligentes
Nuevos Servicios y Aplicaciones en Redes	Sistemas de ayuda a la decisión
Redes Móviles	Sistemas Multiagente y Sistemas Autónomos
Servicios Middleware: Seguridad, Criptografía y Agentes	Tratamiento Inteligente de la Información y Aplicaciones
Sistemas Integrados	Tecnologías del Conocimiento
	Tareas Inteligentes
Intensificación: Informática Industrial	**Intensificación: Tecnología del Software**
Automatización Industrial	Algoritmos y Programación Paralela
Control por Ordenador	Ampliación de Bases de Datos
Robótica	Arquitectura del Software
Sistemas de Eventos Discretos	Auditoría y Calidad del Software
Sistemas de Percepción	Desarrollo Basado en Componentes
Visión por Computador	Desarrollo de Aplicaciones Distribuidas
	Técnicas Formales en Ingeniería del Software
	Visualización y Realismo
Intensificación: Arquitectura de Computadores y Sistemas Operativos	**Bloque General**
Diseño de Arquitecturas de Alto Rendimiento	Filosofía, Tecnología Digital y Sociedad
Diseño y Estructura Interna de un Sistema Operativo	Técnicas de Resolución Óptima y Heurística por Ordenador
Diseño VLSI	Informática y Legislación
Gestión Avanzada de Sistemas Operativos	Dirección de Empresas y Sistemas de Información
Sistemas Distribuidos	
Sistemas Multiprocesadores	

Todas las optativas tienen una carga de 6 créditos, 3 teóricos y 3 prácticos.
El alumno debe realizar un total de 42 créditos optativos

Tabla 2.18: Plan de estudios de Ingeniería Informática, plan de 2002, asignaturas optativas de segundo ciclo, organizadas por intensificaciones.

parecer de forma bastante positiva. Es uno de los casos de integración de una asignatura semestral más una trimestral dentro de una anual. La nueva asignatura es “Algoritmos y Estructuras de Datos”, de segundo curso. Rectificando la idea de separar teoría y prácticas en dos asignaturas diferentes,

se vuelven a unir en la nueva asignatura, la cual cuenta con 6 créditos teóricos y 6 prácticos[28]. Esto permite solucionar algunos de los problemas ya expuestos, posibilitando un seguimiento del aprendizaje de los alumnos a lo largo del curso y mejorando la coordinación entre los profesores implicados en la materia. La extensión de la asignatura a lo largo de todo un curso facilita, además, un reparto más espaciado de los contenidos teóricos, a la vez que la aplicación simultánea de estos conocimientos en las prácticas.

Finalmente, comentaremos también, aunque más brevemente, los planes nuevos de las ingenierías técnicas. Las asignaturas troncales y obligatorias de estos planes son mostradas en la tabla 2.19.

I.T.I. de Gestión (Plan 2002)

Primer Curso

Dur.	Ti	Asignatura	C	CT	CP
An	T	Cálculo	12	9	3
An	T	Estructura y Tecnología de Computadores	15	9	6
An	T	Metodología y Tecnología de la Programación	15	9	6
1	T	Matemáticas para la Computación	9	6	3
1	T	Introducción a la Contabilidad	6	3	3
2	T	Administración de Empresas	9	6	3
2	T	Álgebra	6	4.5	1.5

Segundo Curso

Dur.	Ti	Asignatura	C	CT	CP
An	**T**	**Algoritmos y Estructuras de Datos**	**12**	**6**	**6**
An	T	Sistemas Operativos	12	6	6
1	T	Estadística	9	6	3
1	B	Programación Concurrente	6	3	3
1	B	Teoría de Autómatas y Lenguajes Formales	6	4.5	1.5
2	T	Fundamentos de Bases de Datos	6	3	3

Tercer Curso

Dur.	Ti	Asignatura	C	CT	CP
1	T	Fund. de Ingeniería del Soft.	6	3	3
1	B	Prog. Orientada a Objetos	6	3	3
1	B	Redes de Computadores	7.5	4.5	3
1	B	Tecnologías Avanzadas de Gestión de la Información	6	3	3
2	T	Desarrollo de Aplicaciones de Gestión	6	3	3
2	B	Diseño de Bases de Datos	6	3	3
2	B	Tecnologías de Servicios Telemáticos	6	3	3

I.T.I. de Sistemas (Plan 2002)

Primer Curso

Dur.	Ti	Asignatura	C	CT	CP
An	T	Cálculo	12	9	3
An	T	Estructura y Tecnología de Computadores	15	9	6
An	T	Metodología y Tecnología de la Programación	15	9	6
1	T	Matemáticas para la Computación	9	6	3
1	T	Fundamentos Físicos de la Informática	6	4.5	1.5
2	T	Álgebra	6	4.5	1.5
2	B	Componentes y Circuitos Elec.	4.5	3	1.5

Segundo Curso

Dur.	Ti	Asignatura	C	CT	CP
An	**T**	**Algoritmos y Estructuras de Datos**	**12**	**6**	**6**
An	T	Sistemas Operativos	12	6	6
1	T	Estadística	7.5	4.5	3
1	B	Programación Concurrente	6	3	3
1	T	Teoría de Autómatas y Lenguajes Formales	6	4.5	1.5
2	T	Computabilidad	4.5	3	1.5
2	B	Sistemas Digitales	6	3	3

Tercer Curso

Dur.	Ti	Asignatura	C	CT	CP
1	T	Fundamentos Bases de Datos	6	3	3
1	B	Progr. Orientada a Objetos	6	3	3
1	T	Redes	7.5	4.5	3
1	B	Sistemas Informáticos de Control	4.5	3	1.5
2	B	Administración de Sistemas Operativos y Periféricos	7.5	5.4	3
2	B	Ampliación de Estructura de Computadores	9	6	3
2	B	Sistemas Embebidos	4.5	3	1.5

Tabla 2.19: Planes de estudio de Ingeniería Técnica en Informática de Gestión (izquierda) e Ingeniería Técnica en Informática de Sistemas (derecha). Planes de 2002. No se muestran las asignaturas optativas.

Puede verse que hay una gran similitud entre las asignaturas de ambos planes, y de estas con

[28] Pero como resultado se han *perdido* 2 créditos en el proceso: la suma de "Ampliación de Algoritmos y Estructura de Datos" (8 créditos) más "Laboratorio de Programación" (6 créditos) es de 14 créditos, frente a los 12 créditos de la asignatura del plan nuevo.

las de la Ingeniería Superior. El parecido es aún mayor en los dos primeros cursos, reconociendo que la base de la formación de los titulados en una u otra carrera debe ser prácticamente la misma. La diferenciación entre ambas titulaciones la podemos encontrar, por un lado, en el tercer curso y, por otro, en la oferta de optativas.

Pero la homogeneidad entre los tres planes no sólo se refiere a las materias impartidas, sino también a su organización en las asignaturas. De esta forma, "Algoritmos y Estructuras de Datos" aparece en todos los planes como una asignatura anual de segundo curso y con los mismos créditos en las tres –cuando antes en las ingenierías técnicas aparecía separada en "Algorítmica" y "Estructuras de Datos"–. Sin duda alguna, este hecho facilita la coordinación entre los profesores involucrados en las tres asignaturas; lo que puede resultar en interesantes mejoras en la calidad de la docencia, gracias a la posibilidad de compartir experiencias, preparar mejor material docente, propuesta de actividades conjuntas, etc.

2.2.4. Asignaturas relacionadas con la materia

En el anterior apartado hemos analizado los planes de estudios de 1996 y de 2002 de las tres titulaciones de informática impartidas en la FIUM. En este apartado nos vamos a centrar en el plan de 2002 de Ingeniería en Informática, donde se encuentra la asignatura "Algoritmos y Estructuras de Datos" –la que toma el relevo de "Ampliación de Algoritmos y Estructura de Datos", a la que se refiere la plaza objeto de concurso–. Dentro del contexto curricular específico de AED como parte de su plan de estudios, podemos diferenciar, a su vez, dos ámbitos: uno más interno, compuesto por las asignaturas que presentan una relación más clara y directa con la asignatura objeto de estudio, en función de sus objetivos y contenidos; y otro más exterior, donde situamos otras asignaturas cuyos contenidos se aplican en algún tema de AED, o que usan conceptos estudiados en AED. La figura 2.6 muestra gráficamente la situación y las principales relaciones de AED dentro de su plan de estudios.

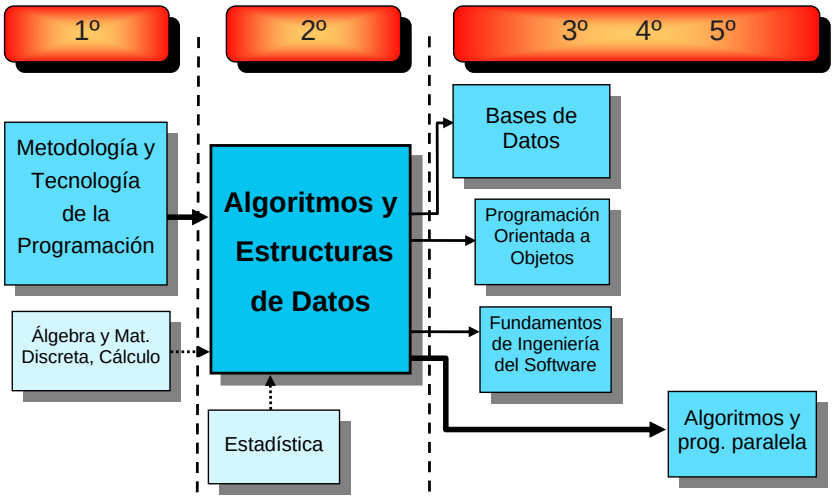


Figura 2.6: Relaciones de AED con otras asignaturas de su plan de estudios. Plan de II de 2002.

Asignaturas del ámbito interno de AED

Por las razones que detallamos más adelante, identificamos dentro del ámbito interno de AED las siguientes asignaturas: "Metodología y Tecnología de la Programación" anual de primer curso; "Bases

de Datos" anual de tercero; "Programación Orientada a Objetos" y "Fundamentos de Ingeniería del Software" cuatrimestrales de tercero; y "Algoritmos y Programación Paralela" optativa de quinto. La descripción de estas asignaturas –de acuerdo con el BOE del 30 de enero de 2003, donde se hace público el plan de estudios del título de Ingeniero en Informática– es la siguiente:

- Metodología y Tecnología de la Programación (MTP)
Tipo: Troncal. **Curso:** 1º. **Duración:** Anual. **Créditos:** 15 (9 teóricos y 6 prácticos). **Descripción:** Diseño de algoritmos. Análisis de algoritmos. Lenguajes de programación. Diseño de programas: Descomposición modular y documentación. Técnicas de verificación y pruebas de programas.

- Algoritmos y Estructuras de Datos (AED)
Tipo: Troncal. **Curso:** 2º. **Duración:** Anual. **Créditos:** 12 (6 teóricos y 6 prácticos). **Descripción:** Tipos abstractos de datos. Estructura de datos y algoritmos de manipulación. Ampliación de análisis y diseño de algoritmos.

- Bases de Datos (BBDD)
Tipo: Troncal. **Curso:** 3º. **Duración:** Anual. **Créditos:** 12 (6 teóricos y 6 prácticos). **Descripción:** Estructura de la información: ficheros, bases de datos. Fases del diseño de Bases de Datos. Métodos y técnicas de análisis y diseño de bases de datos. Sistemas de bases de datos.

- Programación Orientada a Objetos (POO)
Tipo: Obligatoria. **Curso:** 3º. **Duración:** Cuatrimestral. **Créditos:** 6 (3 teóricos y 3 prácticos). **Descripción:** Bases de la programación modular. Lenguajes de programación orientados a objetos. Reusabilidad.

- Fundamentos de Ingeniería del Software (FIS)
Tipo: Obligatoria. **Curso:** 3º. **Duración:** Cuatrimestral. **Créditos:** 9 (6 teóricos y 3 prácticos). **Descripción:** Introducción a los sistemas de información. Modelos de procesos. Métodos de desarrollo de software. Modelado de procesos y de datos. Ingeniería del software asistida por ordenador.

- Algoritmos y Programación Paralela (APP)
Tipo: Optativa. **Curso:** 5º. **Duración:** Cuatrimestral. **Créditos:** 6 (3 teóricos y 3 prácticos). **Descripción:** Técnicas avanzadas de diseño de algoritmos. Complejidad de problemas. Análisis y Diseño de algoritmos paralelos.

En el anexo A se incluyen los programas de estas asignaturas durante el curso 2003/04, que se pueden encontrar en la página web de la Facultad (`http://www.fi.um.es/estudios/programas/ii.html`).

A gran escala, podemos considerar la asignatura AED como una profundización en las habilidades de programación de los estudiantes, haciendo hincapié en la visión ingenieril de la programación y, en general, en las buenas prácticas de programación que permiten la construcción de programas eficientes y de calidad. En este sentido, los alumnos de segundo curso verán AED como la continuación natural de la asignatura anual de primero MTP. De hecho, en su planteamiento actual, AED retoma muchos temas abordados superficialmente en MTP, desarrollándolos de manera más dilatada y rigurosa. Entre estos aspectos comunes a ambas asignaturas podemos señalar: las abstracciones y especificaciones, el análisis de eficiencia, los tipos y estructuras de datos, y los esquemas algorítmicos. Todos ellos son introducidos en primero y ampliados después en segundo; y, no casualmente, constituyen el núcleo básico de AED.

Por otro lado, el componente de "estructuras de datos" de AED la relaciona de forma muy estrecha con la asignatura de tercero "Bases de Datos". En esta asignatura los alumnos estudian problemas relacionados con la organización y almacenamiento de información en bases de datos, es decir, en medios persistentes. Las estructuras de datos estudiadas en MTP y en AED, normalmente, están más orientadas a la organización de información en memoria RAM, con los matices que ello implica. No obstante, se pueden encontrar numerosos aspectos comunes, fundamentalmente en los temas de BBDD relativos a estructuras de almacenamiento, métodos de acceso y diseño físico de bases de datos. De acuerdo con la exposición del proyecto docente [Ortín'02], *"la asignatura AED introduce aspectos de estructuras de datos (árboles, dispersión) que suponen una base para BBDD"*.

Los aspectos de calidad del software mencionados previamente –conseguida a través de un proceso metódico–, son un paso previo para las asignaturas de tercero "Programación Orientada a Objetos" y "Fundamentos de Ingeniería del Software". Tales objetivos no deben ser exclusivos de las asignaturas del grupo de ingeniería del software, sino que desde los primeros años de su formación, los estudiantes deben concienciarse de la importancia de las abstracciones y de la utilización de un proceso ingenieril en la construcción de programas. Estos aspectos están subyacentes en AED, aunque son desarrollados en profundidad en POO y en FIS.

Finalmente, la asignatura "Algoritmos y Programación Paralela" –que surge tras la fusión de "Programación Paralela" y "Ampliación de Algoritmia", de los planes de 1996– pretende ser, en parte, una continuación de las técnicas de diseño de algoritmos estudiadas en segundo. Se trata de una asignatura optativa de quinto curso. Entre las técnicas que presenta su programa encontramos: algoritmos probabilistas, genéticos y problemas de cálculo matricial.

Asignaturas del ámbito externo de AED

Suele ocurrir con cierta frecuencia que, en el desarrollo de una clase o de unas prácticas, demos por supuestos conceptos y técnicas estudiados en otras asignaturas no directamente relacionadas con la nuestra. O bien que no los demos por supuestos, y estemos explicando algo que los alumnos ya han visto con anterioridad. Por este motivo, consideramos interesante hacer un repaso de otras asignaturas distantes, en principio, de AED, pero con una relación que resulta adecuado hacer explícita. Podemos señalar las siguientes:

- **Álgebra y Matemática Discreta** (curso 1º, anual). Además de los necesarios conceptos sobre funciones y lógica booleana, esta asignatura incluye un amplio tema de grafos en el que se estudia terminología, problemas, teoremas y algoritmos sobre grafos. Es importante tener esto en cuenta, ya que hasta hace unos años la asignatura de segundo correspondiente a AED era el primer lugar donde los alumnos estudiaban grafos, situación que ya no se da.

- **Cálculo** (curso 1º, anual). La parte de análisis de algoritmos de AED requiere una buena base matemática, que es proporcionada en esta asignatura. Destacamos de su temario, como necesarios para AED, las sucesiones, límites, demostraciones por inducción, funciones reales de variable real, derivadas e integrales.

- **Sistemas Operativos** (curso 2º, anual). Señalamos de esta asignatura el interés que ha surgido en este y otros años de coordinar el lenguaje de programación usado en las prácticas, así como la enseñanza del mismo. Durante el presente curso, las prácticas de ambas asignaturas usan el lenguaje C sobre Linux; el estudio de C se realiza en AED, y Linux en "Sistemas Operativos".

- **Estadística** (curso 2º, 1er cuatrimestre). La estadística es un elemento fundamental del análisis experimental de algoritmos, donde se intentan obtener conclusiones a partir de muestreos de

tiempos de ejecución de programas para diferentes condiciones de ejecución. Las técnicas que pueden ser interesantes en AED son el análisis de regresión y el análisis de la varianza.

- **Programación Concurrente** (curso 2º, 2º cuatrimestre). En este caso, la relación es en sentido contrario: puede ser interesante en esta asignatura la implementación concurrente de algunos algoritmos estudiados en AED.

- **Computabilidad** (curso 2º, 2º cuatrimestre). Tradicionalmente, se ha hecho referencia en AED a cuestiones relativas a la complejidad computacional y las clases de problemas P y NP. La presencia de esta asignatura troncal aconseja no entrar en detalles, en AED, más de lo estrictamente necesario.

- **Sistemas Inteligentes e Inteligencia Artificial** (cursos 2º y 3º). En las asignaturas de este tipo se incidirá en el componente más algorítmico de AED, aunque con las peculiaridades propias del área de inteligencia artificial. En estas asignaturas se estudian técnicas de resolución de problemas, uso de heurísticas, búsquedas en árboles y grafos, y árboles de juegos.

- **Análisis y Diseño de Software** (curso 4º, 2º cuatrimestre). Esta asignatura puede requerir conocimientos adquiridos en AED, aunque a través de una relación más débil. En cierto sentido, los objetivos de la asignatura son parecidos a los de AED, pero usando una filosofía orientada a objetos, sustituyendo los esquemas algorítmicos por patrones de diseño.

- **Proyecto Informático** (curso 5º, anual). Casi todas las asignaturas pueden argumentar una relación con esta, en cuanto a la posibilidad de realizar un proyecto fin de carrera sobre un tema tratado en la misma. No obstante, por su carácter básico y su ubicación en 2º curso, resulta poco atrayente para los alumnos realizar un proyecto sobre aspectos tratados en AED.

- **Técnicas Formales en Ingeniería del Software** (curso 5º, 1er cuatrimestre, optativa). La asignatura AED ha incluido, tradicionalmente, en su primer tema, el estudio de un lenguaje de especificación formal algebraico. El lenguaje utilizado en AED es parecido a una de las técnicas formales estudiadas en esta asignatura. Su estudio puede ser un aliciente, en el mejor caso, para que los alumnos elijan esta optativa.

- **Técnicas de Resolución Óptima y Heurística** (curso 5º, 2º cuatrimestre, optativa). Esta asignatura da la posibilidad, a los alumnos interesados, de ampliar los conocimientos adquiridos en AED sobre algoritmos óptimos y heurísticos. No obstante, su carácter es netamente matemático.

2.3. El contexto personal

Sin duda alguna, los alumnos son la materia prima de las Universidades y la razón de su existencia. Analizar las características de nuestros alumnos, su procedencia, objetivos, intereses, motivaciones, aspiraciones personales y procesos de aprendizaje, son elementos imprescindibles en todo proyecto docente. En esta sección haremos un estudio apriorístico de los alumnos universitarios en general, y de los alumnos de la materia objeto de concurso en particular. Las conclusiones extraídas de este análisis deberán verse reflejadas de alguna manera, y fundamentalmente, en las decisiones pedagógicas y metodológicas del plan docente. La enseñanza de la materia debe adecuarse en sus formas a las circunstancias personales de los que serán sus receptores.

Pero conviene no olvidar que el estudio verdaderamente relevante es el que se debe hacer en el trascurso de la propia acción docente, a través del seguimiento en clase de los alumnos, la realimentación obtenida en las tutorías y la valoración de los resultados de las evaluaciones parciales. La capacidad de adaptación *dinámica* a estos factores constituye un requisito quizás más necesario que el establecimiento de un plan inicial y fijo de acción docente.

2.3.1. La procedencia de los alumnos

Igual que el mundo de la educación superior está en un continuo proceso de reforma y reestructuración –como vimos en los apartados anteriores–, también los niveles inferiores de enseñanza son reformados con cierta frecuencia. Estos cambios pueden afectar a la docencia universitaria, en cuanto que condicionan la preparación de los alumnos que llegan a nuestras universidades. En la actualidad, el sistema educativo español consta de los niveles de cualificación que se muestran en la figura 2.7. Recientemente, esta situación se ha visto alterada por la Ley Orgánica de Calidad de la Enseñanza (LOCE), aprobada por el Congreso de los Diputados en diciembre de 2002, que configura un nuevo sistema educativo para los niveles inferiores de enseñanza. Paliando algunas deficiencias del sistema actual, se mantiene básicamente un esquema de niveles parecido al de la figura 2.7, y establece como una de sus prioridades elevar el nivel de formación y conocimientos de los alumnos.

Se puede observar que existen fundamentalmente dos itinerarios para el ingreso de los alumnos a la Universidad: a través del Bachillerato, o a través de los Ciclos Formativos de Formación Profesional (FP). Los estudios de Bachillerato ofrecen una formación de carácter más general, dentro de alguna de las opciones o especialidades dispuestas por la LOGSE (opciones que son modificadas parcialmente por la LOCE). Las opciones prioritarias para los estudios de informática son la rama científico-técnica y la de ciencias de la naturaleza. Por su parte, la formación de FP está más orientada a la integración de los alumnos en el mundo laboral, por lo que es de carácter más práctico y especializado.

Estas dos grandes "fuentes de alumnos" hacen que los estudios universitarios, sobre todo en los primeros cursos, deban adecuarse en métodos y contenidos a la doble procedencia. Tradicionalmente, los alumnos de Bachillerato tienen ventaja en el razonamiento abstracto, teórico y metódico, mientras que los procedentes de FP tienen más habilidades de aplicación de conocimientos y posiblemente una formación previa más relacionada con la materia[29]. No obstante, el efecto *homogeneizador* de los primeros cursos hace que esta diversidad de procedencias sea menos pronunciada en las asignaturas de segundo curso; en parte debido también a los altos índices de abandono de los primeros cursos.

Por otro lado, como veremos más adelante, el acceso al primer ciclo de la Ingeniería superior sólo es posible para los alumnos con el título de Bachillerato. Los alumnos procedentes de FP pueden acceder exclusivamente a las Ingenierías Técnicas. Este factor debe ser tenido en cuenta para la materia objeto de estudio, situada en el segundo curso de primer ciclo de la Ingeniería superior.

Normativa legal sobre el acceso a las Universidades

Las Leyes y Reales Decretos que rigen los procedimientos para el acceso a las Universidades tienen como finalidad garantizar los principios de igualdad, mérito y capacidad de los estudiantes, independientemente de su condición social o procedencia geográfica. Los artículos 42, 43 y 44 de la LOU tratan estos aspectos, y son desarrollados normativamente en los siguientes Reales Decretos:

[29] Si bien, como apuntan diferentes profesores, esta ventaja inicial puede tener un doble filo: *"El hecho de que esta ventaja se convierta en un serio impedimento para su progresión* [...] *depende de su actitud.* [...] *Aquellos que adoptan una actitud de rebeldía y desprecio ante los nuevos conocimientos que se les intenta transmitir, y se aferran a lo que ya saben como únicos conocimientos necesarios y suficientes* [...], *suelen acumular convocatorias suspensas una tras otra sin mostrar ningún síntoma de cambio de actitud ni propósito de enmienda entre una y la siguiente"*, [Montoya'01].

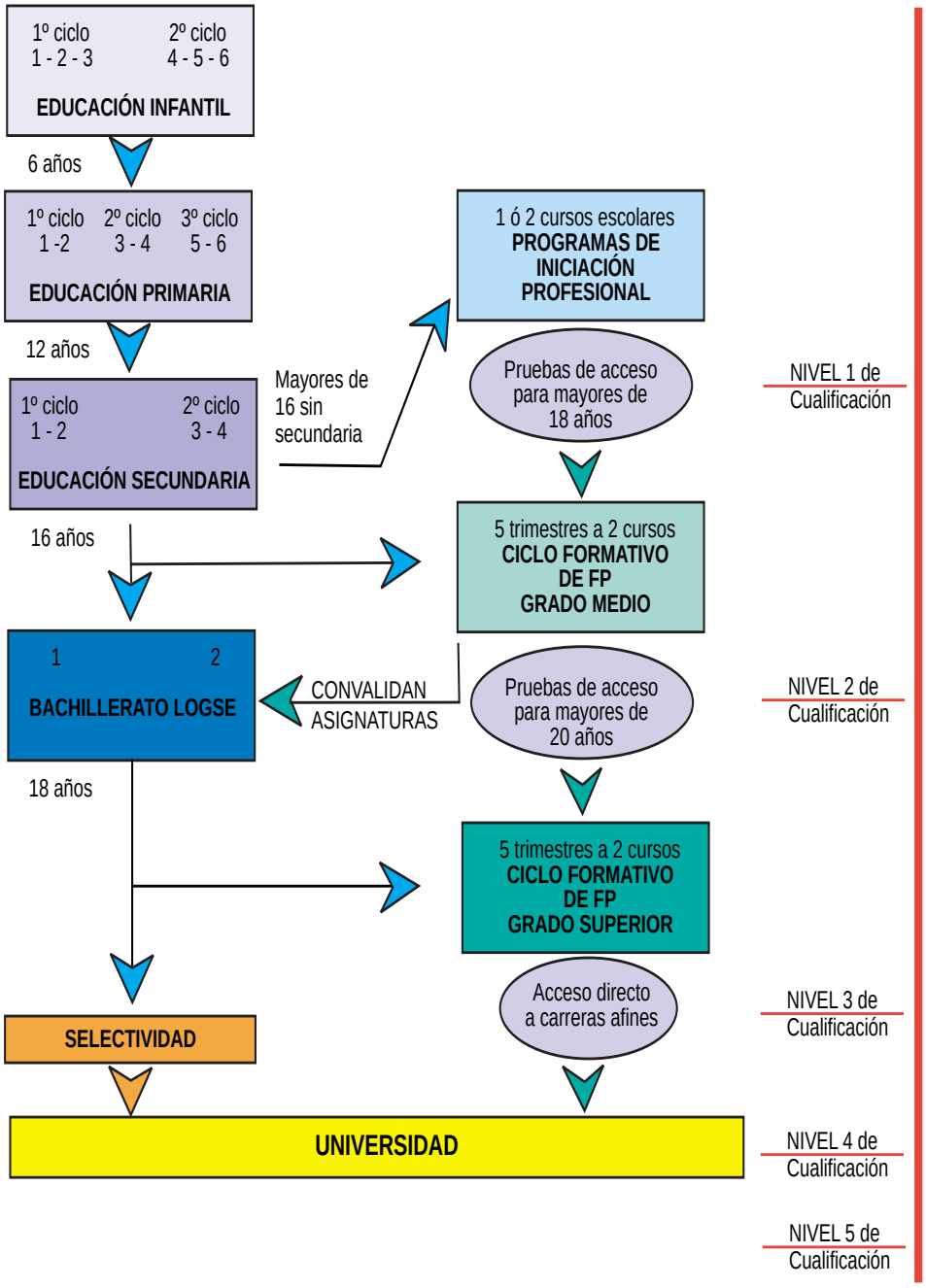


Figura 2.7: El sistema educativo español actual, en todos sus niveles de cualificación. Extraído de la Guía de Titulaciones de la UM, curso 2003/04, [TitulUM'03].

- Real Decreto 1640/1999, de 22 de octubre (BOE de 27 de octubre), posteriormente modificado por el Real Decreto 990/2000, de 2 de junio (BOE de 3 de junio), por el que se regula la prueba de acceso a estudios universitarios.

- Real Decreto 69/2000, de 21 de enero, (BOE de 22 de enero) por el que se regulan los procedi-

mientos de selección para el ingreso en los centros universitarios de los estudiantes que reúnan los requisitos legales necesarios para el acceso a la Universidad.

El primero de ellos es el que establece las características y condiciones de la prueba de acceso a la Universidad –denominada normalmente **selectividad**–, que sirve para valorar los conocimientos y la madurez académica de los alumnos de Bachillerato. Su superación es imprescindible para el acceso a la mayoría de las titulaciones universitarias. Junto con las calificaciones obtenidas en el Bachillerato, el resultado de esta prueba sirve para determinar los alumnos que tendrán acceso a los estudios universitarios. A corto plazo, con la aprobación de la LOCE, está prevista la supresión de esta prueba, que podrá ser sustituida por los procedimientos de admisión que las universidades estimen oportunos. En el nuevo escenario, los alumnos deberán realizar una **Prueba General de Bachillerato** al finalizar la educación secundaria, necesaria para la obtención del título de Bachiller. En la práctica, esta prueba es similar a la actual selectividad, pero con una diferencia sustancial: es requerida para todos los alumnos, independientemente de que vayan a continuar sus estudios en la Universidad o no.

El segundo de los decretos establece los mecanismos de ingreso a las Universidades, desde los diferentes itinerarios de entrada posibles. Este Decreto implanta el denominado **distrito abierto**, en virtud del cual los estudiantes pueden solicitar plaza en la universidad de su elección, con independencia de la universidad en la que hayan superado la prueba de acceso. Con esta medida se pretende favorecer tanto la movilidad estudiantil como el acceso a las enseñanzas por las que cada estudiante tenga más vocación. Se garantiza la igualdad de oportunidades de los alumnos, en función de su capacidad y no de su procedencia geográfica. Si bien en cursos anteriores ya se reservaba cierto porcentaje de las plazas para alumnos que realizaban las pruebas de acceso en otras universidades, en la actualidad el 100 % de las plazas ofertadas por la Universidad de Murcia lo son en régimen de distrito abierto.

Cupos de acceso y notas de corte

Además de las normativas generales antes comentadas, las Universidades tienen la capacidad de establecer el número máximo de alumnos admitidos en cada curso para las diferentes titulaciones, en función de sus posibilidades materiales y docentes. El Boletín Oficial de la Región de Murcia (BORM) publica anualmente un listado donde se concretan estas ofertas de plazas para las titulaciones de las universidades públicas de la región (UM y UPCT). El total de estas plazas se distribuye, por porcentajes, entre una serie de cupos predefinidos, en función de la procedencia formativa de los estudiantes. Los candidatos pueden acceder a la FIUM por alguno de los siguientes cupos[30]:

- **Cupo 1: Bachiller, COU o Selectividad.** Para el ingreso, los solicitantes han de tener superada la selectividad. En caso de que el cupo no se complete, y sólo en las Ingenierías Técnicas, también pueden acceder aquellos solicitantes que hayan aprobado el COU o el Bachillerato LOGSE. El número de plazas reservadas para este cupo es el resultante de quitar del total las cubiertas por los cupos restantes (los cupos de 2 a 6). En consecuencia, este cupo tiene reservado un mínimo del 54 % de las plazas.

  A los efectos de este cupo, los nacionales de los estados miembros de la Unión Europea tienen el mismo tratamiento que los solicitantes de nacionalidad española. Para el resto de alumnos extranjeros se reserva un 5 % del total de las plazas, siempre que hayan superado las Pruebas de Acceso a las universidades españolas en el curso precedente.

[30] Datos extraídos de la Guía de Estudios de la FIUM: `http://www.fi.um.es/estudios`

- **Cupo 2: Formación Profesional de Segundo Grado, Módulo Profesional de Nivel III o equivalente.** Estos solicitantes sólo tienen acceso a los estudios de ciclo corto, es decir las Ingenierías Técnicas, y se les reserva un 30 % de las plazas ofertadas.

- **Cupo 3: Titulación Universitaria o Estudios Superiores asimilados.** Para estos solicitantes se reservará un 5 % de las plazas ofrecidas.

- **Cupo 4: Minusválidos.** Se reserva un 3 % de las plazas para aquellos solicitantes que tengan reconocido un grado de minusvalía igual o superior al 65 %, o con menoscabo total del habla o pérdida total de audición.

- **Cupo 5: Deportistas de alto nivel.** Se reserva un 3 % de las plazas para aquellos solicitantes que acrediten su condición de deportista de alto nivel. Hay que notar que, hasta la fecha, ningún alumno en la FIUM ha entrado en este cupo.

- **Cupo 6: Mayor de 25 años.** Los alumnos mayores de 25 años que hayan superado las pruebas de acceso a la UM y hayan optado por realizar alguna Ingeniería, tienen garantizado un acceso directo.

En la actualidad, para el curso académico 2003/04, la FIUM ofrece 140 plazas para ITIG, 140 plazas para ITIS y 140 plazas para el primer ciclo de II. No hay límite de plazas para el segundo ciclo de II, en el que, hasta la fecha, han podido continuar sus estudios todos los alumnos de las titulaciones técnicas que así lo han elegido. Para las titulaciones con límite de plazas, el orden de acceso viene dado por la nota de selectividad o de los estudios correspondientes. En la tabla 2.20 se muestran las notas de corte para las titulaciones ofertadas por la FIUM, durante el curso 2002/03. Debemos hacer notar que durante este curso el número de plazas ofertadas en II era de 125.

Titulación	**Plazas**	**Cupo Bachill.**	**Cupo FP**	**Cupo Titulados**	**Cupo 25 años**
I.T.I. Gestión	140	5.63 (Jun.)	5.00 (Sep.)	1.90	5.00
I.T.I. Sistemas	140	5.40 (Jun.)	5.00 (Sep.)	1.19	5.00
I. Informática	125	5.72 (Sep.)	-	1.00	5.00

Tabla 2.20: Número de plazas y notas de corte de los diferentes cupos, para las titulaciones ofertadas por la FIUM, durante el curso 2002/03.

En concreto, las cifras de alumnos que accedieron a los estudios de nuestra Facultad, durante el pasado curso 2002/03, por cada cupo, son mostradas en la tabla 2.21. Por uno u otro motivo, los números totales de alumnos admitidos pueden ser algo mayores que los establecidos a priori.

Titulación	**Cupo Bachill.**	**Cupo FP**	**Cupo Titulados**	**Cupo 25 años**	**Cupo Extranj.**	**Total**
I.T.I. Gestión	125 (89 %)	15 (11 %)	1	0	0	141
I.T.I. Sistemas	115 (80 %)	27 (19 %)	1	1	0	144
I. Informática	118 (98 %)	0	0	0	2	120

Tabla 2.21: Número de alumnos, por cupo, que accedieron a las titulaciones ofertadas por la FIUM, durante el curso 2002/03.

Podemos observar en las tablas 2.20 y 2.21, que a pesar de la alta diversidad de procedencias que los mecanismos de acceso permiten, claramente el mayor porcentaje de alumnos que ingresan en las tres titulaciones provienen del Bachiller. En la II este porcentaje alcanza el 98 % del total, ya que los alumnos de FP sólo pueden optar por las titulaciones de ciclo corto; ningún alumno entró tampoco durante el curso pasado a través de los cupos de titulados, minusválidos o mayores de 25 años.

En cuanto a las notas de corte, que aparecen en la tabla 2.20, podemos extraer dos conclusiones. En primer lugar, la demanda es sensiblemente mayor para las Ingenierías Técnicas, que cubrieron completamente su cupo en la convocatoria de junio. Y, entre ellas, las notas de corte de ITI de Gestión son algo mayores que las de ITI de Sistemas. Estos datos muestran el interés de muchos alumnos por realizar una titulación de ciclo corto que les permita una rápida incorporación al mercado laboral. La posibilidad de continuar después con los estudios de segundo ciclo, tras haber obtenido un título intermedio de Ingeniero Técnico, son otro aliciente para empezar en alguna de las titulaciones técnicas aunque el alumno tenga intención de aspirar al grado de Ingeniero superior.

La segunda conclusión que podemos extraer es el bajo nivel general de las notas de corte. Es más, si comparamos los datos con los de otros años, podemos ver que el descenso ha sido progresivo. Por ejemplo, durante el curso 2001/02 los cupos de II se completaron en la convocatoria de junio. Estos datos se enmarcan dentro de una estrategia de crecimiento continuo y mantenido de la oferta de plazas en nuestra Facultad, que no se ha visto correspondida con un aumento acorde de la demanda. El descenso en el nivel de conocimientos de los alumnos ha sido ya advertido tanto en nuestra Universidad[31] como a nivel nacional[32]. Entre las carencias más destacadas están las relativas a la expresión oral y escrita, y a la base matemática, todos ellos conocimientos esenciales para los estudios universitarios por su carácter instrumental.

En el sentido de mejorar estas capacidades de los alumnos se encuentran algunas actuaciones llevadas a cabo en la FIUM, como las comentadas en el apartado 2.1.4 relativas a la oferta de cursos de promoción educativa. Pero es evidente que la solución de estas deficiencias sólo puede provenir de los propios niveles inferiores de educación.

2.3.2. Análisis retrospectivo del alumnado

Hasta ahora hemos analizado la procedencia, antecedentes formativos y procedimientos de acceso de los alumnos a la Universidad en general y a la FIUM en particular. En este apartado vamos a centrarnos en el contexto más próximo y específico, compuesto por el alumnado de la materia objeto de concurso. En esencia se trata de un estudio retrospectivo, basado en la experiencia docente personal del candidato, hasta la fecha, en la impartición de la asignatura "Ampliación de Algoritmos y Estructura de Datos".

El número de alumnos matriculados en la asignatura "Ampliación de Algoritmos y Estructura de Datos" ha crecido de forma constante desde que comenzara a impartirse el segundo año de II en el curso 1997/98. El candidato se incorporó a la Universidad y a la docencia de esta asignatura a partir del siguiente curso. Las cifras de matriculados y aprobados, desde entonces hasta el momento actual, son mostradas en el gráfico de la figura 2.8.

El incremento en el número de alumnos se justifica por la implantación progresiva del primer ciclo de II. El bajo número de alumnos que accedían a estos estudios en los primeros años tras su implantación, y los altos índices de fracaso en los primeros cursos de II, han dado lugar a este

[31] *"En general, parece que entre el profesorado universitario existe la impresión de que, por alguna razón, el nivel educativo con el que los alumnos llegan a la Universidad ha decrecido en los últimos años"*, [López'03].

[32] Siendo el máximo exponente de este hecho la aprobación de la LOCE, en el intento de *"elevar el nivel de formación y de conocimientos de los alumnos que, según informes nacionales e internacionales, presenta carencias en las materias básicas"*.

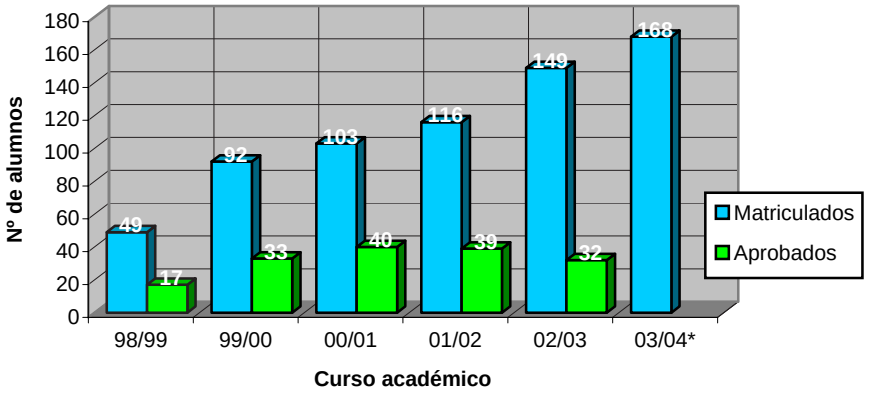


Figura 2.8: Número de alumnos matriculados y aprobados en "Ampliación de Algoritmos y Estructura de Datos". *Los datos del curso 2003/04 corresponden a la asignatura "Algoritmos y Estructuras de Datos" del plan nuevo.

fenómeno de crecimiento lento y continuado. Como consecuencia, la masificación en el aula es cada vez mayor, lo cual es ciertamente un obstáculo para la introducción de dinámicas de trabajo centradas en el seguimiento del trabajo individual de los alumnos. Estimativamente, el nivel de asistencia a clase se puede cifrar en un rango entre el 40 % y el 60 % sobre el número de alumnos matriculados en la asignatura. El porcentaje es significativo si tenemos en cuenta sólo los alumnos de nueva matrícula.

El número de aprobados, a lo largo de las tres convocatorias del curso, se ha mantenido de forma más o menos constante en torno al 35 % del número de matriculados. Debemos tener en cuenta que los datos del curso 2002/03 no recogen la convocatoria de diciembre, al haberse producido la adaptación de la mayoría de los alumnos al plan nuevo. En concreto, las notas de examen de las seis últimas convocatorias[33] son mostradas en la figura 2.9. La gráfica presenta una típica forma gaussiana, centrada en el 4,5. Es indudable que el formato de la asignatura, que concentraba toda la teoría en un solo semestre, es un factor añadido a la mayor o menor dificultad implícita de la materia.

El porcentaje de alumnos con una nota final de aprobado o mayor, en relación al número de presentados –en las mismas convocatorias de la figura 2.9– es del 50,3 %. Obviamente, no todos los alumnos se presentan al examen, variando el porcentaje de alumnos presentados entre el 25 % y el 40 % de los matriculados; valores realmente bajos, pero comunes en el resto de asignaturas de la titulación. Por otro lado, en referencia a los dos grandes bloques de la asignatura, típicamente la parte de estructuras de datos resulta más *asequible* para los estudiantes que la de algoritmos. Mientras que la nota media total de los ejercicios de la primera parte es de 5,93 puntos, sobre 10, la correspondiente a algorítmica es alrededor de 4,63, siendo el diseño de algoritmos el punto más débil, con una media que no supera el 4,3.

Al comienzo de los cursos 2001/02 y 2002/03 se realizaron sendas encuestas/tests de evaluación a los alumnos, con el objetivo de investigar sus preferencias, situación personal y nivel de conocimientos de partida. Los resultados de estas encuestas se pueden consultar en el apéndice B. Analizando los datos, podemos extraer algunas conclusiones de utilidad:

- Respecto a la formación de los alumnos **antes de comenzar sus estudios universitarios**, la gran mayoría, el 81 %, admite que no tenía muchos conocimientos de informática o que usaba exclusivamente aplicaciones de usuario. Sólo el 19 % dice que sabía programar, aunque ninguno

[33] Sin considerar las dos últimas, por las circunstancias de cambio de planes en que se producen.

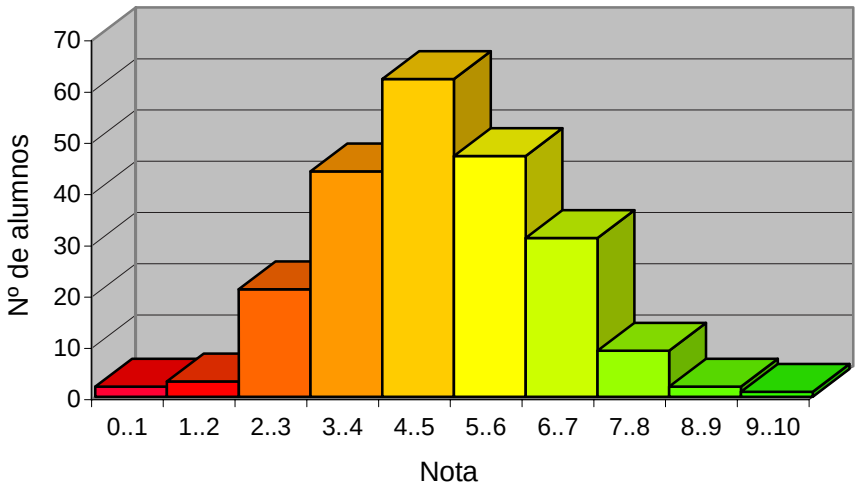


Figura 2.9: Notas de examen de la asignatura "Ampliación de Algoritmos y Estructura de Datos".

reconoce amplios conocimientos. Esta homogeneidad de partida es debida, principalmente, a que sólo los alumnos de Bachillerato pueden acceder a los estudios de II, como antes comentamos.

Sin embargo, el dato contrasta con la alta variabilidad en la experiencia de programación: el 34 % ha escrito programas de más de 4 páginas de largo, y un 7 % de más de 8 páginas. Una parte importante de este ítem puede ser asociado con la *vocación* –o ausencia de la misma– por los estudios de informática. La existencia de alumnos que acceden a los estudios de informática motivados por las amplias expectativas de empleo, pero carentes de vocación por la disciplina, es un hecho evidente. Es muy significativo el 30 % de alumnos que, al llegar a segundo curso, reconocen que el programa más largo que han escrito no sobrepasa las 100 líneas de código.

- El **nivel de conocimientos al iniciar segundo curso** presenta una gran diversidad. Como dato destacado, sobre el 20 % de los alumnos se matriculan en la asignatura sin tener aprobadas las asignaturas de programación de primero. Posiblemente, este porcentaje se vería aumentado con los alumnos que no realizaron las encuestas. Lógicamente, el futuro de estos alumnos en la asignatura es poco prometedor, teniendo en cuenta que AED es una continuación natural de esas asignaturas, como vimos en el apartado 2.2.4. Entre los demás alumnos, las notas de las asignaturas de primer curso se reparten de forma aproximada en un 60 % de aprobados, 25 % de notables y 15 % de sobresalientes o matrículas de honor.

  Estos valores están en consonancia con el test de conocimientos, que en general muestra un nivel medio-bajo. El nivel de partida es algo mayor en la parte de estructuras de datos que en la de algoritmos. Los datos muestran también una escasa formación matemática, aunque en este apartado hemos de tomar la encuesta de forma relativa.

- En lo que respecta a la **situación actual y expectativas** en la asignatura, volvemos a encontrar diferentes actitudes entre los alumnos. Casi un 60 % de los mismos se *conformaría* con un aprobado, mientras que un 25 % aspira a un notable, y un 15 % a una nota mayor. Teniendo en cuenta las notas de primer curso, estos datos reflejan un bajo deseo por *progresar* mejorando los resultados de otros años. Aun así, no es despreciable el número de alumnos que pretenden alcanzar un nivel alto de conocimientos en la asignatura.

Por otro lado, es bien conocido el hecho de que muchos alumnos de las titulaciones de informática combinan sus estudios con el desempeño de un trabajo. Esta situación es previsiblemente menor en la Ingeniería Superior, pero aún de cierta trascendencia. El 74 % de los alumnos no trabaja actualmente mientras cursa sus estudios[34], mientras que el 4 % tiene un trabajo más o menos estable. El 22 % realiza trabajos de forma esporádica.

- Finalmente, en cuanto a las **preferencias del alumnado** sobre la asignatura, las encuestas se centraron fundamentalmente en el lenguaje de programación y en el tipo de prácticas. Durante los años en que fueron realizadas, los alumnos habían estudiado Java en los cursos previos, por lo que muestran claramente sus preferencias por este lenguaje. A pesar de ello, se puede comprobar que el dominio del lenguaje es escaso, debido a que se usó este lenguaje orientado a objetos para explicar el paradigma imperativo.

En definitiva, a gran escala, podemos clasificar la actitud de los estudiantes[35] frente a la asignatura en dos grandes grupos: la de aquellos cuyo objetivo es alcanzar el mínimo para obtener el aprobado, y la de los que aspiran a lograr la mejor nota posible. En cierta medida, el diseño de la asignatura debería tener en cuenta esta doble vertiente, estableciendo un nivel mínimo exigible para todos los alumnos, a la vez que dando un margen de mejora y ampliación de los conocimientos a los interesados en ello. La posibilidad de incluir partes *opcionales* en las prácticas podría ser una manera de lograrlo. Asimismo, la metodología docente usada, el clima de clase y las tareas propuestas deben motivar en todos los alumnos la actitud por lograr las mayores cotas de conocimiento.

2.3.3. Cómo aprenden los alumnos

El objetivo final de la docencia universitaria es, esencialmente, lograr el aprendizaje de los alumnos. Pero, ¿cómo aprenden los alumnos? Para poder responder a esta pregunta haremos un breve repaso de las teorías psicológicas que explican el proceso de aprendizaje adulto, y las recomendaciones pedagógicas que de ello se desprenden.

Tradicionalmente, el proceso de enseñanza/aprendizaje se ha entendido como una transmisión de conocimientos, entre el que enseña y el que aprende. En este modelo, los estudiantes actúan como receptores pasivos de la información, limitándose la labor del profesor a exponer y explicar los contenidos. Frente a estas tesis, surge la teoría de aprendizaje denominada **constructivismo** –la predominante en la actualidad–, que se desarrolla *"a partir de las aportaciones de diferentes enfoques teóricos ensamblados unos con otros"* [Doménech'99]. De acuerdo con esta teoría, el aprendizaje no es un mero proceso de transmisión de conocimientos, sino que los alumnos construyen sus propias estructuras mentales de conocimiento a partir de la información que reciben. Por lo tanto, el aprendizaje es, ante todo, un proceso activo de elaboración y reelaboración.

Según el modelo constructivista, el proceso de construcción del conocimiento tiene un carácter *recursivo*, de manera que cada nueva idea es asimilada por el que aprende a través de **modelos previamente conocidos**. En este proceso se aplican mecanismos de relación, comparación, contraste y analogía de lo que se sabía anteriormente con la nueva información. Por ejemplo, en el caso concreto de la programación, para aprender el concepto de "variable" un estudiante puede construir un modelo mental de "caja donde se almacenan cosas". Aplicando la analogía, el estudiante puede mejorar su comprensión del nuevo concepto. En el caso de las variables, el alumno puede partir del hecho de

[34] Aunque, de ellos, el 66 % tendría la intención de hacerlo si pudiera.

[35] Clasificamos las actitudes, no los estudiantes en sí. No podemos cometer el error simplista de pensar que los alumnos pueden ser clasificados de antemano en una u otra categoría.

que el contenido de una caja se puede observar y modificar, y por lo tanto deducir que una variable también se puede leer y escribir.

Un error de aprendizaje es debido a la construcción de un modelo no viable. El anterior modelo de cajas para las variables, por ejemplo, puede no ser viable para explicar una asignación A:= B, si se interpreta que la caja B queda sin contenido. Este problema debería ser detectado por el profesor, orientando hacia la construcción de un modelo más adecuado.

La aplicación del constructivismo a la enseñanza de la informática es estudiada en [Ben-Ari'98]. El autor propone que los educadores de informática deberían estudiar y aplicar la teoría del constructivismo, y propone unas guías prácticas para su uso:

> ≪ *Independientemente de la técnica de enseñanza (lección magistral, laboratorios, ejercicios), debes analizar el cambio cognitivo que pretendes que tenga lugar en los estudiantes y estructurar la actividad para lograr ese propósito. Pretender una mera transmisión de conocimientos carece de sentido.*
>
> - *Debes usar tu propio conocimiento como experto para encontrar lo que es necesario saber a priori para poder construir modelos viables del material que enseñas. Debes asegurarte de que los estudiantes tienen ese conocimiento a priori.*
> - *En cualquier curso estarás enseñando en un nivel de abstracción específico; debes presentar explícitamente un modelo viable del nivel de abstracción que está un paso por debajo.*
> - *Cuando un estudiante comete un fallo o exhibe un error de comprensión, debes suponer que el estudiante tiene un modelo mental más o menos consistente, pero no viable. Tu tarea es descubrir ese modelo y guiar al estudiante para su modificación.*
> - *Debes proveer las máximas oportunidades posibles para la reflexión individual (por ejemplo, mediante análisis de errores) y para la interacción social (por ejemplo, con prácticas en grupos).* ≫

A modo de síntesis personal, consideramos que se pueden extraer tres conclusiones interesantes de la teoría del constructivismo, que deberían ser tenidas en cuenta en la enseñanza de la materia. Primera, que el aprendizaje es siempre un proceso activo y no pasivo –aun en la lectura de un texto–, y por lo tanto requiere motivación y esfuerzo por parte del estudiante. Segunda, que la utilización de analogías es una técnica muy útil. Encontrar las analogías más apropiadas y presentarlas explícitamente a los alumnos es una buena forma de orientarlos a la comprensión de la materia. Tercera, que resulta fundamental que el profesor "se ponga en la cabeza del alumno", es decir, que trate de averiguar los procesos mentales que están ocurriendo en él, para poder guiarlo de la forma más adecuada.

Como comentábamos al principio de la sección, todas estas consideraciones serán tenidas en cuenta en el capítulo 8, donde se tratan los aspectos pedagógicos del presente proyecto docente.

2.4. El contexto social y profesional

En su reciente y relativamente corta historia, el desarrollo de las conocidas como Tecnologías de la Información y las Comunicaciones (TIC) ha producido un impacto en nuestras sociedades tan solo comparable a los grandes avances históricos, como la revolución industrial del siglo XIX. Indudablemente, en este nuevo escenario la informática desempeña un papel esencial, como elemento central e insustituible de muchas de estas tecnologías. Los cambios producidos han afectado a todos los órdenes

de la vida: cultural, social, laboral, económico; y las tendencias que prevalezcan en el futuro repercutirán, al mismo tiempo, en la manera en que planificamos y organizamos la docencia de la informática en la Universidad. A modo de ejemplo, hoy en día podemos dar por hecho que nuestros alumnos tienen un ordenador en casa con el que pueden hacer sus prácticas, algo que no ocurría cuando la FIUM empezó su andadura. Es más, muchos de ellos disponen de conexión a Internet, y la aprovecharán para contrastar con otras fuentes la información que les proporcionamos, para buscar material extra para el estudio de la teoría o para la resolución de las prácticas.

En esta sección empezaremos haciendo un breve análisis de la penetración de las TIC en nuestra sociedad, con especial atención a los datos referidos a la Región de Murcia. A continuación estudiaremos las salidas profesionales y la situación actual del mercado laboral que se presenta a los titulados en informática al salir de nuestra Universidad. Finalmente, ofreceremos una visión más amplia –tanto en el tiempo como en el espacio– de las nuevas perspectivas que se abren en el mundo profesional de las TIC, revisando las previsiones y las recomendaciones contenidas en el informe elaborado por Career Space, un consorcio en el que participan la Comisión Europea, las principales compañías europeas en TIC y un conjunto representativo de instituciones de educación superior.

2.4.1. Las cifras de la sociedad de la información

La introducción de las TIC en España, tanto en los hogares como en el sector productivo, es un hecho cada vez más notable. Pero, ¿hasta qué grado? El Ministerio de Ciencia y Tecnología realiza periódicamente una serie de encuestas sobre la influencia de estas nuevas tecnologías en nuestra sociedad, dando como resultado un conjunto de indicadores que ayudan al análisis de la evolución temporal y en el entorno europeo, [MCYT'03]. Estos indicadores hacen referencia a la industria de las TIC, las infraestructuras, los terminales de acceso existentes, los servicios, usos más relevantes, precios e inversiones, mercado en España y la administración en línea.

A continuación vamos a hacer un resumen de los datos más significativos. Debemos señalar que, por las características la información que recoge, la mayoría de los indicadores disponibles se refieren al año 2002. Los datos que se refieren a la Región de Murcia se muestran en la tabla 2.22.

Indicador	Murcia	España	Europa
Mercado interior neto de Informática (millones de euros)	101,1	8.340,8	-
Mercado interior de Informática *per cápita* (euros)	91,9	216,6	-
Variación del mercado de Informática 02/01 (%)	-1,81	-6,67	-
Hogares con PC	100.274	4.405.638	-
Hogares con PC (%)	27,7	32,2	35,12
Incremento de hogares con PC 02/01 (%)	23,66	11,42	-
Población con acceso a Internet y usuarios del último mes (%)	23,6	25,7	-

Tabla 2.22: Indicadores del sector de las TIC en Murcia y en España, durante el año 2002, [MCYT'03].

1. **Industria de las TIC.** El mercado de las TIC ocupa un papel cada vez más destacado en las economías de todos los países. Durante el año 2002, el mercado de productos y servicios de las TIC alcanzó el 5,8 % del Producto Interior Bruto (PIB) en España. De esta cantidad, el 33 % corresponde a los productos y servicios informáticos, y el resto al mercado de las telecomunicaciones. Mientras que el crecimiento de este último ha sido más notable, las cifras del sector informático indican un estancamiento en los últimos 4 años. La situación en relación al contexto europeo no es muy halagüeña; el gasto informático *per cápita* fue de 291 euros, lo cual es 2,6 veces menos que la media europea y 5 menos que la de EE.UU.

2. **Infraestructuras.** Los datos referentes a infraestructuras, de teléfonos fijos, móviles y de redes de nacionales de investigación, indican una mayor proximidad a los niveles medios de la Unión Europea. Las cifras de 2002 indican que existían 43,7 líneas telefónicas por cada 100 habitantes, y aproximadamente el doble de abonados a telefonía móvil.

3. **Terminales de acceso.** Este apartado es el que más claramente refleja la introducción de las tecnologías de la información en la sociedad española. El número de ordenadores por hogar en España se sitúa en el 32,3 %, en 2002, con un crecimiento superior al 10 % anual. En Murcia esta cifra no supera el 28 %, aunque el incremento es el doble que la media nacional. El porcentaje de hogares con acceso a Internet es alto dentro del contexto europeo, teniendo el 31 % de los hogares acceso a Internet, de los cuales el 55 % es con ADSL y el 19 % con cable-módem. En cuanto a las empresas, el 82,6 % tiene acceso a Internet, mientras que casi el 33 % de los trabajadores usan algún ordenador en su trabajo. Estas cifras son bastante significativas, y muestran en general una buena posición de España en su contexto europeo.

4. **Servicios.** El indicador más interesante de este apartado se refiere al porcentaje de profesionales de informática respecto al empleo total. Los últimos datos disponibles son de 2001, e indican que el 1,1 % de los empleos ocupados en España eran de informática. De nuevo, encontramos un crecimiento continuado, desde el 0,5 % de 1995, aunque aún algo lejos de la media europea del 1,8 %. Teniendo en cuenta el número de afiliados a la Seguridad Social durante aquel año (unos 16 millones de personas), el número de profesionales informáticos en España es de unos 170.000. El crecimiento continuo de 0,1 puntos cada año supone, aproximadamente, unos 16.000 nuevos empleos al año.

5. **Usos.** Este apartado contiene indicadores tan variados como el número de máquinas que han resultado infectadas por algún virus informático, el número de centros educativos conectados a Internet y el número de ordenadores por alumno. Casi el 40 % de los usuarios de Internet admite haber experimentado algún problema, siendo el 19 % infectados por un virus. Es preocupante el crecimiento continuo que estas cifras han tenido en los últimos 5 años. En la educación superior, el número de ordenadores por cada 100 alumnos es de 27, algo por encima de la media europea de 24, aunque lejos de los países más aventajados (como Irlanda con 80). De estos ordenadores, sobre el 53 % de los mismos están conectados a Internet.

6. **Precios e inversiones.** Los indicadores demuestran una impresión general bastante común: mientras que los precios de los productos y servicios de TIC son más altos en nuestro país que en el resto de los países de la Unión, las inversiones son muy inferiores. Por ejemplo, el precio de un acceso de cable en Alemania es de 21 euros, y en España de 153. Sin embargo, la inversión en Alemania, 10.869 millones de euros, es casi 5 veces mayor que en España.

7. **Mercado interior de España.** Los indicadores de este apartado reflejan la distribución de la facturación entre los distintos sectores productivos que componen el ámbito de las TIC. Por su interés, mostramos los datos en la tabla 2.23. Se puede ver que la Informática constituye casi el 50 % del mundo de las TIC. Y, dentro de la informática, los servicios informáticos son claramente la actividad con mayor facturación. A mucha distancia se coloca la facturación en hardware y software. Estas cifras ofrecen a nuestros titulados unas expectativas de trabajo fundamentalmente en la prestación de servicios. Un dato algo preocupante es el descenso de la facturación, que es del 14,4 % en total y del 11,4 % en la informática. Sólo las actividades de software y consumibles han aguantado esta recesión generalizada.

Sector	2001	2002	%
Componentes electrónicos	**1.870,0**	**1.568,0**	**-16,1**
Componentes electrónicos	1.011,0	816,0	-19,3
Antenas	201,0	213,0	6,0
Subcontratación	658,0	539,0	-18,1
Electrónica de consumo	**1.608,0**	**1.610,0**	**0,1**
Audio	45,0	32,0	-28,9
TVC	1.386,0	1.383,0	-0,2
Vídeo	177,0	195,0	10,2
Electrónica profesional	**1.112,0**	**1.141,0**	**2,6**
Inst. y equipos didácticos	46,0	43,0	-6,5
Electrónica de defensa	274,0	279,0	1,8
Electrónica industrial	632,0	657,0	4,0
Electromedicina	52,0	50,0	-3,8
Radiodifusión y TV	108,0	112,0	3,7
Informática	**5.845,8**	**5.179,6**	**-11,4**
Hardware	1.053,4	517,8	-50,8
Software	613,0	652,9	6,5
Servicios informáticos	3.917,9	3.732,4	-4,7
Consumibles	261,5	276,5	5,8
Telecomunicaciones	**2.206,0**	**1.328,0**	**-39,8**
TOTAL	**12.641,8**	**10.826,6**	**-14,4**

Tabla 2.23: Distribución de la facturación de los distintos sectores de las TIC, en España, durante los años 2001 y 2002, [MCYT'03]. Millones de euros y tasas de variación interanual.

8. **Administración en línea.** Una de las preocupaciones del Ministerio es potenciar los servicios que las distintas administraciones ofrecen a través de Internet. Los datos indican altas cifras de uso de estos servicios, en consonancia con la situación de los países de la Unión.

En definitiva, los datos muestran un claro crecimiento de la penetración de las tecnologías informáticas en la sociedad española. Aunque las cifras están por debajo de las de otros países de nuestro entorno europeo, la tendencia apunta hacia un progresivo acercamiento. No obstante, los indicadores advierten también de la cierta recesión en la actividad productiva de las empresas propias de las TIC. A pesar de ello, la demanda de informáticos es cada vez mayor, debido posiblemente a otros sectores industriales que también necesitan informáticos. Entre estos, la actividad predominante es claramente la relacionada con la prestación de servicios.

2.4.2. Perspectivas laborales en la Región de Murcia

En consonancia con la buena situación de crecimiento del sector informático a nivel nacional y europeo, las expectativas laborales que se presentan a los titulados en la FIUM son bastante optimistas. En principio, las tres titulaciones de informática ofertadas se orientan a perfiles más o menos definidos. De acuerdo con la Guía Académica de la FIUM, [GuíaFIUM'03]:

- La **Ingeniería Técnica en Informática de Gestión** pone énfasis en una formación que permita al alumno el desarrollo de aplicaciones para las empresas y que favorezca su integración en los departamentos de informática de cualquier organización.
- La **Ingeniería Técnica en Informática de Sistemas** sugiere una formación orientada al desarrollo de aplicaciones de sistemas, esto es, aplicaciones que requieren un mayor conocimiento

del hardware y de algunos pilares teóricos de la informática.

- La **Ingeniería en Informática** proporciona a los alumnos una formación que les capacita para algunos trabajos de alta cualificación requeridos por las empresas, como son integración y administración de sistemas, gestión de proyectos, análisis y diseño de sistemas informáticos, gestión de redes, evaluación de arquitecturas, etc., además de estar preparado para trabajar en departamentos de I+D en las empresas, dedicarse a la investigación en la Universidad o en centros de investigación, y acceder a niveles altos de la administración pública.

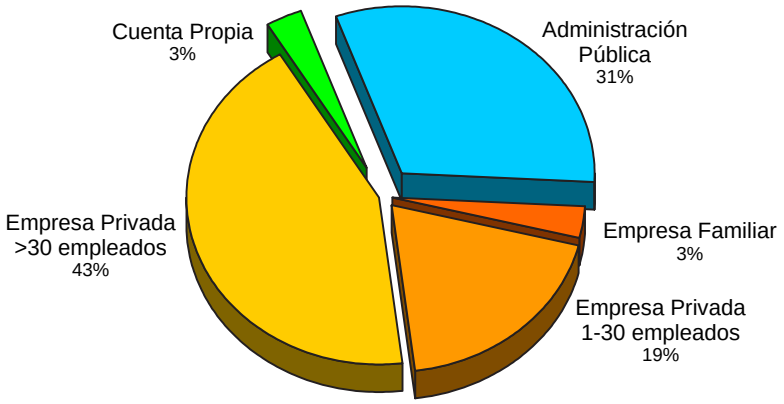
Pero en la práctica, como reconoce la propia guía, el mercado laboral no diferencia entre las tres titulaciones, o a lo sumo distingue entre las ingenierías técnicas y la superior. A menudo, incluso, existe un gran intrusismo profesional, al cual ha contribuido en gran medida la relativa juventud de la informática como disciplina académica independiente.

La Guía de titulaciones de la UM contiene información sobre las salidas profesionales de las diferentes titulaciones. De acuerdo con la misma, nuestros egresados están capacitados para trabajar en todos los departamentos de la empresa, aunque fundamentalmente formarán parte de los departamentos de informática. Las salidas profesionales que se mencionan para una u otra titulación son muy parecidas, siendo las más frecuentes: centros de cálculo, empresas de hardware, entidades financieras, de telecomunicaciones, de electricidad, de alta tecnología, de seguridad y consultoras informáticas. También es frecuente que se dediquen al ejercicio libre de la profesión como analistas y programadores. Dentro de las empresas, los puestos ocupados por los Ingenieros en Informática son del tipo: dirección de informática y departamentos de desarrollo, dirección y organización de proyectos informáticos y centros de proceso de datos, diseño, selección y evaluación de infraestructura de computación y lógica, mantenimiento de infraestructuras, optimización de métodos y medios de comunicación con el computador y los usuarios, concepción de proyectos y aplicaciones para su posterior análisis y ejecución, arquitectura, análisis y diseño de sistemas informáticos, técnica de sistemas, bases de datos y comunicaciones, consultoría técnica, auditoría informática, inteligencia artificial y nuevas tecnologías en general, investigación, formación y docencia.

En particular, en la actualidad la demanda de informáticos en nuestra región proviene de tres sectores claramente diferenciados [López'03]:

- **Organismos públicos**, dependientes de la Administración Central, la Comunidad Autónoma y los Ayuntamientos.
- **Empresas especializadas** en el sector informático, que proporcionan servicios a terceros –fabricantes, empresas de desarrollo de software y soluciones integradas, consultorías, enseñanza, etc.–. Podemos incluir dentro de este apartado los que se dedican al ejercicio libre de la profesión, como analistas y programadores.
- **Empresas no especializadas**, fundamentalmente pequeñas y medianas empresas –aunque también algunas grandes empresas ubicadas en el territorio regional–, pero que necesitan de las TIC en sus actividades cotidianas.

En comparación con otras carreras ofertadas por la UM, las titulaciones de informática de la FIUM están entre las que más altos índices de *empleabilidad* obtienen, lo cual es sin duda el mejor indicador de éxito académico posible. Según el último informe de la UM sobre la inserción laboral, [UM-lab'03], sobre el 87 % de los titulados en las tres carreras de informática durante los cursos del 1996/97 al 1999/00 se encuentran trabajando actualmente. La media de la Universidad se sitúa en el 79,22 %. La tabla 2.24 refleja el reparto de los distintos lugares donde trabajan actualmente los egresados, según el informe.



	Nº trabajad. encuestados	Admon. pública	Empresa familiar	Privada 1-30	Privada >30	Cuenta propia
I.T.I. Gestión	30	7	1	6	15	1
I.T.I. Sistemas	27	12	1	5	8	1
I. Informatica	10	2	0	2	6	0
Total	67	21	2	13	29	2
Porcentaje		31,3%	3%	19,4%	43,3%	3%

Tabla 2.24: Inserción laboral de los egresados de la FIUM en los cursos del 1996/97 al 1999/00, [UM-lab'03]. Lugares de trabajo.

El informe no distingue entre empresas especializadas en la informática o no (al ser un estudio de carácter general). A grandes rasgos, podemos establecer que aproximadamente 1/3 de los titulados trabajan en la Administración Pública y 2/3 en empresas privadas. Dentro de estas, la mayoría lo hacen en empresas de tamaño grande que pueden ser, a su vez, grandes industrias de las TIC o departamentos de informática de empresas de cualquier otro sector productivo. Aunque no está reflejado cuantitativamente en el informe, son también muchos los titulados de nuestra Universidad que deben marchar a las grandes capitales, Madrid o Barcelona, donde las perspectivas de trabajo y las condiciones laborales que se les ofrecen son mucho mejores que las que actualmente pueden encontrar en nuestra región.

Lo que sí muestra el informe, no obstante, es que nuestros egresados desempeñan trabajos que están muy relacionados con la formación recibida. Los datos indican que el 84 % de los antiguos alumnos admiten que su trabajo está totalmente relacionado con sus estudios. El porcentaje medio de la UM no llega al 64 %, siendo el más bajo el de las titulaciones de Humanidades con un 48,3 %.

Existe, por lo tanto, un amplio abanico de posibilidades que no se limita simplemente al primitivo perfil del informático como programador. En resumen, el **retrato robot** de un titulado en informática por la FIUM sería una persona que tarda menos de un año en encontrar empleo y trabaja en una empresa privada de tamaño grande; la empresa puede ser especializada o no, pero las labores desempeñadas son propias de la informática, fundamentalmente en lo que serían servicios informáticos.

2.4.3. El informe del consorcio Career Space

En un mundo cada vez más globalizado, limitarse al estudio del mercado laboral a nivel regional parece claramente insuficiente. Por esta razón, extenderemos el estudio a las expectativas profesionales que se presentan a nivel europeo y con una amplia visión de futuro.

Career Space es un consorcio fundado en 1999 por algunas de las principales compañías europeas en TIC: BT, Cisco Systems, IBM Europa, Intel, Microsoft Europa, Nokia, Nortel Networks, Philips Semiconductors, Siemens AG, Telefónica S.A. y Thales. Esta alianza nació con el propósito de impulsar el desarrollo de la denominada *Europa Electrónica* (*e*-Europe) estrechando el vacío existente entre las necesidades de los nuevos perfiles de trabajo y las posibilidades de formación que las universidades ofrecen. Junto con la Asociación Europea de las TIC (*European Information, Communications and Consumer Electronics Industry Technology Association*, o EICTA) y la Comisión Europea, trabaja para proponer ciertas adaptaciones en los currículos relacionados con dichas tecnologías con el fin de adaptarlos a las nuevas necesidades del mundo empresarial. Además, desde su fundación el consorcio ha contado con el apoyo de la comunidad académica, de manera que actualmente cuenta con la colaboración de más de 20 universidades europeas.

En el año 2001, Career Space elaboró una propuesta para el diseño de los nuevos currículos en TIC, [CareerSpace'01], tratando de responder a las nuevas necesidades del mercado laboral. En primer lugar, el documento analiza la situación de las industrias de las TIC, y los perfiles profesionales y competenciales que estas demandan. A continuación desarrolla una serie de recomendaciones curriculares, usando como base la estructura de titulaciones definida en el Espacio Europeo de Educación Superior, como tratamos en el apartado 2.1.2.

De acuerdo con el planteamiento de la propuesta, actualmente los planes de estudios presentan dos perfiles diferenciados y centrados en ingeniería eléctrica o en informática. En cierto sentido, estos perfiles suponen los dos extremos opuestos en el ámbito de las TIC: hardware y software. Pero aunque la industria aún necesita profesionales especializados en ambos perfiles, las necesidades se consideran de apenas un tercio del total de puestos de trabajo. Los dos tercios restantes deben ser cubiertos por personas con una cualificación que abarque el espacio entre ambos campos. Esta es una de las bases de la propuesta de Career Space. Concretando más, se articulan las necesidades de las industrias de las TIC en torno a seis puntos:

1. **Se necesitan nuevos programas en TIC**, que combinen la cultura ingenieril con la informática, y potencien otras habilidades fundamentales como las relativas al mundo de los negocios y a las capacidades personales. Los profesionales deben ser capaces de desarrollar soluciones orientadas a la aplicación, y a la implementación, manejo y soporte de sistemas TIC.

2. **Combinar elementos de ingeniería eléctrica e informática.** Como ya hemos visto, Career Space detecta la existencia de un importante vacío en el rango de cualificación que va entre uno y otro perfil: desde el puramente ingenieril (formado en aspectos de hardware, electrónica, etc.), hasta el informático tradicional (formado en software, computación, etc.). La industria está demandando personas preparadas en el amplio espectro que une ambos perfiles, sin que la oferta de titulaciones de las universidades respondan con una oferta acorde. En concreto, se distinguen los siguientes perfiles técnicos, ordenados por su proximidad a uno de los dos extremos:

   a) Ingeniería de radio frecuencia. (Cubierto por programas tradicionales de ingeniería.)

   b) Diseño digital.

   c) Diseño de aplicaciones de procesamiento de señal digital.

   d) Soporte técnico.

   e) Diseño de redes de comunicaciones.

   f) Diseño de productos.

   g) Integración y prueba de sistemas.

h) Ingeniero de comunicación de datos.

i) Multimedia.

j) Especialista de sistemas.

k) Consultor de negocio en tecnologías de la información.

l) Desarrollo de software y aplicaciones.

m) Diseño y arquitectura de software. (Cubierto por programas tradicionales de informática.)

3. **Se necesita una visión amplia de sistemas.** La formación de los titulados no puede ser una simple combinación de los elementos mencionados; se necesita que tengan una visión amplia, global y abstracta de los sistemas, comprendiendo las posibilidades que la tecnología ofrece. En la actualidad, existen grandes carencias en los graduados en cuanto a las habilidades para crear soluciones que abarquen sistemas completos.

4. **Se requieren conocimientos empresariales y de negocio.** Los sistemas de TIC son el núcleo de muchas compañías, e inseparables de sus propias actividades de negocio. Los graduados deben comprender los fundamentos del mundo empresarial, a lo cual ahora mismo se dedica escasa o nula atención.

5. **Se necesitan nuevas habilidades de comportamiento.** En la actualidad los proyectos implican equipos multidisciplinares trabajando en paralelo. Incluso los clientes tienden a involucrarse desde el principio en el desarrollo de los productos. Por lo tanto, los titulados en TIC deben ser capaces de desenvolverse con soltura en estos ambientes, donde las barreras nacionales son cada vez menos importantes. Además, en un campo donde las tecnologías evolucionan y cambian tan rápidamente es fundamental que los titulados estén preparados y concienciados de la importancia del aprendizaje a lo largo de la vida.

6. **Incrementar la movilidad entre la academia y la industria.** Se debe facilitar y fomentar la movilidad no sólo de los estudiantes sino también del personal docente. Se considera muy interesante la posibilidad de invitar a personal experto de las industrias de las TIC a impartir clases en las universidades, al mismo tiempo que permitiendo que miembros de las universidades se involucren en proyectos de las empresas durante periodos más o menos largos. Esto puede tener resultados beneficiosos para ambos mundos.

En resumen, los graduados en TIC deben tener una sólida base de habilidades técnicas propias de las culturas de la ingeniería y la informática, con un énfasis especial en la perspectiva sistémica amplia. Se necesita capacitar a los alumnos para el trabajo en equipo, con experiencias en proyectos reales en equipos donde diversas actividades tienen lugar en paralelo. Además se requieren conocimientos de economía, de mercados y del mundo de los negocios en general.

Para alcanzar estos objetivos el consorcio Career Space emite sus propuestas para el desarrollo de los nuevos programas en TIC. Se definen los perfiles antes expuestos, y se establece la distribución de la carga que deberían tener los planes de estudios entre los distintos tipos de habilidades y conocimientos. En la figura 2.10 se muestra gráficamente el reparto de competencias que debería ofrecer la formación universitaria. Los aspectos clave de la propuesta son los siguientes:

- Se necesita una **amplia base de conocimientos científico-técnicos**. Estos conocimientos sirven para crear un lenguaje común entre especialistas, y deberían ser más en anchura que en profundidad. La base en matemáticas y ciencias deberían constituir un 30 %, y hacer énfasis en los métodos de análisis y diseño. Por su parte, la base tecnológica debería ser otro 30 %, mostrando

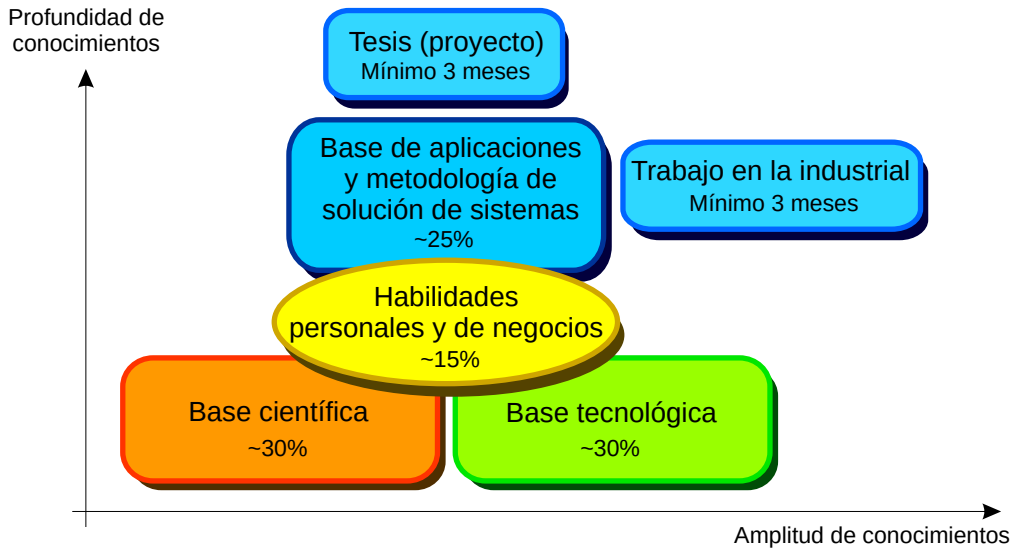


Figura 2.10: Ámbito de competencias del modelo de currículum propuesto por Career Space, [CareerSpace'01].

una visión amplia de las tecnologías existentes y las que están en desarrollo. Se reconoce la dificultad de rellenar huecos en este aspecto una vez comenzada la carrera profesional.

- Fuerte **relación entre las bases científica y tecnológica**. La distinción entre ambas no significa que deban ser enseñadas por separado. Se debe evitar en los estudiantes la idea de que la ciencia no tiene aplicación práctica, o que las tecnologías no tienen una fundamentación analítica.

- Base de **habilidades de aplicación y sistemas**. Además de la anterior base en anchura, es necesaria una profundización en un campo de especialización, con atención especial a la capacidad de resolución de problemas. Dada la alta complejidad de los dispositivos, equipos y sistemas actuales, el titulado debe tener la capacidad de ver cómo encajan sus soluciones dentro de un sistema global, y las relaciones de interface con las otras partes del mismo. A esta base debería reservarse el 25 % de la preparación de los alumnos.

- Las **habilidades personales y de negocio**. Los planes de estudio no deben caer en la tradicional separación entre *saber* y *hacer*, sino más bien hacer énfasis en la *capacidad de transferencia*, esto es, saber encontrar la oportunidad de aplicar lo aprendido. Para ello se sugiere la posibilidad de realizar proyectos en equipos, simulaciones comerciales, negociación, presentaciones, etc. Estas habilidades deberían estar embebidas en la enseñanza de asignaturas técnicas, y se recomienda una dedicación de al menos el 15 % del currículum.

- **Experiencia práctica en empresas**, como mínimo de 3 meses, aunque preferiblemente mayor. No es suficiente con tener muchos conocimientos académicos que sirven para *"superar los exámenes"*. Puesto que estos conocimientos deben ser aplicados en situaciones reales, los estudiantes deben tener experiencia en la aplicación de las técnicas y resolución de problemas en el mundo real. Este periodo de trabajo en la industria podría también ayudar a los estudiantes a identificar el tipo de trabajo que les gustaría desempeñar tras su graduación.

- **Trabajo en un proyecto**, también con un mínimo aconsejado de 3 meses. Este proyecto es esencial para desarrollar las habilidades antes mencionadas. Se recomienda encarecidamente que se realicen proyectos en equipos, aun reconociendo que su evaluación es más difícil que la de los proyectos individuales.

Finalmente, el documento [CareerSpace'01] ofrece algunas guías y recomendaciones a las universidades para la implantación de las propuestas del consorcio. Entre estas recomendaciones destacamos la necesidad de implementar un proceso de control de calidad de los planes de estudios, cuyos resultados se aplicarían en la mejora de los planes. Se sugiere que este proceso podría estar basado en recabar información de los egresados y de sus empresas, en un periodo entre uno y tres años tras su salida de la universidad.

En general, consideramos que las recomendaciones de Career Space son interesantes y deben ser tenidas muy en cuenta. En algunos puntos, se aprecia en las propuestas el claro carácter empresarial de la composición del consorcio. Creemos que la FIUM está en una buena situación para cumplir los requerimientos que se plantean. La revisión y actualización de sus planes de estudios, la inclusión en los planes nuevos de la asignatura "Ingeniería de sistemas de información" –basada en proyectos en equipo, a propuesta de las empresas murcianas en TIC–, la introducción de asignaturas como "Informática y legislación" o "Filosofía, tecnología digital y sociedad", y los cursos de promoción educativa impulsados desde el Decanato de nuestra Facultad, son algunos pasos decididos en este sentido. Por lo demás, el grado de logro de los objetivos depende lógicamente del adecuado planteamiento de los programas de cada asignatura concreta de los planes de estudio.

Capítulo 3

Finalidades y objetivos

"No aprendemos para la escuela, sino para la vida."
Lucio Anneo Séneca

En cualquier orden de la vida, analizar, decidir y hacer explícitas las finalidades y objetivos de lo que se pretende es un paso previo y necesario al diseño de un plan de actuación, cuyo propósito no será otro sino alcanzar tales objetivos. En nuestro caso, consideramos importante plantear tanto los **objetivos formativos** propios de la materia de estudio –definidos como *"el comportamiento esperado en el alumno como consecuencia de una determinada actividad docente, y que es susceptible de ser observado y evaluado"* [Rodríguez'00]–, como aquellas **metas y finalidades** más amplias que se orientan al desarrollo de los individuos como personas –en el sentido apuntado por el político francés Guizot, *"... el fin de la educación es enseñar al hombre a educarse a sí mismo cuando los demás hayan terminado de educarlo..."*–. Mientras que los primeros desempeñarán un papel fundamental en la selección de contenidos, en las decisiones sobre la metodología docente y en el diseño de los procesos de evaluación, los segundos deben guiar la actuación del profesor universitario como profesional responsable y comprometido.

Por lo tanto, siendo conscientes de la importancia de un buen planteamiento de los objetivos formativos, estudiaremos en esta sección los que son propios de la materia "Ampliación de Algoritmos y Estructura de Datos". Pero antes haremos un repaso más amplio de las metas y finalidades de la educación universitaria, como principios universales y subyacentes al propio proceso docente.

3.1. Finalidades de la educación universitaria

Funciones y fines de la institución universitaria

Como ya vimos en el análisis del contexto institucional, en la sección 2.1, las dos funciones básicas de la Universidad son la investigación y la docencia. Pero la misión de la Universidad en nuestros días no se limita a estas dos facetas –entendidas desde un simple punto de vista económico–, sino que orienta sus esfuerzos a la alta meta de ser *"un instrumento eficaz de transformación social, al servicio de la libertad, la igualdad y el progreso social para hacer posible una realización más plena de la dignidad humana"*, como manifiesta abiertamente la LRU, [LRU'83]. De esta forma, tanto la LRU como la LOU y los Estatutos de la UM incluyen entre las funciones propias de la Universidad: la

creación del conocimiento, la extensión de la cultura universitaria en la sociedad, el apoyo al desarrollo cultural y económico, con atención singular a la propia Comunidad Autónoma, y la participación en el estudio y debate de los asuntos de interés social.

En similares términos expresa la UNESCO en la "Declaración Mundial sobre la Educación Superior en el Siglo XXI", [UNESCO'98], las misiones de la Universidad:

> *"Las misiones tradicionales de los establecimientos de enseñanza superior (educar, realizar investigaciones y ofrecer servicios a la comunidad) siguen estando vigentes, pero nosotros afirmamos que la enseñanza superior de hoy tiene por* ***misión principal educar a ciudadanos responsables, constituyendo un espacio abierto de aprendizaje de alto nivel y de aprendizaje a lo largo de toda la vida****. Además, la educación ha adquirido una función sin precedente en la sociedad como elemento fundamental del desarrollo cultural, social, económico y político, y como pilar del refuerzo de las capacidades endógenas, de la democracia y la paz."*

La Declaración define la educación superior como un servicio público, abierto por igual a todos los ciudadanos en función de sus méritos y no de su raza, sexo, religión o situación económica, de conformidad con la "Declaración Universal de los Derechos Humanos". Añade, además, tres nuevas funciones que sitúan a la Universidad como garante de la paz y la democracia:

- **Función crítica,** mediante la búsqueda de la verdad y de la justicia, sometiendo todas sus actividades a la exigencia del rigor ético y científico.
- **Función prospectiva,** mediante un análisis constante de las nuevas tendencias sociales, económicas, culturales y políticas.
- **Función preventiva,** actuando como vigías capaces de prever, adelantarse y alertar.

Para que las Universidades puedan realizar sus funciones alcanzando los fines propuestos, es necesario garantizar las plenas libertades académicas y preservar su autonomía, siendo al mismo tiempo responsables para con la sociedad y rindiéndole cuentas de su actividad. Y, como servicio público que son, necesitan el apoyo de las instituciones políticas para asegurar que sus misiones educativas y sociales se llevan a cabo de manera equilibrada.

Finalidades de la docencia universitaria

En lo que se refiere más particularmente a la acción docente, es bien reconocido que la transmisión de conocimientos no puede ser el único fin de la labor del profesor universitario. La formación integral de las personas transforma la tarea del docente de *enseñar*, sin más, a *educar* en el sentido más amplio. De esta forma, de acuerdo con Wilder [Wilder'71], los propósitos de la docencia en la educación superior se pueden concretar en los siguientes puntos:

- Conseguir que los alumnos alcancen una **formación comprensiva e integrada** de las materias que estudian.
- Estimular en los estudiantes el desarrollo de una **actitud crítica constructiva**, que les conduzca al cuestionamiento permanente de cualquier conocimiento o forma de saber.
- Fomentar en los estudiantes **hábitos de indagación, observación, reflexión y auto-evaluación**, que les permita aprender de los errores, profundizar en el conocimiento, aprender a aprender y aprender a pensar.

- Desarrollar en los estudiantes la **capacidad de planificar, desarrollar y evaluar** estrategias de acción, conociendo el contexto social y profesional en el que tendrán que desarrollar su actividad.
- Fomentar el **trabajo en equipo** a través de actitudes y técnicas de coordinación adecuadas.
- Fomentar en los estudiantes la actitud de **aprender a emprender**, de tomar iniciativas creativas, de proponer proyectos de diferente índole, de implicarse en la elaboración y construcción personal.
- Fomentar el **aprendizaje no dirigido**, al que no suelen estar acostumbrados antes de llegar a la Universidad, a través de las vías de la documentación y la investigación.

Aunque enunciadas de forma abstracta y general, podemos extraer de estas metas algunas conclusiones prácticas interesantes, que deberían ser consideradas en la preparación y desarrollo de la actividad docente, y de forma muy particular en la asignatura objeto de estudio. Por ejemplo, de la meta de fomentar el trabajo en equipo se desprende la adecuación de plantear prácticas en grupos, frente a las prácticas individuales[1]. También se intuye la necesidad de dejar espacios abiertos para el aprendizaje de los alumnos por sí mismos, lo cual se puede traducir en plantear preguntas o problemas sin resolver en clase.

En general, todos estos propósitos requieren como elemento esencial e indispensable el **papel activo de los alumnos**; a lo cual el profesor sólo puede aspirar a impulsar mediante la motivación adecuada. A falta de esa actitud activa y participativa, la labor del alumno se convierte en una tarea rutinaria de "estudiar para aprobar". En este sentido, el mismo autor propone en [Wilder'71] un decálogo de consejos prácticos, que deberían aplicar los profesores en su actividad cotidiana:

1. No introducir un nuevo concepto sin motivarlo previamente.
2. Ser honrado, lo que exige no pretender tener conocimientos de algo que no se sepa realmente.
3. No preparar en exceso, dado que el universitario tiene un espíritu combativo y quiere tener derecho a alternar en la discusión de un tema.
4. No sentir temor por seguir un tema aparentemente polémico si es traído a discusión por los estudiantes.
5. Cuidar de no caer en la memorización sin participación activa o crítica.
6. Conservar el entusiasmo por la materia que se explica.
7. Aceptar preguntas de índole general.
8. Aclarar el papel que desempeñan las definiciones.
9. Lograr un grado de conocimiento del alumno tan amplio como sea posible.
10. Considerar la actividad de la enseñanza como una labor mancomunada, en la que el profesor asume el papel de tutor respecto de la iniciativa de los alumnos.

[1] En muchas ocasiones, los alumnos se muestran reacios a esta forma de trabajo –a la que no están habituados–, y no es extraño encontrar en una práctica planteada para grupos de dos, un porcentaje de alumnos que la hacen individualmente.

En una línea parecida Paul Ramsden [Ramsden'92] añade unos cuantos puntos, también en tono de consejos prácticos y concretos, sobre cómo debería ser y actuar un buen profesor universitario:

1. Poseer un amplio repertorio de habilidades docentes específicas.
2. No olvidar que su meta es el aprendizaje de los estudiantes.
3. Escuchar y aprender de los alumnos.
4. Evaluar constantemente su actuación docente.
5. Pensar que enseñar es hacer posible el aprendizaje.
6. Enseñar con entusiasmo.
7. Mostrar preocupación y respeto por los alumnos.
8. Tener facilidad para hacerse entender por los estudiantes.
9. Hacer de los estudiantes aprendices autónomos.
10. Usar métodos que exijan al estudiante aprender de forma activa y cooperativa.
11. Proporcionar realimentación de alta calidad a los estudiantes sobre sus trabajos.
12. Enseñar los conceptos clave de su materia y evitar la sobrecarga de trabajo.

En definitiva, la docencia universitaria requiere que el profesor *se involucre* de lleno en la enseñanza. Motivar y estar motivado son dos aspectos clave para lograr una enseñanza de calidad. Y, sin duda, las mejores armas de motivación de las que dispone el profesor son la capacidad de transmitir entusiasmo por la materia y el propio ejemplo personal que ofrece en el desempeño de su profesión. Por lo tanto, en lo relativo a su función docente, un profesor universitario de informática no es un mero informático que da clases, sino que es ante todo y sobre todo un profesional de la enseñanza.

3.2. Proceso y criterios de selección de objetivos

Una vez analizadas las finalidades de la docencia universitaria en general, vamos a centrarnos en los objetivos educativos específicos de la materia "Ampliación de Algoritmos y Estructura de Datos". Pero antes de tomar decisiones al respecto, consideramos necesario definir unos criterios y un proceso de selección que garanticen que la elección es realizada de manera justificada y fundamentada. Dentro de esta sección veremos en primer lugar la necesidad de establecer unos objetivos adecuados; después realizamos una clasificación de los objetivos, en función del tipo de propósitos que persiguen; y, finalmente, repasamos toda la información que pueda ser de interés para la selección de los objetivos específicos, como directrices propias, recomendaciones internacionales y los programas actuales.

3.2.1. La importancia de los objetivos educativos

El establecimiento de unos objetivos claros y consensuados para una asignatura tiene repercusiones positivas tanto sobre el alumno como sobre el profesor. Al alumno le permite tener una perspectiva general de lo que se pretende desde el principio, constituyendo una constante orientación

para el estudio. Para el profesor, los objetivos son una guía en la planificación y programación de las actividades docentes y en la elección del sistema de evaluación más adecuado.

La lista de objetivos de una asignatura es un documento abierto para la discusión, el refinamiento y la actualización continua; un documento que se podría entender como el **contrato inicial** que se establece en la relación entre profesor y alumnos. Es más, algunos expertos proponen incluso la participación de los propios alumnos en la planificación de los objetivos. De acuerdo con [García-Valcárcel'01], *"la planificación de objetivos y actividades con los alumnos aumentará su motivación, ya que el compromiso por llevar a cabo algo que uno mismo ha planificado siempre es mayor que en caso de resultar impuesto por el profesor"*. Aunque en principio pueda parecer algo inviable en muchas de nuestras asignaturas, la propuesta de actividades de este tipo puede tener resultados positivos, como despertar el interés de los alumnos y poder conocer sus intereses y necesidades personales.

Desafortunadamente, en las enseñanzas universitarias la atención que reciben las tareas de planificación de las metas y objetivos es escasa o nula [deMiguel'99]. Tanto a nivel de planes de estudios como a nivel de cada asignatura concreta, la planificación se elabora directamente sobre los contenidos. Según las conclusiones extraídas por Mario de Miguel sobre un estudio realizado dentro del Plan Nacional de Evaluación de la Calidad de las Universidades, la mayoría de las titulaciones en las Universidades *"se implantan sin que previamente se haya establecido por consenso claro y explícito el perfil del titulado que se desea formar, limitándose, en todo caso, a transcribir el texto que aparece en las directrices generales correspondientes"*. Tampoco los programas de las asignaturas suelen incluir un listado de objetivos y normalmente *"la elaboración del programa se limita a una especificación detallada de los contenidos de la asignatura y una relación de las referencias bibliográficas pertinentes"*.

3.2.2. Clasificación de los objetivos

Como vimos en la introducción, los objetivos educativos se definen como los comportamientos, en sentido amplio, que se espera de los alumnos como resultado de la acción docente. Es posible encontrar diversas taxonomías en la literatura especializada en el tema, [Bloom'56], [Bloom'64], [deMiguel'99], en función de los diversos enfoques empleados. Un compendio de todas estas aportaciones se puede considerar la clasificación propuesta en [Marcelo'01], que clasifica los objetivos en conocimientos, habilidades y actitudes.

Los **conocimientos** esperados, también denominados objetivos cognitivos, se refieren lo que deben saber los alumnos, estando ligados directamente con los contenidos de la asignatura. Las **habilidades**, u objetivos psicomotores, tienen que ver con el *saber hacer*, es decir, la capacidad de transferir los conocimientos adquiridos a la resolución de problemas. Las **actitudes**, u objetivos afectivos, están asociados con el *saber ser*, entendiendo que la formación universitaria no es una mera instrucción, sino que busca extender su margen de influencia a la educación de la persona y del profesional. Siguiendo la clasificación de [Marcelo'01], podemos derivar los siguientes objetivos concretos dentro de cada una de estas categorías:

- **Objetivos cognitivos o conocimientos.**
  - **Conocimiento** de realidades, de principios, de modos y medios para la adquisición y tratamiento de los conocimientos.
  - **Análisis:** capacidad de descomponer un conjunto de información en sus partes.
  - **Comprensión:** explicar o razonar una información.
  - **Síntesis:** capacidad para componer con elementos y partes, un todo o conjunto de información.

  - **Aplicación:** capacidad para trasladar los planteamientos teóricos a situaciones concretas.
  - **Evaluación:** juicios de valor.

- **Objetivos psicomotores o habilidades.**
  - **Destrezas académicas:** leer, ver, oír, tomar notas, hacer gráficos, interpretar documentos gráficos, construir diagramas, tabular, diseñar.
  - **Destrezas de investigación:** observar, plantear hipótesis, analizar, valorar, aplicar, buscar documentación, utilizar instrumentos de investigación, manipular materiales.
  - **Destrezas sociales:** cooperar, saber discutir, defender las ideas propias, argumentar, trabajar en equipo, dirigir discusiones de grupo, liderar grupos, resolver conflictos.

- **Objetivos afectivos o actitudes.**
  - **Percibir:** tomar conciencia, sensibilizarse.
  - **Responder:** interesarse, apreciar, estar motivado.
  - **Valorar:** tomar en cuenta, aceptación, adaptación.
  - **Implicarse:** tomar parte, esforzarse.
  - **Curiosidad y compromiso,** por continuar aprendiendo.
  - **Participación** y compromiso social.
  - **Actitud de iniciativa:** aprender a emprender.

3.2.3. Consideraciones sobre los objetivos propios de la materia

En este apartado exponemos las consideraciones más importantes que se deben tener en cuenta en la selección de los objetivos de la asignatura "Ampliación de Algoritmos y Estructura de Datos", sintetizando en parte la información más relevante presentada en la fundamentación disciplinar y en el análisis del contexto. Empezaremos por un repaso de las indicaciones contenidas en las directrices generales propias y en los planes de estudios en los que se enmarca la asignatura, ya estudiados en la sección 2.2 (Contexto curricular). Después revisamos las recomendaciones curriculares propias contenidas en el CC2001, así como los programas actuales de las asignaturas.

Directrices y planes de estudios

Como vimos en el punto 2.2.1, las directrices generales propias de los planes de estudios de informática –tanto de la Ingeniería superior como de las Ingenierías Técnicas– señalan dos contenidos troncales con los que, en principio, puede identificarse la asignatura AAED: "Estructuras de datos y de la información" y "Metodología y tecnología de la programación". Los descriptores asociados a estas materias son los siguientes:

- **Estructuras de datos y de la información.** Tipos Abstractos de Datos. Estructura de Datos y algoritmos de manipulación. Estructura de Información: ficheros, bases de datos.

- **Metodología y tecnología de la programación.** Diseño de Algoritmos. Análisis de Algoritmos. Lenguajes de Programación. Diseño de Programas: Descomposición modular y documentación. Técnicas de Verificación y prueba de programas

Los planes de estudios reparten los contenidos de estas materias troncales entre las distintas asignaturas del mismo, garantizando la carga que se establece en las directrices. En los planes actualmente vigentes, de 2002, la asignatura AAED pasa a denominarse "Algoritmos y Estructuras de Datos", como ya analizamos en el punto 2.2.3. Además, la estructura y descripción de la asignatura es la misma para las tres titulaciones. La información que aparece en los correspondientes BOE es la siguiente:

- **Tipo:** Troncal.
- **Curso:** 2º.
- **Duración:** Anual.
- **Créditos:** 12 (6 teóricos y 6 prácticos).
- **Descripción:** Tipos abstractos de datos. Estructura de datos y algoritmos de manipulación. Ampliación de análisis y diseño de algoritmos.

Necesariamente, el desarrollo de estos contenidos deberá ser el primero de los objetivos propios de la asignatura AED. No obstante, la generalidad y ambigüedad de las descripciones es lo suficientemente amplia como para permitir un gran margen de actuación.

Recomendaciones curriculares del informe CC2001

Ya hemos comentado en el apartado 2.2.2 la interesante guía que suponen las recomendaciones curriculares contenidas en el informe **Computing Curricula 2001** –fruto de una labor conjunta entre las asociaciones ACM e IEEE–, no sólo en el diseño de planes de estudios sino también en la planificación de objetivos y contenidos de las asignaturas del mismo. Gracias al elevado nivel de detalle de este documento, es posible tener un apoyo sólido, contrastado y consensuado internacionalmente a la hora de diseñar los aspectos más concretos de las asignaturas.

En este punto haremos un repaso de los objetivos educativos correspondientes a las unidades del CC2001 que –teniendo en cuenta los descriptores del plan de estudios– consideramos que están relacionadas directamente con la asignatura "Algoritmos y Estructuras de Datos". En particular, podemos señalar las siguientes unidades: DS5 - Grafos y árboles; PF2 - Algoritmos y resolución de problemas; PF3 - Estructuras de datos fundamentales; PF4 - Recursividad; AL1 - Análisis de algoritmos básico; AL2 - Estrategias algorítmicas; AL3 - Algoritmos fundamentales en computación; PL5 - Mecanismos de abstracción; SE10 - Métodos formales. Los objetivos contenidos en estas unidades son los siguientes[2]:

DS5. Grafos y árboles [núcleo]

1. Ilustrar mediante ejemplos la terminología básica de la teoría de grafos, y algunas de las propiedades y casos especiales cada uno.
2. Mostrar diferentes métodos de recorrido de árboles y grafos.
3. Modelar problemas de computación usando grafos y árboles.
4. Relacionar los grafos y árboles con estructuras de datos, algoritmos y conteo.

[2]Mostramos sólo los objetivos educativos. Los contenidos de cada unidad –que también están definidos en el CC2001– serán analizados en el siguiente Capítulo.

PF2 - Algoritmos y resolución de problemas [núcleo]

1. Discutir la importancia de los algoritmos en el proceso de resolución de problemas.
2. Identificar las propiedades necesarias de un buen algoritmo.
3. Crear algoritmos para resolver problemas sencillos.
4. Usar pseudocódigo o un lenguaje de programación para implementar, probar y depurar algoritmos para resolver problemas sencillos.
5. Describir las estrategias que son más útiles en depuración.

PF3 - Estructuras de datos fundamentales [núcleo]

1. Discutir la representación y uso de tipos de datos primitivos y estructuras de datos pre-construidas.
2. Describir cómo las estructuras de datos estudiadas son reservadas y usadas en memoria.
3. Describir aplicaciones comunes para cada una de las estructuras de datos estudiadas.
4. Implementar las estructuras de datos definidas por el usuario en un lenguaje de alto nivel.
5. Comparar métodos alternativos de implementación de las estructuras de datos con respecto al rendimiento.
6. Escribir programas que usen cada una de las siguientes estructuras de datos: arrays, registros, cadenas, listas enlazadas, pilas, colas y tablas de dispersión.
7. Comparar y contrastar los costes y beneficios de las implementaciones estáticas y dinámicas de las estructuras de datos.
8. Elegir las estructuras de datos apropiadas para modelar un problema dado.

PF4 - Recursividad [núcleo]

1. Describir el concepto de recursividad y dar ejemplos de uso.
2. Identificar el caso base y el caso general de un problema definido recursivamente.
3. Comparar soluciones iterativas y recursivas para problemas elementales como el cálculo del factorial.
4. Describir la estrategia divide y vencerás.
5. Implementar, probar y depurar funciones y procedimientos recursivos sencillos.
6. Describir cómo la recursividad puede ser implementada usando una pila.
7. Discutir problemas para los cuales el backtracking puede ser una buena solución.
8. Determinar cuándo una solución recursiva es apropiada para un problema.

AL1 - Análisis de algoritmos básico [núcleo]

1. Explicar el uso de las notaciones O-grande, Omega y Theta para describir la cantidad de trabajo realizado por un algoritmo.

2. Usar las notaciones O-grande, Omega y Theta para dar las cotas superior, inferior y orden exacto del tiempo y espacio consumidos por un algoritmo.

3. Determinar la complejidad de algoritmos sencillos, en cuanto a tiempo y uso de memoria.

4. Deducir relaciones recurrentes para describir la complejidad de algoritmos definidos de manera recursiva.

5. Resolver relaciones de recurrencia elementales.

AL2 - Estrategias algorítmicas [núcleo]

1. Describir los inconvenientes de los algoritmos de fuerza bruta.

2. Para cada uno de entre varios tipos de algoritmos (fuerza bruta, voraces, divide y vencerás, backtracking, ramificación y poda, y heurístico), identificar un ejemplo del comportamiento cotidiano de una persona que ejemplifique el concepto básico.

3. Implementar un algoritmo voraz para resolver un problema adecuado.

4. Implementar un algoritmo de divide y vencerás para resolver un problema adecuado.

5. Usar backtracking para resolver problemas como los de navegación en un puzzle.

6. Describir varios métodos heurísticos de resolución de problemas.

7. Usar búsqueda de patrones para analizar subcadenas.

8. Usar algoritmos de aproximación numérica para resolver problemas matemáticos, tales como encontrar las raíces de un polinomio.

AL3 - Algoritmos fundamentales en computación [núcleo]

1. Implementar los algoritmos de ordenación más comunes de orden cuadrático y $O(N \log N)$.

2. Diseñar e implementar una función de dispersión adecuada para una aplicación.

3. Diseñar e implementar un algoritmo de resolución de colisiones para una tabla de dispersión.

4. Discutir la eficiencia computacional de los principales algoritmos de ordenación, búsqueda y dispersión.

5. Discutir otros factores, aparte de la eficiencia computacional, que influyen en la elección de un algoritmo, como el tiempo de programación, facilidad de mantenimiento, y el uso de patrones específicos de entrada para la aplicación dada.

6. Resolver problemas usando los algoritmos de grafos fundamentales, incluyendo la búsqueda primero en profundidad y en anchura, los caminos mínimos desde un nodo origen o entre todos los nodos, el cierre transitivo, la ordenación topológica, y al menos un algoritmo de árboles de expansión de coste mínimo.

7. Demostrar las siguientes capacidades: evaluar algoritmos, seleccionar entre una variedad de posibles opciones, dar una justificación de tal selección, e implementar el algoritmo en un contexto de programación.

PL5 - Mecanismos de abstracción [núcleo]

1. Explicar cómo los mecanismos de abstracción permiten la creación de código reutilizable.
2. Demostrar la diferencia entre paso de parámetros por valor y paso por referencia.
3. Defender la importancia de las abstracciones, especialmente con respecto a la programación en proyectos grandes.
4. Describir cómo el ordenador utiliza registros de activación para manejar los módulos de un programa y sus datos.

SE10 - Métodos formales [opcional]

1. Aplicar técnicas de verificación formal a fragmentos de software de baja complejidad.
2. Discutir el papel de las técnicas de verificación formal en el contexto de validación y prueba de software.
3. Explicar los beneficios e inconvenientes potenciales de usar lenguajes de especificación formal.
4. Crear y evaluar pre- y postcondiciones sobre una variedad de situaciones desde algunas sencillas hasta otras más complejas.
5. Usar un lenguaje de especificaciones formales común, formular la especificación de un sistema software sencillo y demostrar los beneficios desde la perspectiva de la calidad.

Hay que tener en cuenta que no todos los objetivos antes expuestos son necesariamente propios y exclusivos de la asignatura objeto de estudio, debido a que muchas de las unidades del CC2001 incluyen tópicos tratados en otras asignaturas del plan de estudios. Por ejemplo, una buena parte de los objetivos son compartidos con la asignatura "Metodología y Tecnología de la Programación", de primer curso. En la siguiente sección seleccionaremos los objetivos de esta lista que forman parte de la asignatura AED y que, recordemos, coinciden con los de AAED.

Por otro lado, la última de las unidades anteriores, SE10, es la única que hace referencia a contenidos opcionales de las recomendaciones del CC2001. Consideramos que su inclusión es necesaria como una forma de incidir en el descriptor del plan de estudios que hace referencia a los Tipos Abstractos de Datos.

Antecedentes previos

Es evidente que la planificación de una asignatura debe tener en cuenta la dimensión temporal en la que se desarrollará: qué objetivos se han perseguido en cursos anteriores, cuál es la situación actual, y cuáles de estos objetivos deben ser revisados, sustituidos o añadidos. Por este motivo, examinaremos aquí los objetivos propuestos para las asignaturas predecesoras de AED en la FIUM. Los antecedentes más próximos que encontramos son los proyectos docentes [Giménez'90] y [Marín'90], así como los planes de estudios durante el curso anterior.

Los proyectos docentes mencionados corresponden ambos a la asignatura "Algoritmos y Estructuras de Datos" y enumeran los siguientes objetivos[3]:

[3]Los objetivos están formulados desde "el punto de vista del alumno", es decir, se refieren a lo que los alumnos deberían conocer o saber aplicar. En el caso de los objetivos del CC2001 no está claro; en su mayor parte se pueden entender como enunciados desde "el punto de vista del profesor", esto es, qué debe hacer el profesor.

1. Aprender el concepto de Tipo Abstracto de Datos, de su especificación, de Tipo Opaco y de Programación Modular, y comprender la utilidad de su utilización.

2. Ser capaz de identificar dentro de un problema su estructura modular y los tipos de datos necesarios para su resolución.

3. Aprender algunas técnicas para calcular el costo de tiempo y de espacio de diversos algoritmos para poder elegir algoritmos eficaces.

4. Aprender las técnicas básicas de diseño de algoritmos entendiendo que estas técnicas ante un problema real van a servir normalmente sólo como referencia y no nos van a resolver el problema.

5. Ser capaces de identificar problemas que correspondan a los tipos estudiados y de aplicar los esquemas a los casos concretos.

6. Aprender a familiarizarse con técnicas de diseño modular, trabajo en equipo, documentación y prueba.

7. Aprender las estructuras básicas de datos y ser capaces de determinar qué estructura puede corresponder a un problema dado.

8. Aprender a manejar estas estructuras básicas y ser capaces de crear otras más complejas teniendo estas como referencia.

9. Aprender los conceptos básicos de la programación concurrente y sus técnicas características.

10. Aprender un nuevo lenguaje de programación.

Los objetivos son bastante generales y abstractos, sin referirse a técnicas algorítmicas o de representación de datos concretas. Destaca por su enunciado el punto número 4, en el que se argumenta que las técnicas de diseño sólo serán una guía, pero no un proceso automático de resolución de problemas. En este sentido, está claro que conocer las técnicas no tiene utilidad si no se tiene la habilidad de saber aplicarlas y el juicio para valorar su adecuación y sus limitaciones.

Hay que tomar en consideración estos objetivos educativos con la debida cautela, ya que su elaboración se remonta al año 1990. Además de los avances de la disciplina en este periodo, los planes de estudios han evolucionado, siendo eliminados los descriptores que hacían referencia a la programación concurrente –que ahora se imparte como una asignatura troncal independiente–. En el proyecto [Giménez'97], correspondiente a la asignatura "Algorítmica", se seleccionan algunos de estos objetivos, suprimiendo los que son propios de la parte de estructuras de datos. Se pone énfasis en las habilidades para seleccionar la técnica de diseño que resulte más adecuada para cada problema, y en la capacidad de aplicar las técnicas sobre problemas reales.

Los programas de las asignaturas "Estructura de Datos" y "Algorítmica", ambas de los planes de 1996 en las Ingenierías Técnicas, presentan objetivos similares a los ya comentados. En el segundo de ellos se plantea como objetivo general *"que los alumnos aprendan las técnicas básicas de análisis y diseño de algoritmos, y que vean el proceso de realización de programas como un proceso riguroso"*. Por su parte, los programas de "Ampliación de Algoritmos y Estructura de Datos" –del plan de 1996 en la Ingeniería superior– y de "Algoritmos y Estructuras de Datos" –de los planes de 2002 en las tres titulaciones– no especifican los objetivos formativos, y se limitan a desarrollar los contenidos.

3.3. Objetivos específicos de la materia

Una vez analizados y sintetizados todos los aspectos que deben ser tenidos en cuenta, comenzamos la toma de decisiones que supone la selección de objetivos para la materia objeto de concurso. Esta especificación de objetivos es previa, y por lo tanto independiente, a la determinación de contenidos. El proceso de selección irá de lo más general a lo más abstracto, desde el objetivo general de la asignatura hasta un nivel de detalle similar al de las recomendaciones del CC2001. Dentro de estos objetivos diferenciamos los globales y los puntuales. Los primeros se refieren a la asignatura en su conjunto, mientras que los segundos pueden ser asociados con temas particulares de la asignatura.

3.3.1. Objetivos globales de la asignatura

Podemos formular el **objetivo básico de la materia "Ampliación de Algoritmos y Estructura de Datos"**, desde el punto de vista del alumno, de la siguiente forma:

> Ser capaz de analizar, comprender y resolver una amplia variedad de problemas computacionales, diseñando e implementando soluciones eficientes y de calidad, como resultado de la aplicación de un proceso metódico.

Por lo tanto, la meta final de la asignatura es independiente de cualquier técnica o paradigma particular, que podrían verse afectados por las tendencias futuras o por la dificultad de incluirlas en el temario. Las técnicas pueden ser sustituidas por otras, y pueden extenderse más o menos según el año; pero en cualquier caso los alumnos deberán estar capacitados para abordar con soltura problemas de programación, aplicando un proceso que va desde un enunciado en lenguaje natural hasta la implementación y verificación de un programa. Este objetivo se puede articular en un conjunto de grandes objetivos globales:

- **Conocer y comprender los fundamentos de la disciplina de los algoritmos y las estructuras de datos.** El alumno debe alcanzar una comprensión profunda de la materia de "Algoritmos y Estructuras de Datos", conociendo los propósitos que se persiguen, valorando su importancia dentro de la informática y manejando los hábitos de trabajo propios de la disciplina. Tiene que quedar claro que la asignatura va mucho más allá que simplemente saber programar. En cierto sentido, este objetivo hace hincapié en los conceptos, bases y principios que fueron introducidos en las asignaturas previas de programación. Podemos mencionar los siguientes objetivos concretos, que surgen a raíz de este:
  - Conocer y comprender los fines, propósitos y métodos de la disciplina, situándolos dentro de una formación más amplia en ingeniería en informática.
  - Diferenciar y saber definir los conceptos de problema, programa, proceso, algoritmo, esquema algorítmico, tipo abstracto de datos, tipo de datos y estructura de datos.
  - Distinguir y relacionar el papel que desempeñan los algoritmos y las estructuras de datos en la resolución de problemas de programación.
- **Conocer y saber aplicar una amplia variedad de técnicas de diseño de algoritmos y técnicas de representación de datos.** En su conjunto, el alumno debe ver todas las técnicas estudiadas, tanto de estructuración de datos como de diseño de algoritmos, como una batería de herramientas a su disposición para afrontar con garantías la resolución de una gran cantidad de problemas. Además, estas técnicas no deben aparecer como componentes aislados, sino que el

estudiante debe crearse su propio "mapa mental de técnicas" en el que se ubiquen las relaciones, afinidades y contrastes entre las diferentes técnicas, así como las ventajas y limitaciones de cada una. Podemos establecer, a su vez, un conjunto de subobjetivos más específicos:

- Valorar las ventajas, inconvenientes, limitaciones y ámbito de aplicación de cada una de las técnicas estudiadas.
- Tener la capacidad de decidir por uno mismo qué técnica o conjunto de técnicas resultan más adecuadas para abordar cierto problema dado, realizando la elección de forma bien fundamentada.
- Saber aplicar las técnicas sobre problemas de la vida real, partiendo de un enunciado en lenguaje natural.
- Encontrar y analizar las diferentes formas en las que se puede aplicar una misma técnica sobre un problema dado, seleccionando la más adecuada de manera justificada.
- Analizar y reconocer las técnicas y principios subyacentes que fueron aplicados en la creación de muchos de los algoritmos o sistemas conocidos por los alumnos.

- **Conocer y utilizar las técnicas de análisis de eficiencia, tomando conciencia de la importancia del factor eficiencia en la resolución de problemas.** Los alumnos deben tener la capacidad de analizar con soltura tanto el tiempo de ejecución como la memoria de los algoritmos y de las estructuras de datos estudiados. Pero el análisis de eficiencia no puede estar completamente desligado del diseño; un buen ingeniero debe acostumbrarse a tener en mente el factor eficiencia siempre que diseña una solución a un problema. De esta manera, al resolver un problema se podrán descartar posibilidades que conducen a resultados correctos pero a costa de un consumo de recursos desorbitado. Señalamos los siguientes objetivos concretos:

  - Conocer, comprender y valorar los objetivos que persigue el área del análisis de algoritmos, siendo capaz de reconocer sus distintos usos y de diseñar un plan de análisis para un objeto de interés dado.
  - Ser consciente de las limitaciones que el factor eficiencia impone en la resolución de un problema, haciendo hincapié en la importancia del orden de complejidad frente a las optimizaciones de bajo nivel.
  - Saber definir, diferenciar y relacionar los conceptos de tiempo de ejecución, uso de memoria, eficiencia computacional, orden de complejidad y notaciones asintóticas.
  - Conocer y saber aplicar las técnicas que permiten la estimación del consumo de recursos de los algoritmos y las estructuras de datos estudiadas, con mención a los casos recursivos e iterativos.
  - Tener la habilidad de expresar el orden de complejidad de los algoritmos usando las notaciones asintóticas más adecuadas para cada situación.

- **Saber afrontar la resolución de problemas nuevos, utilizando las técnicas estudiadas como herramientas flexibles y adaptables.** En función de los contenidos que puedan ser desarrollados cada año, los alumnos aprenderán un conjunto de técnicas que ayudan en la resolución de ciertos tipos de problemas. Pero es necesario dotar a los alumnos de las capacidades que les permitan ir más allá de lo estrictamente estudiado. En este sentido, el objetivo de las técnicas es enseñar a pensar, más que memorizar las peculiaridades de las mismas. La conclusión práctica es la necesidad de extenderse más en anchura que en profundidad; cuantos más problemas y

cuanto más variados, mejor. A raíz de este objetivo podemos distinguir las siguientes habilidades concretas:

- Desarrollar la creatividad en la aplicación de las técnicas, y en general en la resolución de problemas, aunque siempre sobre una sólida base conceptual y metódica.
- Mejorar la comprensión de las peculiaridades y restricciones de problemas nuevos, partiendo de un enunciado en lenguaje natural, así como de los posibles acercamientos a su resolución.
- Saber combinar diferentes técnicas –estudiadas en la misma o en otras asignaturas, o en cualquier otro sitio– para abordar un problema nuevo.
- Tener capacidad de generalizar las técnicas estudiadas, abstrayendo y sabiendo diferenciar la esencia y la filosofía subyacente a las mismas de los detalles y características superfluos.

- **Concienciarse de la importancia de seguir un proceso metódico y de lograr los factores de calidad del software.** Un ingeniero en informática debe entender la resolución de problemas de programación como un proceso metódico e ingenieril, en el que la codificación de un programa va precedida de los necesarios pasos de análisis y diseño. Los alumnos deben conocer y aplicar este proceso, siendo conscientes de que cuanto más tiempo se dedique a las etapas iniciales mejores serán los resultados obtenidos. Hacer explícito el proceso, por ejemplo en el desarrollo de las prácticas, es un buen medio de mejorar esta visión. Destacamos en este punto los siguientes objetivos:

  - Conocer, comprender y saber aplicar los pasos que componen el proceso de resolución de problemas, siendo conscientes del interés y la importancia de seguir un proceso metódico.
  - Concienciarse de la importancia de manejar abstracciones, conocer los tipos y mecanismos de abstracción que aparecen, y saber trabajar en un problema sobre diferentes niveles de abstracción.
  - Conocer y utilizar adecuadamente los mecanismos de los lenguajes de programación que dan soporte al uso de abstracciones, como las funciones, procedimientos, módulos, paquetes y clases.
  - Tener habilidad en el modelado conceptual de los problemas, en lo relativo a abstracción de las características del problema, determinación de su descomposición y estructura modular, y especificación de los distintos componentes.
  - Concienciarse de la importancia de las especificaciones en el ámbito de la programación, conociendo y sabiendo utilizar técnicas formales e informales de especificación.

- **Desarrollar las habilidades y actitudes sociales que permiten la práctica de la actividad profesional en entornos de trabajo en equipo.** Las actividades de clase y las prácticas propuestas deben fomentar estas actitudes, necesarias para todo profesional informático y sin las cuales los conocimientos más técnicos pierden su efectividad. Realmente esta finalidad es compartida por todas las asignaturas del plan de estudios, ocupándose cada una de su parcela de actuación y estudio. El alumno debe cambiar la mentalidad que trae de bachillerato por una mentalidad profesional. Como parte de este objetivo, podemos mencionar los siguientes propósitos:

  - Fomentar los hábitos de investigación y aprendizaje no guiado. Entre estos hábitos señalamos: la lectura de documentos técnicos, y la búsqueda, elaboración y síntesis de información relacionada con la materia.

- Desarrollar la capacidad de trabajo en equipo, habituándose a la discusión, debate, coordinación y reparto de tareas y responsabilidades entre los miembros del grupo.
- Ser capaz de integrar los conocimientos aprendidos en diferentes temas, o incluso en distintas asignaturas.
- Tener una actitud abierta y positiva hacia la disciplina, en cuanto a la valoración personal de los conocimientos adquiridos y el interés personal por seguir avanzando en la materia.

En definitiva, la asignatura se debe apoyar en un conjunto de técnicas concretas de especificación, análisis y diseño de algoritmos y estructuras de datos, como medio para lograr el objetivo de enseñar a los alumnos a pensar de forma reflexiva, abstracta, fundamentada, crítica y creativa, en el proceso metódico conducente a la resolución de problemas de programación.

3.3.2. Objetivos puntuales de la asignatura

Dentro de este apartado estableceremos los objetivos puntuales de la materia "Ampliación de Algoritmos y Estructura de Datos", es decir, aquellos que pueden ser asociados directamente a temas concretos en la planificación de la asignatura. Estos objetivos nos acercan más a la selección de contenidos, y aportarán una ayuda bastante precisa y directa sobre los contenidos que deben ser incluidos. No obstante, aún no se pretende entrar al nivel de detalle de decidir los temas de la asignatura, que serán establecidos en el siguiente capítulo.

Los objetivos enumerados abajo son el resultado de seleccionar, depurar y reorganizar los objetivos del CC2001 relacionados con la materia, que fueron expuestos en el apartado 3.2.3. Los objetivos han sido estructurados en torno a cuatro grandes bloques: abstracciones, estructuras de datos, análisis de eficiencia, y diseño de algoritmos. Se puede apreciar que, en este caso, los objetivos están enunciados desde el punto de vista del docente.

A. Abstracciones.

A1 Defender la importancia de las abstracciones, especialmente en la programación en proyectos grandes.

A2 Presentar los tipos de abstracciones que aparecen en programación: abstracciones de datos, abstracciones funcionales y abstracciones de iteradores.

A3 Discutir los mecanismos de abstracción que se usan en programación, su significado y utilidad: abstracción por especificación y por parametrización.

A4 Explicar los mecanismos de los lenguajes de programación que permiten el uso de tipos abstractos de datos: tipos definidos por el usuario, módulos y clases.

A5 Explicar cómo los mecanismos de abstracción permiten la creación de código legible, extensible y reutilizable.

A6 Defender el interés de definir tipos genéricos, como resultado de la abstracción por parametrización de tipos.

A7 Presentar varias técnicas de especificación formal e informal.

A8 Discutir el papel de las técnicas de especificación y verificación formal en el contexto de validación y prueba de software.

A9 Usar un lenguaje de especificaciones formales común, formular la especificación de una variedad de tipos abstractos de datos sencillos, y demostrar los beneficios desde la perspectiva de la calidad.

A10 Aplicar técnicas de verificación formal a tipos abstractos de datos de baja complejidad.

A11 Explicar los beneficios e inconvenientes potenciales de usar lenguajes de especificación formales.

A12 Usar y evaluar la técnica de especificación mediante pre- y postcondiciones, aplicada sobre una variedad de tipos de datos y abstracciones funcionales.

B. Estructuras de datos.

B1 Explicar y escribir programas que usen las técnicas: tablas de dispersión, representación de conjuntos mediante árboles y grafos.

B2 Describir cómo las estructuras de datos estudiadas son reservadas y usadas en memoria.

B3 Describir aplicaciones comunes para cada una de las estructuras de datos estudiadas.

B4 Definir el funcionamiento de las estructuras de datos en pseudocódigo e implementarlas en un lenguaje de alto nivel.

B5 Comparar métodos alternativos de implementación de las estructuras de datos con respecto al rendimiento.

B6 Diseñar e implementar funciones de dispersión adecuadas para diversas aplicaciones.

B7 Diseñar e implementar un algoritmo de resolución de colisiones para una tabla de dispersión.

B8 Comparar y contrastar los costes y beneficios de las implementaciones estáticas y dinámicas de las estructuras de datos.

B9 Ilustrar mediante ejemplos la terminología básica de la teoría de grafos, los tipos de grafos y algunas de las propiedades fundamentales de los grafos.

B10 Mostrar diferentes métodos de recorrido de árboles y grafos.

B11 Modelar problemas de computación usando árboles y grafos.

B12 Resolver problemas usando los algoritmos de grafos fundamentales, incluyendo la búsqueda primero en profundidad y en anchura, los caminos mínimos desde un nodo origen o entre todos los nodos, el cierre transitivo, la ordenación topológica, y al menos un algoritmo de árboles de expansión de coste mínimo.

B13 Elegir las estructuras de datos más apropiadas para modelar un problema dado.

C. Análisis de eficiencia.

C1 Explicar el uso de las notaciones asintóticas O-grande, o-pequeña, Ω y Θ para describir la cantidad de trabajo realizado por un algoritmo.

C2 Usar las notaciones O-grande, Ω y Θ para dar las cotas superior, inferior y orden exacto del tiempo y espacio consumidos por un algoritmo.

C3 Determinar la complejidad de algoritmos sencillos, en cuanto a tiempo y uso de memoria.

C4 Deducir relaciones recurrentes para describir la complejidad de algoritmos definidos de manera recursiva.

C5 Resolver relaciones de recurrencia elementales y algunos casos no triviales.

C6 Discutir la eficiencia computacional de los principales algoritmos estudiados en la asignatura.

C7 Discutir otros factores –aparte de la eficiencia computacional– que influyen en la elección de un algoritmo, como el tiempo de programación, facilidad de mantenimiento, legibilidad y robustez.

D. Diseño de algoritmos.

D1 Discutir la importancia de los algoritmos en el proceso de resolución de problemas.

D2 Identificar las propiedades necesarias de un buen algoritmo.

D3 Explicar el concepto de técnica de diseño de algoritmos y esquema algorítmico.

D4 Presentar las siguientes técnicas de diseño: fuerza bruta, avance rápido, divide y vencerás, backtracking, ramificación y poda, y algoritmos heurísticos.

D5 Usar pseudocódigo y/o un lenguaje de programación para implementar, probar y depurar los algoritmos y las técnicas de diseño estudiados.

D6 Para cada una de las técnicas estudiadas, buscar analogías en la vida real que ejemplifiquen el concepto básico subyacente.

D7 Describir los inconvenientes de los algoritmos de fuerza bruta.

D8 Implementar un algoritmo voraz para resolver un problema adecuado.

D9 Implementar un algoritmo de divide y vencerás para resolver un problema adecuado.

D10 Identificar el caso base y el caso general de un problema definido recursivamente.

D11 Comparar soluciones iterativas y recursivas para problemas elementales como el cálculo del factorial.

D12 Usar backtracking para resolver problemas como los de navegación en un puzzle.

D13 Discutir problemas para los cuales el backtracking puede ser una buena solución.

D14 Presentar la ramificación y poda como una generalización y mejora de backtracking.

D15 Describir varios métodos heurísticos de resolución de problemas.

D16 Discutir el compromiso entre tiempo y exactitud en el cálculo de las cotas para un algoritmo de ramificación y poda.

D17 Demostrar las siguientes capacidades: evaluar algoritmos, seleccionar entre una variedad de posibles opciones, dar una justificación de tal selección, e implementar el algoritmo en un contexto de programación.

En el siguiente Capítulo analizaremos la propuesta de estructuración de la asignatura, incluyendo el reparto de estos objetivos entre los diferentes temas.

Capítulo 4

Selección y organización de contenidos

"Hay dos aspectos clave en la calidad de la enseñanza:
vocación por la docencia e interés en la materia.
Lo primero te concederá una actitud entregada,
lo segundo te permitirá transmitir pasión por la disciplina."
José L. Fernández Alemán, Proyecto Docente, 2002

La determinación de los contenidos de un proyecto docente es, en cierto sentido, la culminación del trabajo de recopilación, análisis y reflexión realizado hasta alcanzar este punto. Los contenidos seleccionados se deben ajustar a las particularidades y restricciones del contexto de desarrollo de la asignatura, asentándose sobre las bases que sustentan y fundamentan la disciplina, y orientándose hacia la consecución de los objetivos propuestos. Son, por lo tanto, muchos los factores que deben ser tenidos en cuenta. Y, al mismo tiempo, resulta imprescindible hacer una buena elección para lograr una docencia de calidad y alcanzar los propósitos que se persiguen.

Afortunadamente, no partimos de cero en esta selección de contenidos. En primer lugar, es evidente que el propio **programa actual** de la asignatura en la FIUM será un factor a considerar, puesto que es necesario alcanzar un justo equilibrio entre actualización y continuidad. Un cambio radical en la planificación de la asignatura podría provocar desorientación en los alumnos e inseguridad en el profesor; pero igualmente imprescindible es una revisión y actualización continua de los contenidos, así como de los materiales y la metodología docente. Por otro lado, las distintas **recomendaciones internacionales** disponibles –fundamentalmente las contenidas en el informe CC2001– suponen una buena guía, que aporta una dimensión más amplia que lo meramente local en la determinación de lo que es adecuado y lo que no. Finalmente, el estudio y contraste de los **planes de estudios de otras universidades** españolas nos dará una visión global de los contenidos que se están impartiendo actualmente en nuestro entorno nacional.

Tras analizar los distintos programas y recomendaciones disponibles, realizamos la propuesta de contenidos para la asignatura, la estructuración y organización temporal de los mismos, y la distribución entre actividades teóricas y prácticas. En este Capítulo se hace una descripción global y general de la organización propuesta en este proyecto docente. En los siguientes capítulos se desarrollan y explican estos contenidos de manera más detallada.

4.1. Programa actual de la asignatura

Aunque omitido en muchos proyectos docentes, consideramos fundamental la inclusión y el análisis del programa actual de la asignatura objeto de estudio. Excepto en los casos de nueva implantación, el diseño de cualquier asignatura siempre tomará como referencia la planificación de la misma en años anteriores. Esto no implica necesariamente un continuismo sino que, más bien al contrario, aporta una información esencial para conocer los puntos débiles del plan –ya sea en cuanto a contenidos, organización o enfoque general– que deben ser reformados. Presentaremos en primer lugar el programa de la parte teórica y a continuación el de la parte de prácticas.

Programa de teoría de AAED y AED

Revisando los programas de varios años anteriores, podemos apreciar que el temario de teoría de la asignatura "Algoritmos y Estructuras de Datos", del plan de 2002, coincide básicamente con el de "Ampliación de Algoritmos y Estructura de Datos", del plan de 1996. Este hecho resulta lógico, ya que ambas tienen similares descriptores, la misma carga teórica y un entorno curricular casi idéntico. En ambos casos, la asignatura se divide en dos grandes partes, una dedicada a las estructuras de datos y otra a los algoritmos. El programa de teoría durante el curso 2002/03 fue el siguiente.

- **PARTE I. ESTRUCTURAS DE DATOS**

1. Abstracciones y especificaciones
   1.1 Introducción
   1.2 Especificaciones informales
   1.3 Especificaciones formales algebraicas
   1.4 Especificaciones formales constructivas

2. Conjuntos
   2.1 Introducción a los conjuntos
   2.2 Implementaciones sencillas de conjuntos
   2.3 El TAD diccionario
   2.4 Las tablas de dispersión. Tipos de dispersión
   2.5 Asociaciones muchos a muchos
   2.6 Estructuras de datos duales

3. Métodos avanzados de representación de conjuntos
   3.1 Árboles Trie
   3.2 Relaciones de equivalencia
   3.3 Árboles de búsqueda balanceados
   3.4 Árboles B

4. Grafos
   4.1 Introducción y definiciones
   4.2 Representaciones de grafos
   4.3 Problemas y algoritmos sobre grafos
      4.3.1 Recorridos sobre grafos
      4.3.2 Árboles de expansión

4.3.3 Problemas de caminos mínimos
4.3.4 Algoritmos sobre grafos dirigidos
4.3.5 Puntos de articulación y componentes biconexos
4.3.6 Otros problemas con grafos

- **PARTE II: ALGORÍTMICA**

5 Algorítmica
5.1 Definición y propiedades
5.2 Análisis y diseño de algoritmos
5.3 Heurísticas para una buena programación

6 Análisis de algoritmos
6.1 Introducción
6.2 Notaciones asintóticas
6.3 Ecuaciones de recurrencia
6.4 Ejemplos

7 Divide y vencerás
7.1 Método general
7.2 Análisis de tiempos de ejecución
7.3 Ejemplos de aplicación

8 Algoritmos voraces
8.1 Método general
8.2 Análisis de tiempos de ejecución
8.3 Ejemplos de aplicación
8.4 Técnicas heurísticas

9 Programación dinámica
9.1 Método general
9.2 Análisis de tiempos de ejecución
9.3 Ejemplos de aplicación

10 Backtracking
10.1 Método general
10.2 Análisis de tiempos de ejecución
10.3 Ejemplos de aplicación

11 Ramificación y poda
11.1 Método general
11.2 Análisis de tiempos de ejecución
11.3 Ejemplos de aplicación

Programa de prácticas de AAED y AED

La diferencia entre las dos asignaturas es más sustancial en lo referente al programa de prácticas, debido a la distinta carga asignada a cada una. La asignatura AAED tiene únicamente 2 créditos prácticos, por lo que las prácticas son planteadas como ejercicios de pizarra y en papel. El programa indica lo siguiente:

- Los alumnos recibirán un **boletín de ejercicios** por cada uno de los temas de teoría. Las prácticas consisten en la resolución de algunos de estos de ejercicios en papel. En concreto habrán dos entregas de estos ejercicios: una para la primera parte y otra para la segunda parte de la asignatura, con sus correspondientes entrevistas. Estas prácticas son individuales.

- Por otro lado, se realizarán **prácticas de pizarra** a la finalización de cada tema de teoría, que consistirán en la resolución por parte de los alumnos, en la pizarra, de algunos de los ejercicios entregados.

Por su parte, la asignatura AED tiene asignados 6 créditos prácticos, distribuidos a lo largo de los dos cuatrimestres. Esta carga da opción a la realización de prácticas de programación. Durante el presente curso 2003/04 esta asignatura ha sido impartida por primera vez, decidiéndose la utilización de los lenguajes C/C++ en las prácticas. La planificación de las prácticas ha sido la siguiente:

- Las prácticas consistirán en el análisis, diseño, implementación y prueba de algunos problemas de algoritmos y de estructuras de datos, usando el lenguaje de programación C++.

- Los 6 créditos de prácticas se reparten en 1,5 créditos en el seminario de introducción a C/C++, y 4,5 créditos repartidos en 2 ó 3 prácticas en modalidad de laboratorio abierto.

  - **Seminario 1:** Programación en C (9 horas)
  - **Seminario 2:** Programación en C++ (6 horas)
  - **Práctica 1:** Implementación y manejo de estructuras de datos (1,5 créditos)
  - **Práctica 2:** Eficiencia, evaluación y predicción (1,5 créditos)
  - **Práctica 3:** Resolución de problemas (1,5 créditos)

4.2. Recomendaciones internacionales

En la mayoría de las recomendaciones curriculares, tanto nacionales como internacionales, la enseñanza de los conceptos fundamentales de programación es planteada como un aprendizaje cuyo desarrollo tiene lugar a lo largo de los dos o tres primeros cursos. Este es el contexto natural en el que se sitúa la materia objeto de concurso. Pero, obviamente, las recomendaciones no necesariamente incluyen una denominación "Ampliación de Algoritmos y Estructura de Datos", por lo que es necesario encontrar un equivalente[1] en las mismas a la asignatura AAED. Este equivalente suele recibir distintas denominaciones, como: "Estructuras de Datos y Algoritmos", "Abstracción de Datos", "Esquemas Algorítmicos", o "Análisis y Diseño de Algoritmos", entre otras. En cualquier caso, nos guiamos por los objetivos educativos formulados en el capítulo 3.

[1] O los equivalentes, porque incluso puede que los contenidos estén distribuidos entre distintas asignaturas.

4.2.1. La propuesta ACM-IEEE CC2001

Hasta este punto, hemos hecho uso de las recomendaciones del informe CC2001 dentro del estudio del contexto curricular, en el apartado 2.2.2, y en el establecimiento de los objetivos de la asignatura, en el apartado 3.2.3. En este apartado volvemos a considerar este valioso documento en lo relativo a los contenidos concretos de la asignatura AAED.

Las aportaciones del CC2001 pueden venir en dos sentidos. Por un lado, en cuanto a la **organización y estructura** general que se propone para la asignatura "Estructuras de Datos y Algoritmos", incluida en varios de los modelos de currículum del CC2001. Por otro lado, en lo relativo a los **contenidos concretos** que se sugieren para cada una de las unidades definidas en el CC2001[2]. Ambas facetas se complementan para ofrecer unas recomendaciones muy útiles y detalladas. Mientras que la primera provee una idea general de los grandes bloques de la asignatura, la segunda concreta los aspectos a tratar en cada tema.

Recomendaciones sobre la asignatura AED

Los planes de estudios de la FIUM –tanto los antiguos como los nuevos, y en las tres titulaciones impartidas– siguen lo que el informe CC2001 denomina el acercamiento **imperativo primero**, como vimos en el apartado 2.2.2. Dentro de este enfoque el documento propone dos implementaciones: una tradicional y otra compacta. En la primera, la enseñanza de la programación se distribuye a lo largo de tres semestres, en los que se impartirían las siguientes asignaturas:

- CS101I. Fundamentos de Programación.
- CS102I. El Paradigma Orientado a Objetos.
- CS103I. Estructuras de Datos y Algoritmos.

La segunda implementación se desarrolla en sólo dos semestres, estando definidas las asignaturas:

- CS111I. Introducción a la Programación.
- CS112I. Abstracción de Datos.

Podemos establecer que la asignatura que más se asemeja a "Ampliación de Algoritmos y Estructura de Datos" –no sólo por la similitud de denominación, sino también por los contenidos y la ubicación en el plan de estudios– es la CS103I, "Estructuras de Datos y Algoritmos". La descripción que hace el CC2001 de esta asignatura puede ser una interesante orientación sobre la estructuración de contenidos en nuestro caso. Pero no hay que perder de vista que se parte de un contexto de plan de estudios bien distinto, por lo que no es posible su aplicación inmediata a este proyecto docente. Veamos las recomendaciones del CC2001 para la asignatura CS103I.

> **CS103I. Estructuras de Datos y Algoritmos.**
> Sobre las bases provistas por la secuencia de asignaturas CS101I-CS102I, se introducen los conceptos fundamentales sobre estructuras de datos y algoritmos como continuación de los anteriores. Los tópicos incluyen la recursividad, la filosofía subyacente a la programación orientada a objetos, las estructuras de datos fundamentales (incluyendo pilas, colas, listas enlazadas, tablas de dispersión, árboles y grafos), las bases del análisis de algoritmos, y una introducción a los principios de la traducción de lenguajes.
> *Prerrequisitos:* CS102I; matemáticas discretas es también deseable.
> *Programa:*

[2] Recordemos que el CC2001 define áreas, unidades y asignaturas. Un área contiene varias unidades, y una asignatura se forma combinando unidades de varias áreas distintas.

- Repaso de conceptos elementales de programación
- Estructuras de datos fundamentales: pilas; colas; listas enlazadas; tablas de dispersión; árboles; grafos
- Programación orientada a objetos: diseño orientado a objetos; encapsulación y ocultamiento de información; clases; separación de comportamiento e implementación; jerarquías de clases; herencia; polimorfismo
- Algoritmos fundamentales en computación: algoritmos de ordenación $O(N \log N)$; tablas de dispersión, incluyendo estrategias de resolución de colisiones; árboles binarios de búsqueda; representación de grafos; recorridos en profundidad y en anchura
- Recursividad: el concepto de recursividad; funciones matemáticas recursivas; procedimientos recursivos sencillos; estrategias divide y vencerás; backtracking recursivo; implementación de la recursividad
- Análisis de algoritmos básico: análisis asintótico de las cotas superior y promedio de complejidad; identificando las diferencias entre los casos mejor, peor y promedio; notaciones O-grande, o-pequeña, Omega y Theta; clases estándar de complejidad; medidas empíricas del rendimiento; compromisos entre tiempo y espacio en los algoritmos; uso de relaciones de recurrencia para analizar algoritmos recursivos
- Estrategias algorítmicas: algoritmos de fuerza bruta; algoritmos voraces; divide y vencerás; backtracking; ramificación y poda; heurísticas; búsqueda de patrones y algoritmos con cadenas/texto; algoritmos de aproximación numérica
- Repaso de lenguajes de programación: paradigmas de programación
- Ingeniería del software: validación del software; fundamentos de pruebas, incluyendo creación de planes de prueba y generación de casos de prueba; pruebas orientadas a objetos

Unidades cubiertas:

DS5	Grafos y árboles	2	horas (de 4)
PF3	Estructuras de datos fundamentales	12	horas (de 14)
PF4	Recursividad	5	horas (de 5)
AL1	Análisis de algoritmos básico	2	horas (de 4)
AL2	Estrategias algorítmicas	3	horas (de 6)
AL3	Algoritmos fundamentales en computación	5	horas (de 12)
AL5	Computabilidad básica	1	hora (de 6)
PL1	Lenguajes de programación	1	hora (de 2)
PL6	Programación orientada a objetos	8	horas (de 10)
SE6	Validación de software	1	hora (de 3)

En conjunto, la asignatura propuesta presenta tres grandes bloques: estructuras de datos (DS5 + PF3 = 14 horas), algorítmica (PF4 + AL1 + AL2 + AL3 = 15 horas) y orientación a objetos (PL6 = 8 horas). Estos se complementan con otras unidades, tratando de incorporar todos los conceptos fundamentales de programación. En cuanto a la disposición temporal de estos contenidos, se sugiere una organización del tipo: 1º) orientación a objetos; 2º) estructuras de datos; 3º) análisis de algoritmos; 4º) diseño de algoritmos. En el fondo, la estructura es bastante parecida a la organización actual de la asignatura AED –como vimos en la sección 4.1–, que consta de los grandes bloques: 1º) abstracciones y especificaciones; 2º) estructuras de datos; 3º) análisis de algoritmos; 4º) diseño de algoritmos. De esta manera se resuelven algunas de las grandes cuestiones de base sobre la organización de contenidos: ¿van primero los algoritmos o las estructuras de datos?, ¿va antes el análisis o el diseño de algoritmos?

La similitud en los contenidos concretos es también bastante notable, principalmente en cuanto a las técnicas de diseño de algoritmos y a las estructuras de datos. No obstante, algunas diferencias en las asignaturas previas o posteriores del plan aconsejan la elección de distintos contenidos en algunos puntos. Podemos destacar las siguientes cuestiones:

- **Programación orientada a objetos.** Los planes de estudios de la FIUM retrasan la enseñanza de la POO hasta el tercer curso, donde existe una asignatura específica. Frente a esto, el CC2001 aconseja un estudio mucho más temprano de este paradigma, incluso en el acercamiento imperativo primero. Esto explica la elevada carga que se establece para la POO en las recomendaciones. Por lógica, la planificación de la asignatura AAED debe reducir sustancialmente esta carga, evitando redundancias con asignaturas posteriores. No obstante, creemos que todos los factores apuntan a una necesaria introducción progresiva de los aspectos más básicos de la POO en las asignaturas de segundo curso. Discutiremos este tema en profundidad más adelante.

- **Recursividad.** Las recomendaciones fijan la recursividad como un tema básico y exclusivo de la asignatura CS103I, lo que justifica la elevada carga establecida para esta unidad. Esto contrasta con la tradición y los planes de estudios de la FIUM, donde el estudio de la recursividad tiene lugar desde el primer curso. La importancia del tema aconseja tratarlo también en segundo curso, al que los alumnos llegarán habiendo estudiado divide y vencerás y backtracking como ejemplos de aplicación de la recursividad.

- **Computabilidad y traducción.** Aunque con una baja carga, las recomendaciones incluyen la computabilidad y la traducción de lenguajes entre los contenidos de la asignatura CS103I. No obstante, estos temas están suficientemente cubiertos en los planes de 2002 de II, que incluyen las asignaturas optativas u obligatorias "Autómatas y Lenguajes Formales", "Computabilidad" y "Traductores". Entre las tres suman 24 créditos.

Como ya hemos mencionado, la diferencia básica entre la propuesta imperativo primero del informe CC2001 y los planes actuales de la FIUM es el intercambio de POO por AED, en cuanto a la secuenciación temporal. Esto conlleva otros cambios, debido a la necesidad de incorporar algunas unidades básicas que el CC2001 incluye en la asignatura CS102I, "El Paradigma Orientado a Objetos". Entre estas unidades señalamos: PF2 - Algoritmos y resolución de problemas; y PF3 - Estructuras de datos fundamentales. En definitiva, las unidades cubiertas por cada una de las asignaturas de los planes de estudios actuales de Ingeniero en Informática en la FIUM, y las propuestas en la estrategia imperativo primero son mostradas en la tabla 4.1. Una X mayúscula indica un contenido esencial de la asignatura, y una x minúscula un aspecto tratado de manera superficial. En el siguiente punto entraremos en detalle sobre los contenidos que se aconsejan para cada una de estas unidades de interés.

En general, se puede observar que todas las unidades que forman parte del núcleo de las recomendaciones curriculares están suficientemente cubiertas por el plan de estudios, aunque la distribución entre las asignaturas difiera en parte de la propuesta. Recordemos que las horas asignadas a las unidades no corresponden exactamente a horas de clase, ya que se supone un trabajo previo del alumno de tres horas por cada hora de clase. Por otro lado, las unidades que no tienen horas asignadas son contenidos optativos.

Recomendaciones sobre los contenidos de las unidades

Después de analizar las unidades del CC2001 que forman parte de la asignatura "Ampliación de Algoritmos y Estructura de Datos", vamos a examinar los tópicos o contenidos específicos que se proponen para cada unidad. Recordemos que las unidades seleccionadas son: DS5 - Grafos y árboles; PF2 - Algoritmos y resolución de problemas; PF3 - Estructuras de datos fundamentales; PF4 - Recursividad; AL1 - Análisis de algoritmos básico; AL2 - Estrategias algorítmicas; AL3 - Algoritmos fundamentales en computación; y PL5 - Mecanismos de abstracción; todas ellas parte del núcleo básico de conocimientos.

			Ing. Informática, planes de 2002					**CC'2001, imperativo 1º**		
Cod.	**Tópicos y Unidades**	**Horas núcleo**	**MTP**	**AED**	**POO**	**Otras**		**CS101I**	**CS102I**	**CS103I**
DS	**Estructuras Discretas**									
DS1	Funciones, relaciones y conjuntos	6		x		MD				
DS2	Lógica básica	10	x			MD				
DS3	Técnicas de demostración	12	x			MD				
DS4	Bases del conteo	5				MD				
DS5	Grafos y árboles	4	X	X		MD				X
PF	**Fundamentos de Programación**									
PF1	Construcciones fundam. de program.	9	X					X	X	
PF2	Algoritmos y resolución de problemas	6	X	X				X	X	
PF3	Estructuras de datos fundamentales	14	X	X				X	X	X
PF4	Recursividad	5	X	X		CM				X
PF5	Programación dirigida por eventos	4			X				X	
AL	**Algoritmos y Complejidad**									
AL1	Análisis de algoritmos básico	4	X	X						X
AL2	Estrategias algorítmicas	6	X	X						X
AL3	Algoritmos fund. en computación	12	X	X					X	X
AL4	Algoritmos distribuidos	3				AP				
AL5	Computabilidad básica	6				CM				X
AL6	Las clases de complejidad P y NP		x	x		CM				
AL7	Teoría de autómatas					AF				
AL8	Análisis de algoritmos avanzado					AP				
PL	**Lenguajes de Programación**									
PL1	Repaso de lenguajes de programación	2	x		X				X	X
PL2	Máquinas virtuales	1	x		X				X	
PL3	Introducción a traducción de lenguajes	2	x			AF		X	X	
PL4	Declaraciones y tipos	3	x	x	x	AF		X		
PL5	Mecanismos de abstracción	3	x	X	x			X		
PL6	Programación orientada a objetos	10		x	X				X	X
SP	**Cuestiones Sociales y Profesionales**									
SP1	Historia de la informática	1	x		x			X		
SE	**Ingeniería del Software**									
SE5	Requerimientos y especificaciones sw.	4		x					X	
SE6	Validación de software	3	x	x					X	X
SE10	Métodos formales		x	X		AF				

Tabla 4.1: Unidades cubiertas por las asignaturas del plan de estudios de II, de 2002, en la FIUM, y por las asignaturas definidas en el CC2001, acercamiento imperativo primero. Asignaturas: MTP - Metodología y Tecnología de la Programación; AED - Algoritmos y Estructuras de Datos; POO - Programación Orientada a Objetos; MD - Álgebra y Matemática Discreta; CM - Computabilidad; AP - Algoritmos y Programación Paralela; AF - Autómatas y Lenguajes Formales; CS101I - Fundamentos de Programación; CS102I - El Paradigma Orientado a Objetos; CS103I - Estructuras de Datos y Algoritmos.

Ya hemos visto que algunas de estas unidades son compartidas por otras asignaturas –fundamentalmente con MTP–, y al mismo tiempo algunos contenidos pueden no ser abordados en AAED. En la siguiente lista señalamos con el símbolo "•" los contenidos descritos en el informe CC2001 que son propios de AAED, y con "◦" los que son tratados superficialmente, o bien no son tratados en absoluto al ser suficientemente estudiados en otras asignaturas; entre paréntesis se muestran las otras asignaturas que comparten ese contenido particular. Esta información ha sido extraída consultando los programas de las asignaturas para el curso 2003/04, que se encuentran en el apéndice A.

- **DS5 - Grafos y árboles**
  Tiempo mínimo necesario: 4 horas
  Contenidos:
  - (MD) Árboles
  - (MD) Grafos no dirigidos
  - (MD) Grafos dirigidos
  - Árboles de expansión de un grafo
- **PF2 - Algoritmos y resolución de problemas**
  Tiempo mínimo necesario: 6 horas
  Contenidos:
  - (MTP) Estrategias de resolución de problemas
  - (MTP) El papel de los algoritmos en el proceso de resolución de problemas
  - ◦ (MTP) Estrategias de implementación para los algoritmos
  - ◦ (MTP) Estrategias de depuración
  - (MTP) El concepto y propiedades de algoritmo
- **PF3 - Estructuras de datos fundamentales**
  Tiempo mínimo necesario: 14 horas
  Contenidos:
  - ◦ (MTP) Tipos primitivos
  - ◦ (MTP) Arrays
  - ◦ (MTP) Registros
  - ◦ (MTP) Cadenas y procesamiento de cadenas
  - (MTP) Representación de datos en memoria
  - ◦ (MTP) Reserva de memoria estática, en pila y montón
  - ◦ (MTP) Manejo de almacenamiento en tiempo de ejecución
  - ◦ (MTP) Punteros y referencias
  - (MTP) Estructuras enlazadas
  - (MTP) Estrategias de implementación de pilas, colas y tablas de dispersión
  - Estrategias de implementación de grafos y árboles
  - Estrategias para elegir la estructura de datos adecuada

- **PF4 - Recursividad**
  Tiempo mínimo necesario: 5 horas
  Contenidos:
  - (MTP) El concepto de recursividad
  - (Cálculo) Funciones matemáticas recursivas
  - (MTP) Procedimientos recursivos sencillos
  - (MTP) Estrategias divide y vencerás
  - (MTP) Backtracking recursivo
  - (MTP) Implementación de la recursividad
- **AL1 - Análisis de algoritmos básico**
  Tiempo mínimo necesario: 4 horas
  Contenidos:
  - (MTP) Análisis asintótico de las cotas superior y promedio de complejidad
  - Identificando diferencias entre los comportamientos en los casos mejor, peor y promedio
  - (MTP) Notaciones O-grande, o-pequeña, Omega y Theta
  - Clases de complejidad estándar
  - Medidas empíricas del rendimiento
  - El compromiso del tiempo y el espacio en los algoritmos
  - Uso de relaciones recurrentes para analizar algoritmos recursivos
- **AL2 - Estrategias algorítmicas**
  Tiempo mínimo necesario: 6 horas
  Contenidos:
  - (MTP) Algoritmos de fuerza bruta
  - Algoritmos voraces
  - (MTP) Divide y vencerás
  - (MTP) Backtracking
  - Ramificación y poda
  - Heurísticas
  - (MTP) Búsqueda de patrones y algoritmos sobre cadenas/texto
  - (Cálculo) Algoritmos de aproximación numérica
- **AL3 - Algoritmos fundamentales en computación**
  Tiempo mínimo necesario: 12 horas
  Contenidos:
  - (Cálculo) Algoritmos numéricos simples
  - (MTP) Algoritmos de búsqueda secuencial y binaria
  - (MTP) Algoritmos cuadráticos de ordenación (selección e inserción)
  - (MTP) Algoritmos de ordenación $O(N \log N)$ (quicksort, heapsort, mergesort)

- Tablas de dispersión, incluyendo estrategia de resolución de colisiones
- (MTP) Árboles binarios de búsqueda
- Representación de grafos (listas y matrices de adyacencia)
- Recorridos primero en profundidad y en anchura
- Algoritmos de caminos mínimos (algoritmos de Dijkstra y Floyd)
- Cierre transitivo (algoritmo de Floyd)
- Árboles de expansión de coste mínimo (algoritmos de Prim y Kruskal)
- Ordenación topológica

- **PL5 - Mecanismos de abstracción**
  Tiempo mínimo necesario: 3 horas
  Contenidos:

  - (MTP) Procedimientos, funciones e iteradores como mecanismos de abstracción
  - (MTP) Mecanismos de parametrización (referencia vs. valor)
  - (MTP) Registros de activación y manejo de almacenamiento
  - Parámetros de tipo y tipos parametrizados
  - (MTP) Módulos en los lenguajes de programación

Se puede observar que todos los contenidos son tratados, o bien por AED o bien por otra asignatura del plan de estudios. Muchos de los puntos son introducidos en primer curso en MTP, retomándose en AED con el objetivo de profundizar sobre los mismos. Podemos concluir de este hecho la necesidad de coordinar los contenidos de AED con los de MTP, buscando un acoplamiento adecuado que fije los conceptos fundamentales de programación a la vez que evite duplicación de contenidos.

Además de estos tópicos, que forman parte del núcleo de conocimientos del CC2001, la asignatura AAED, en su planificación actual, incluye otros temas que las recomendaciones establecen como optativos. En concreto podemos mencionar dos: métodos de especificación formal y la técnica de programación dinámica; correspondientes a las unidades SE10 y AL8, respectivamente. Los contenidos que se definen para estos temas son los siguientes:

- **SE10 - Métodos formales**
  Contenidos:

  - (MTP) Conceptos de los métodos formales
  - Lenguajes de especificación formal
  - Especificaciones ejecutables y no ejecutables
  - (MTP) Pre- y postcondiciones
  - (TFIS[3]) Verificación formal

- **AL8 - Análisis de algoritmos avanzado**
  Contenidos:

  - Análisis amortizado

[3] "Técnicas Formales en Ingeniería del Software", asignatura optativa de 5º curso.

- (AP) Algoritmos online y offline
- (AP) Algoritmos aleatorizados
- Programación dinámica
- Optimización combinatoria

4.2.2. Las propuestas UNESCO-IFIP

El propósito de estas recomendaciones, contenidas en el informe [UNESCO'94], es ofrecer a los países menos desarrollados la experiencia de las universidades europeas y americanas en el diseño de planes de estudios en informática, como ya comentamos al final del apartado 2.2.2. Lejos de la flexibilidad y el detalle del CC2001, el documento se limita a definir y describir un conjunto de asignaturas, que pueden ser programadas modularmente en secuencias diversas. Quizá esta concreción, y el repaso de la bibliografía más relevante para cada asignatura, son los aspectos más interesantes de estas recomendaciones. Por lo demás, debemos considerarlas con la debida cautela, debido a que la última revisión disponible se remonta a 10 años atrás.

Las asignaturas de este currículum más relacionadas con AAED son: "Algoritmos y Programación Estructurada", "Estructuras de Datos y Algoritmos" y "Diseño y Análisis de Algoritmos". Mientras que las dos primeras se consideran básicas en el currículum mínimo, la tercera no es incluida; y ello a pesar de que es en esta asignatura donde se estudian las técnicas de diseño de algoritmos. Vamos a ver brevemente los contenidos y bibliografía propuestos para estas asignaturas.

Asignatura "Algoritmos y Programación Estructurada"

- **Objetivos.** Continuar con el desarrollo de la disciplina en diseño de programas, en estilo, expresión, depuración y pruebas, especialmente en programas grandes; introducir el análisis de algoritmos; introducir los aspectos básicos de procesamiento de cadenas, recursividad, métodos internos de ordenación y búsqueda y estructuras de datos sencillas.

- **Descripción y contenidos.** Tras una primera asignatura previa de introducción a la programación, esta asignatura pretende hacer una profundización en las buenas prácticas de programación, especialmente en proyectos de tamaño grande. En relación con esto, se argumenta la necesidad de desarrollar proyectos de programación en grupos, donde los alumnos aprendan técnicas de coordinación y trabajo en equipo. También se introduce el análisis de algoritmos, aunque de manera muy básica y *"a este nivel tal análisis debe ser dado por el profesor a los estudiantes"*. Los contenidos incluyen una introducción a la programación orientada a objetos: objetos, clases, métodos y herencia. Para lograr los propósitos de la asignatura se aconseja introducir un segundo lenguaje de programación, diferente al usado en el primer curso.

- **Bibliografía.** Para esta asignatura se proponen dos libros de programación orientada a objetos, [Meyer'88] y [Rumbaugh'91], tres de algoritmos, [Knuth'97], [Linger'79] y [Wirth'80], y uno de programación en C++, [Stroustrup'98][4].

Asignatura "Estructuras de Datos y Algoritmos"

- **Objetivos.** Introducir las estructuras de datos más comunes; dar métodos alternativos de organización y representación de los datos.

[4]Para algunas de estas referencias se indica la versión traducida al español del libro propuesto originalmente en las recomendaciones de la UNESCO.

- **Descripción y contenidos.** El temario de esta asignatura agrupa una gran variedad de tipos y estructuras de datos, desde los bits y los bytes, hasta las estructuras arbóreas. Las estructuras incluyen: arrays simples, arrays multidimensionales, listas, pilas, colas, listas múltiples (u ortogonales), diccionarios, árboles, árboles binarios de búsqueda, y árboles B. Se considera tanto la representación en memoria de estas estructuras como su implementación en ficheros de disco. Se debe hacer énfasis en el análisis del tiempo y el espacio requerido por cada una de las técnicas estudiadas. Las recomendaciones aconsejan introducir un gran número de ejemplos, así como una fuerte carga práctica para esta asignatura, en la que los alumnos puedan trabajar con estas estructuras.

- **Bibliografía.** La bibliografía para esta asignatura no es muy extensa. Además de dos referencias clásicas de programación, [Knuth'97] y [Wirth'87], se propone el libro [Nievergelt'93], que tiene la particularidad de incluir aplicaciones de los algoritmos y las estructuras de datos al ámbito de los gráficos y la geometría.

Asignatura "Diseño y Análisis de Algoritmos"

- **Objetivos.** Hacer una introducción exhaustiva al problema central de la informática: el diseño de estructuras de datos apropiadas y algoritmos eficientes que funcionan sobre las mismas, para resolver problemas en todas las áreas de la informática y sus aplicaciones. Puesto que el campo es realmente mucho más amplio, se deberían incluir asignaturas especializadas en los tópicos más importantes: geometría computacional, algoritmos paralelos, distribuidos, probabilistas, VLSI, algoritmos sobre grafos, etc.

- **Descripción y contenidos.** Por la descripción realizada en el informe [UNESCO'94], esta es la asignatura que más se parece a AAED. En ella se pretende profundizar en el estudio no sólo de algoritmos sino también de tipos y estructuras de datos. El programa empieza con una introducción a los métodos de especificación formal algebraica, que son aplicados sobre las estructuras de datos básicas: listas, pilas, colas, árboles. Después se ven algoritmos de ordenación, representación de conjuntos y algoritmos de manipulación, y grafos, incluyendo representación y algoritmos. Se señala también el estudio de técnicas de análisis de la complejidad de los algoritmos, y métodos generales de diseño de algoritmos. No obstante, estos métodos no son enumerados en la lista de contenidos, sino que se identifican otro tipo de ámbitos donde se usan algoritmos: computación numérica, algoritmos paralelos, distribuidos y probabilistas.

  Es curioso cómo a pesar de la importancia que se le atribuye a esta asignatura en la lista de objetivos, sin embargo no es incluida en muchos de los modelos de planes de estudios que se proponen posteriormente.

- **Bibliografía.** La bibliografía recomendada para esta asignatura es bastante amplia. Se proponen doce libros, entre los que destacamos [Cormen'90], [Gonnet'91], [Kozen'91], [Manber'89], [Mehlhorn'84] y [Sedgewick'88]. También se proponen otras referencias de algoritmos distribuidos y paralelos, y de geometría computacional.

Dejando a un lado la valoración de la vigencia o adecuación de las recomendaciones, podemos extraer algunas conclusiones de interés sobre la asignatura objeto de estudio. Por un lado, resulta fundamental que la planificación de la asignatura combine de manera adecuada la teoría y las prácticas, permitiendo a los alumnos trabajar en grupos sobre los distintos aspectos tratados en teoría. Para estas prácticas el informe sugiere la utilización de un segundo lenguaje, y por tanto distinto al estudiado en

primer curso. En este aspecto resulta coincidente con las recomendaciones del CC2001, que proponía el aprendizaje y dominio de al menos dos lenguajes y dos paradigmas diferentes. También coinciden ambas recomendaciones en la necesidad de introducir la programación orientada a objetos desde los primeros cursos. Por otro lado, la asignatura debe extenderse fundamentalmente en anchura, abordando técnicas de análisis y diseño de algoritmos y estructuras de datos no tratadas en otras asignaturas. Aparte de las técnicas coincidentes con las del CC2001, podemos señalar en el informe UNESCO-IFIP las técnicas de especificación formal algebraica, las listas múltiples, diccionarios, árboles B, y los algoritmos geométricos. Debemos valorar hasta qué punto estos temas deben ser cubiertos por AAED, o son cubiertos ya por otras asignaturas de los planes actuales.

4.3. La asignatura en otras universidades españolas

Antes de entrar de lleno en la selección y organización de los contenidos de la asignatura, vamos a revisar los planes de estudios de Ingeniero en Informática de otras universidades españolas, así como los temarios concretos de las asignaturas relacionadas con los descriptores definidos para la materia "Ampliación de Algoritmos y Estructura de Datos". No es fácil identificar estas asignaturas, ya que los contenidos que en nuestra Facultad se incluyen en una única asignatura, en otros centros pueden estar distribuidos en distintas asignaturas de diferentes cursos. Además, algunas de esas asignaturas pueden incluir contenidos que en la FIUM no son responsabilidad de AAED. Aún así, el interés de analizar estos planes de estudios reside en la posibilidad de conocer qué selección y organización de contenidos ha sido realizada en asignaturas con un contexto personal, curricular e institucional parecido al de la materia objeto de estudio.

En primer lugar haremos un breve repaso de la enseñanza de la programación en los primeros cursos, como parte esencial del ámbito donde se sitúa la materia AAED. El enfoque propuesto para estos cursos introductorios será uno de los elementos de mayor interés a analizar.

4.3.1. La introducción a la programación

La enseñanza de la programación en el primer curso de una titulación de informática debe poner un especial énfasis en la visión metodológica e ingenieril de la programación, y no en la sintaxis o en las cuestiones propias de los lenguajes de programación. De esta forma se tendrá una buena base para la comprensión de la materia AAED. En este apartado haremos una revisión de las conclusiones extraídas en el proyecto docente [Fernández'02], donde se analizan 33 asignaturas de introducción a la programación de 25 universidades españolas. Se trata, por lo tanto, de un estudio muy completo que abarca más de la mitad de los más de 40 centros universitarios que imparten alguna de las titulaciones de informática en España.

Dos de los aspectos en los que se centra el referido estudio son el paradigma y el lenguaje de programación empleados en el primer curso. En la figura 4.1 se muestran los resultados extraídos a este respecto.

Se puede observar un claro predominio del paradigma imperativo, que es impartido de forma exclusiva en el 55 % de las titulaciones. Dentro de este paradigma, los lenguajes tradicionales como Pascal, C y Ada son los más usados. El 33 % de las titulaciones optan por incluir un segundo paradigma en primer curso, fundamentalmente el orientado a objetos; el 3 % –correspondiente a 1 universidad– introduce un lenguaje declarativo, como es Haskell. Sólo el 12 % de las titulaciones adopta plenamente el enfoque objetos primero. Dentro de este, los lenguajes más usados son Java y C++. Este último es muy usado también en centros que adoptan el enfoque mixto PI/POO, empezando con la enseñanza de C para a continuación avanzar a C++.

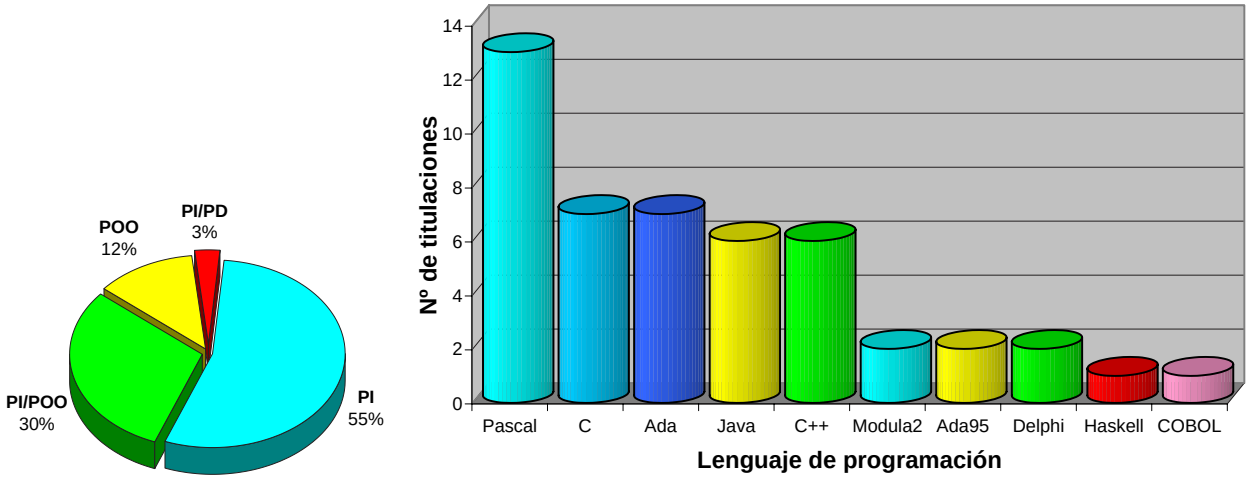


Figura 4.1: Paradigmas y lenguajes de programación usados en 33 titulaciones de 25 universidades españolas, durante el curso 2001/02, [Fernández'02]. PI: paradigma imperativo; POO: paradigma orientado a objetos; PD: paradigma declarativo.

Además del lenguaje y el paradigma, podemos resaltar otros aspectos del estudio. Existe un conjunto de temas que son introducidos de forma generalizada en las asignaturas de programación de primer curso. Entre los más habituales podemos mencionar: abstracciones de control, abstracciones procedimentales, tipos de datos básicos, tipos de datos estructurados, lenguajes de programación, instrucciones básicas, y punteros. Son también una mayoría las titulaciones que incluyen aspectos básicos de metodología de programación (análisis, diseño, verificación, documentación, etc.).

Otros temas presentan una mayor disparidad de criterios, siendo incluidos en primero o en segundo curso según el sitio donde se imparten. De estos temas más discutidos, son mayoritariamente impartidos en primero: la recursividad y los tipos lista, pila y cola. Son más propios de segundo curso: las especificaciones formales y los tipos árbol, grafo y tabla de dispersión. En una situación intermedia entre primero y segundo se encuentran: los algoritmos de clasificación y búsqueda, el análisis de complejidad (a nivel básico), y la programación orientada a objetos.

Finalmente, el autor señala una notable carencia en la mayoría de los planes en relación a los temas de depuración y pruebas de software; y esto a pesar de que los descriptores de la troncalidad "Metodología y Tecnología de la Programación", en las directrices generales propias, incluyen la verificación y pruebas de software. Es más, también en las recomendaciones del CC2001 estos aspectos juegan un papel importante, señalando su necesidad para la formación de los futuros titulados.

4.3.2. La materia Algoritmos y Estructuras de Datos

Centrándonos en la materia objeto de concurso, hemos seleccionado ocho universidades españolas, para las cuales se han consultado sus planes de estudios de Ingeniería en Informática. Para la selección de estas universidades se ha utilizado un criterio de tamaño. En concreto, se han estudiado los planes de aquellos centros universitarios donde el número de plazas ofertadas para Ingeniería Informática, primer ciclo, durante el curso 2003/04 superaba las 200[5]. De esta forma, aunque el número

[5]Los datos han sido extraídos de la página web del Ministerio de Educación y Cultura: `http://www.mec.es/cgi-bin/search.pl?file=xense04&dir=%2F&numsearch=0&title=T%EDtulos+encontrados&type=1&casemach=1&log=1&subsearch=0&searched=Ingeniero+en+Inform%E1tica`

de centros es de a penas un 20 %, estos incluyen algo más del 50 % de las plazas. Las Universidades que cumplen el criterio, ordenadas alfabéticamente, son las siguientes:

- Universidad de Alicante
- Universidad Autónoma de Barcelona
- Universidad Autónoma de Madrid
- Universidad Complutense de Madrid
- Universidad de Málaga
- Universidad Politécnica de Madrid
- Universidad Politécnica de Cataluña
- Universidad de Sevilla

De entre las asignaturas de programación de los planes de cada una, se han analizado en detalle los programas de las asignaturas que, en principio, tienen una correspondencia directa con la materia "Ampliación de Algoritmos y Estructura de Datos". La información ha sido extraída de la página web de cada centro, y se refiere al curso 2003/04. Para cada Universidad, se indica el año en el que fueron aprobados los planes de II actualmente vigentes y el número de plazas ofertadas para el presente curso.

La leyenda usada en las tablas es la siguiente:

- **Tipo**: **Tr** - Troncal; **Ob** - Obligatoria; **Op** - Optativa
- **Dur.**: **A** - Anual; **C1** - Primer cuatrimestre; **C2** - Segundo cuatrimestre
- **Créd.(T+P)**: Créditos totales (Teóricos + Prácticos)

Universidad de Alicante
Escuela Politécnica Superior

Página web del Centro: `http://www.eps.ua.es/`
Año planes de II: 2001
Oferta de plazas: 210
Página web titulación:
`http://informatica.eps.ua.es/informacion/guiaInformatica/guiaInformatica.phtml`

Asignaturas de programación

CURSO	Asignatura	Tipo	Dur.	Créd.(T+P)
PRIMERO	Fundamentos de Programación I	Tr	C1	6 (3+3)
	Fundamentos de Programación II	Tr	C2	6 (3+3)
SEGUNDO	**Programación y Estructuras de Datos**	Tr	A	9 (4,5+4,5)
	Programación Orientada a Objetos	Ob	C1	4,5 (2,3+2,2)
	Diseño y Análisis de Algoritmos	Tr	C2	6 (3+3)
	Lenguajes y Paradigmas de Programación	Tr	C2	6 (3+3)
CUARTO	Algoritmia Avanzada	Ob	C1	4,5 (2,3+2,2)

Asignatura: Programación y Estructuras de Datos

Datos: Segundo curso; anual; 9 créditos (4,5T+4,5P)

Contenidos:

1. Introducción a los Tipos Abstractos de Datos: concepto, especificación y análisis
2. Los tipos lineales
3. El tipo árbol
4. El tipo conjunto
5. El tipo grafo

Bibliografía:

- G. Rodríguez y otros. *Ejercicios de programación creativos y recreativos en C++*. Prentice Hall, 2002.
- F.M. Carrano, J.H. Prichard. *Data Abstraction and Problem Solving with C++*. Addison Wesley, 2002.
- A. Drozdek. *Data Structures and Algorithms in C++*. Brooks/Cole, 2001.
- B. Stroustrup. *El lenguaje de programación C++*. Addison Wesley, 2002.
- W. Savitch. *Resolución de problemas con C++*. Prentice Hall, 2000.
- E. Horowitz, S. Sahni, D. Mehta. *Fundamentals of Data Structures in C++*. W.H. Freeman & Co., 1995.
- R. Peña. *Diseño de programas. Formalismo y abstracción*. Prentice Hall, 1998.
- H.M. Deitel, P.J. Deitel. *Cómo programar en C++*. Prentice Hall, 2003.

Lenguaje de programación: C++

Asignatura: Diseño y Análisis de Algoritmos

Datos: Segundo curso; segundo cuatrimestre; 6 créditos (3T+3P)

Contenidos:

1. Introducción al diseño y análisis de algoritmos
2. La eficiencia de los algoritmos
3. La corrección de los algoritmos
4. Diseño de algoritmos: La programación con esquemas
   a) Divide y Vencerás
   b) Programación Dinámica
   c) Algoritmos Voraces
   d) Algoritmos de Vuelta Atrás

Bibliografía:

- I. Parberry. *Lecture Notes on Algorithm Analysis and Computational Complexity*. Dept. of Computer Sciences. University of North Texas, 2001.
- G. Brassard, P. Bratley. *Fundamentos de Algoritmia*. Prentice Hall, 1997.
- I. Parberry, W. Gasarch. *Problems on Algorithms*. Dept. of Computer Sciences. University of North Texas, 2002.
- H.M. Deitel, P.J. Deitel. *Cómo programar en C++*. Prentice Hall, 2003.
- F. J. Ferri, J. F. Albert, G. Martín. *Introducció al anàlisi i disseny d´algorismes*. Universitat de Valencia, 1998.
- L. Joyanes. *Programación en C++. Algoritmos, estructuras de datos y objetos*. McGraw-Hill, 2000.

Lenguaje de programación: C++

Resumen: Este plan propone un enfoque mixto, más próximo al modelo imperativo primero, pero utilizando un lenguaje OO como C++ desde el principio. La introducción de C++ se plantea como una evolución de C, y es mantenido para todas las asignaturas de programación mostradas. En segundo curso, primer cuatrimestre, hay una asignatura específica de POO; en el segundo cuatrimestre hay otra de paradigmas de programación, donde se estudian el paradigma funcional, la recursión y la concurrencia. La enseñanza de AAED está separada en una asignatura anual de estructuras de datos y una cuatrimestral de algoritmos. En total la carga de ambas suma 15 créditos. Pero la importancia de la materia es aún mayor si tenemos en cuenta la asignatura obligatoria de cuarto curso "Algoritmia Avanzada", donde se profundiza en técnicas de diseño como programación dinámica, ramificación y poda, métodos numéricos y búsqueda de patrones.

Universidad Autónoma de Barcelona
Escuela Técnica Superior de Ingeniería

Página web del Centro: `http://einstein.uab.es/_c_webetse/`
Año planes de II: 2001
Oferta de plazas: 230
Página web titulación:
`http://einstein.uab.es/_c_webetse/webmasters/gestio_acade/estudis/iinform(pla2001).htm`

Asignaturas de programación

CURSO	Asignatura	Tipo	Dur.	Créd.(T+P)
PRIMERO	Algoritmos y Programación	Ob	C1	6 (3+3)
	Lenguajes de Programación	Ob	C1	4,5 (1,5+3)
	Estructuras de Datos	Ob	C2	6 (3+3)
SEGUNDO	**Grafos y Complejidad**	Ob	C1	4,5 (3+1,5)
	Teoría de la Programación	Ob	C2	6 (3+3)

Asignatura: Estructuras de Datos

Datos: Primer curso; segundo cuatrimestre; 6 créditos (3T+3P)

Contenidos:

1. Introducción. Tipos abstractos, clases y objetos
2. Estructuras lineales. Patrones, herencia, vectores, matrices, listas
3. Árboles: binarios, heaps, de búsqueda
4. Grafos: representación y operaciones básicas
5. Tablas hash

Bibliografía:

- E. Horowitz, S. Sahani, D. Mehta. *Fundamentals of data structures in C++*. Computer Science Press, 1995.
- G.L. Heileman. *Estructuras de Datos, Algoritmos y Programación Orientada a Objetos*. McGraw-Hill, 1998.
- B. Stroustrup. *El Lenguaje de Programación C++. 3er ed.* Addison-Wesley, 2002.
- L. Joyanes Aguilar. *Programación en C++. Algoritmos, estructuras de datos y objetos*. McGraw-Hill, 1999.
- R. Robson. *Using the STL. The C++ standard Template Library*. Springer, 1997.
- D. Knuth. *The art of computer programming: sorting and searching*. Addison-Wesley, 1976.

Lenguaje de programación: C++

Asignatura: Grafos y Complejidad

Datos: Segundo curso; primer cuatrimestre; 4,5 créditos (3T+1,5P)

Contenidos:

1. Introducción y fundamentos. Tipos, representaciones, recorridos
2. Árboles, caminos y flujos
3. Emparejamiento y planiraridad
4. Independencia, recubrimiento y coloración
5. Circuitos de Euler y circuitos de Hamilton

Bibliografía:

- J.M. Basart. *Grafs: fonaments i algorismes*. Manuals de la UAB, 13. Publicacions de la UAB, 1994.
- F. Comellas. *Matemàtica discreta*. Politext 26, Edicions UPC, 1996.
- A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

- J. Gimbert, et al. *Apropament a la teoria de grafs i als seus algorismes*. Edicions de la Universitat de Lleida, 1998.
- R.P. Grimaldi. *Matemáticas discreta y combinatoria*. Addison-Wesley Iberoamericana, 1989.

Lenguaje de programación: –

Asignatura: Teoría de la Programación

Datos: Segundo curso; segundo cuatrimestre; 6 créditos (3T+3P)

Contenidos:

1. Conceptos básicos
2. Abstracción: polimorfismo y orden superior
3. Tipos de datos: árboles, grafos, técnicas de backtracking y branch and bound
4. Tipos abstractos de datos: especificación, implementación y parametrización
5. Inducción
6. Evaluación lazy
7. Algoritmos: divide y vencerás, programación dinámica y greedy

Bibliografía:

- C. Reade. *Elements of Functional Programming*. Addison-Wesley, 1989.
- R. Milner, M. Tofte, R. Harper. *The Definition of Standard ML*. The MIT Press, 1990.
- R. Harper. *Introduction to Standard ML*. LFCS Report 86-14, 1989.
- A. Aho, J. Hopcroft, J. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.
- L.C. Paulson. *ML for the working programmer*. Cambridge University Press, 1991.
- A. Wikstrom. *Functional Programming using Standard ML*. Prentice Hall, 1987.

Lenguaje de programación: Standard ML

Resumen: El modelo utilizado en este plan de estudios es un ejemplo de estrategia imperativo primero, pero con una introducción temprana de la orientación a objetos. En el primer cuatrimestre de primero se estudia el lenguaje C, y en el segundo se introduce la POO con C++. Este paradigma es tratado dentro de una asignatura de estructuras de datos, donde se incluyen los temas comunes de estructuras arbóreas, grafos y tablas de dispersión. Existe una asignatura de segundo dedicada específicamente a los problemas y algoritmos sobre grafos, con una considerable carga teórica. Esta incluye algunos contenidos no muy habituales en otros planes como el emparejamiento y la planaridad de grafos. Por otro lado, las técnicas de diseño de algoritmos son introducidas de forma separada, en una asignatura cuatrimestral de segundo. La particularidad de esta asignatura es la utilización de un lenguaje de programación funcional, como es Standard ML.

Universidad Autónoma de Madrid
Escuela Politécnica Superior

Página web del Centro: `http://www.ii.uam.es/`
Año planes de II: 1998
Oferta de plazas: 228
Página web titulación:
`http://www.ii.uam.es/esp/alumnos/planes_inf.html`

Asignaturas de programación

CURSO	Asignatura	Tipo	Dur.	Créd.
PRIMERO	Metodología y Tecnología de la Programación I	Tr	C1	7,5
	Estructura de Datos y de la Información I	Tr	C2	7,5
SEGUNDO	**Metodología y Tecnología de la Programación II**	Tr	C1	7,5
	Estructura de Datos y de la Información II	Tr	C2	7,5
TERCERO	**Análisis de Algoritmos**	Op	C1	7
	Programación Orientada a Objetos	Op	C2	7

Asignatura: Metodología y Tecnología de la Programación II

Datos: Segundo curso; primer cuatrimestre; 7,5 créditos

Contenidos:

1. Análisis de eficacia: medidas, herramientas matemáticas, casos peor, mejor y medio
2. Algoritmos básicos de ordenación: selección, burbuja, inserción y shell
3. Algoritmos avanzados de ordenación: mergesort, quicksort; desigualdades recurrentes
4. Árboles de decisión y cotas inferiores para algoritmos de ordenación; heapsort, radixsort
5. Algoritmos básicos de búsqueda: búsqueda lineal, binaria; TAD diccionario, árboles AVL
6. Tablas hash: funciones hash, resolución de colisiones, encadenamiento, direccionamiento abierto

Bibliografía:

- M.A. Weiss. *Data structures and algorithm analysis in C.* Benjamin Cummings, 1993.
- T. Cormen, C. Leiserson, R. Rivest. *Introduction to algorithms.* The MIT Press-McGraw Hill, 1990.
- S. Baase, A. Van Gelder. *Computer algorithms. Introduction to design and analysis.* Addison-Wesley, 2000.
- A. Aho, J. Hopcroft, J. Ullman. *The design and analysis of algorithms.* Addison-Wesley, 1974.
- B. Kernighan, D. Ritchie. *The C programming language, 2nd ed.* Prentice Hall, 1991.
- B. Kernighan, Pike. *The practice of programming.* Addison-Wesley, 1999.

Lenguaje de programación: C

Asignatura: Estructura de Datos y de la Información II

Datos: Segundo curso; segundo cuatrimestre; 7,5 créditos

Contenidos: (No disponible, se muestran los temas del programa de prácticas)

1. Heaps y colas de prioridad
2. Grafos: representación y problemas
3. Ficheros e índices
4. Compresión de la información
5. Árboles B

Bibliografía:

- T. Cormen, C. Leiserson, R. Rivest. *Introduction to algorithms*. The MIT Press-McGraw Hill, 1990.
- M.J. Folk. *File structures, an object-oriented approach with C++*. 1998.
- M.J. Folk. *Estructuras de archivos un conjunto de herramientas conceptuales*. 1992.

Lenguaje de programación: C/C++

Asignatura: Análisis de Algoritmos

Datos: Tercer curso; primer cuatrimestre; 7 créditos; optativa

Contenidos:

1. Revisión de algoritmos elementales en grafos
2. Árboles abarcadores mínimos: Prim y Kruskal, TAD Conjunto Disjunto
3. Grafos de tareas: Grafos dirigidos acíclicos
4. Aplicaciones de búsqueda en profundidad: grafos biconexos, circuitos eulerianos, hamiltonianos y problema del viajante
5. Flujos en grafos: algoritmo de Ford-Fulkerson, cortes mínimos y flujos máximos, algoritmo de Edmonds-Karp
6. Programación dinámica: criptografía, orden óptimo del producto de matrices, ABB óptimos, Floyd, búsquedas aproximadas
7. Recursividad: problema de selección, multiplicación de matrices, FFT, distancias mínimas entre conjuntos
8. Algoritmos codiciosos: secuenciación de trabajos, codificación Hufman, relación con programación dinámica

Bibliografía:

- T. Cormen, C. Leiserson, R. Rivest. *Introduction to algorithms*. The MIT Press-McGraw Hill, 1990.
- M.A. Weiss. *Data structures and algorithm analysis in C*. Benjamin Cummings, 1993.
- R. Sedgewick. *Algorithms in C++*. Addison-Wesley, 1988.
- A. Aho, J. Hopcroft, J. Ullman. *The design and analysis of algorithms*. Addison-Wesley, 1974.
- A. Aho, J. Hopcroft, J. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

Lenguaje de programación: C

Resumen: Este plan de estudios es uno de los más centrados en el paradigma de programación imperativo, que es estudiado en primero y en segundo, usando los lenguajes Pascal y C. La programación OO no es estudiada en detalle hasta en el segundo cuatrimestre de tercer curso, y además en una asignatura optativa. En cuanto a la materia AAED, hay una separación en varias asignaturas de segundo y tercero. En la primera de estas se estudia análisis de algoritmos, algoritmos de ordenación y algunas estructuras básicas. En otra asignatura de segundo se estudian grafos, árboles –incluyendo árboles B y su representación en disco– y técnicas de compresión. Las técnicas generales de diseño de algoritmos son dejadas para la asignatura de tercer curso "Análisis de Algoritmos", pero con la particularidad de que se trata de una asignatura optativa. Esto hace que el estudio de temas como la programación dinámica, los algoritmos voraces (codiciosos) y algunos algoritmos tradicionales sobre grafos sean opcionales en esta titulación.

Universidad Complutense de Madrid
Facultad de Informática

Página web del Centro: `http://www.fdi.ucm.es/`
Año planes de II: 1998
Oferta de plazas: 220
Página web titulación:
`http://www.fdi.ucm.es/datos/Prog_asignatura.asp`

Asignaturas de programación

CURSO	Asignatura	Tipo	Dur.	Créd.(T+P)
PRIMERO	Introducción a la Programación	Ob	A	9 (9+0)
	Laboratorio de Programación 1	Ob	C2	4,5 (1,5+3)
SEGUNDO	**Estructura de Datos y de la Información**	Tr	A	15 (15+0)
	Laboratorio de Programación 2	Ob	A	9 (3+6)
	Programación Orientada a Objetos	Tr	C1	4,5 (4,5+0)
TERCERO	**Metodología y Tecnología de la Programación**	Tr	A	12 (12+0)
	Laboratorio de Programación 3	Ob	C1	6 (1,5+4,5)

Asignatura: Estructura de Datos y de la Información

Datos: Segundo curso; anual; 15 créditos (15T+0P)

Contenidos:

1. Eficiencia de programas iterativos y recursivos
2. Diseño de programas iterativos
3. Diseño de programas recursivos
4. Tipos abstractos de datos
5. Tipos de datos lineales
6. Tipos de datos arborescentes
7. Tipos de datos funcionales
8. Tipos de datos relacionales

Bibliografía:

- R. Peña. *Diseño de programas. Formalismo y abstracción. Segunda edición*. Prentice Hall, 1998.
- A. Kaldewaij. *The Derivation of algorithms*. Prentice Hall, 1990.
- E. Horowitz, S. Sahni, D. Mehta. *Fundamentals of Data Structures in C++*. W.H. Freeman & Co., 1995.
- N. Martí, Y. Ortega, J.A. Verdejo. *Estructuras de datos y métodos algorítmicos: Ejercicios resueltos*. Colección Prentice Practica, 2003.

Lenguaje de programación: –

Asignatura: Laboratorio de Programación 2

Datos: Segundo curso; anual; 9 créditos (3T+6P)

Contenidos:

1. Introducción a C++ Builder
2. Programación modular
3. Gestión de memoria dinámica
4. Programación orientada a objetos
5. Herencia y polimorfismo
6. Tratamiento de excepciones
7. Tipos abstractos de datos

Bibliografía:

- F. Charte. *Programación en C++ Builder 5*. Anaya Multimedia, 2000.
- H.M. Deitel, P.J. Deitel. *Cómo programar en C++*. Prentice Hall, 2003.
- R. Lafore. *Object-Oriented Programming in C++*. The Waite Group, 1999.

- E. Horowitz, S. Sahni, D. Mehta. *Fundamentals of Data Structures in C++*. W.H. Freeman & Co., 1995.

Lenguaje de programación: C++

Asignatura: Metodología y Tecnología de la Programación

Datos: Tercer curso; anual; 12 créditos (12T+0P)

Contenidos:

1. Transformación de algoritmos recursivos a iterativos
2. Complejidad avanzada de algoritmos: análisis en el caso medio y análisis amortizado
3. Divide y vencerás
4. Programación dinámica
5. Método voraz
6. Vuelta atrás
7. Ramificación y poda
8. Introducción a la NP-completitud
9. Algoritmos de aproximación
10. Otras técnicas: precondicionamiento, transformación de dominio, algoritmos probabilistas

Bibliografía:

- R. Neapolitan, K. Naimipour. *Foundations of algorithms; 2ª o 3ª edición*. Jones and Bartlett Publishers, 1998 o 2003.
- E. Horowitz, S. Shani, S. Rajasekaran. *Computer algorithms; 3ª edición*. Computer Science Press, 1998.
- G. Brassard, P. Bratley. *Fundamentos de Algoritmia; 1ª edición*. Prentice Hall, 1997.
- N. Martí, Y. Ortega, A. Verdejo. *Estructuras de datos y métodos algorítmicos: ejercicios resueltos; 1ª edición*. Prentice Hall, 2003.

Lenguaje de programación: –

Resumen: Nuevamente nos encontramos ante unos planes de estudios que siguen el enfoque imperativo primero, al estilo tradicional. La programación OO en introducida en el primer cuatrimestre de segundo curso, en una asignatura específica de POO, mientras que la algorítmica es desarrollada en el tercer curso. Resulta destacable el número de lenguajes de programación que se utilizan en las asignaturas introductorias de programación: Pascal en primero, C++ en segundo, y Java en POO (asignatura de segundo) y en tercero. Los planes marcan una separación muy clara entre las asignaturas de carácter teórico y las prácticas. Hay también una separación entre estructuras de datos, que son estudiadas en segundo, y análisis y diseño de algoritmos, que tiene lugar en tercero. Debemos mencionar igualmente la elevada carga de los contenidos propios de AAED, que suman 27 créditos

teóricos y 15 créditos en las asignaturas de laboratorio. Esto permite la introducción de técnicas poco frecuentes en otros planes, como los algoritmos de aproximación, los probabilistas y la NP-completitud.

Universidad de Málaga
Escuela Técnica Superior de Ingeniería Informática

Página web del Centro: `http://www.informatica.uma.es/ETSIIPub/`
Año planes de II: 2001
Oferta de plazas: 226
Página web titulación:
`http://www.informatica.uma.es/ETSIIPub/PlanEstudio/AsignaturasPlanPpal.aspx`

Asignaturas de programación

CURSO	Asignatura	Tipo	Dur.	Créd.(T+P)
PRIMERO	Elementos de Programación	Ob	C1	7 (4,5+3)
	Metodología de la Programación	Tr	C2	6 (3+3)
	Laboratorio de Programación	Tr	C2	4,5 (0+4,5)
SEGUNDO	**Tipos Abstractos de Datos**	Tr	C1	6 (3+3)
	Análisis y Diseño de Algoritmos	Tr	C2	6 (3+3)
	Laboratorio de Tecnología de Objetos	Ob	C2	6 (1,5+4,5)

Asignatura: Tipos Abstractos de Datos

Datos: Segundo curso; anual; 6 créditos (3T+3P)

Contenidos:

1. Diseño modular de sistemas de software
2. Especificación y verificación de programas imperativos. Pre y post-condiciones
3. Tipos abstractos de datos. Especificación algebraica de tipos abstractos
4. Tipos abstractos de datos lineales: listas, pilas, colas y otros
5. Tipos abstractos de datos arbóreos: ABB, montículos, AVL, n-arios, 2-3 y B
6. Tipos abstractos cociente: conjunto, bolsa, tabla, función, grafo
7. Diseño e implementación de aplicaciones basadas en TAD

Bibliografía:

- A. Aho, J. Hopcroft, J. Ullman. *Estructuras de Datos y Algoritmos*. Addison-Wesley, 1988.
- M. Clavel, et al. *A Maude Tutorial*. Disponible en: `http://maude.csl.sri.com`
- O.J. Dahl. *Verifiable Programming*. Prentice-Hall International.
- X. Franch. *Estructuras de datos: especificación diseño e implementación*. Ediciones UPC, 1994.
- R. Peña. *Diseño de programas. Formalismo y abstracción. Segunda edición*. Prentice Hall, 1998.

Lenguaje de programación: Maude

Asignatura: Análisis y Diseño de Algoritmos

Datos: Segundo curso, segundo cuatrimestre, 6 créditos (3T+3P)

Contenidos:

1. Complejidad, eficiencia, notaciones asintóticas, calculo del tiempo de ejecución en algoritmos iterativos y recursivos
2. Ordenación y búsqueda
3. Divide y vencerás
4. Avance rápido
5. Programación dinámica
6. Vuelta atrás
7. Ramificación y poda
8. Clasificación de problemas

Bibliografía:

- A. Aho, J. Hopcroft, J. Ullman. *Estructuras de Datos y Algoritmos*. Addison-Wesley, 1988.
- M. Azmoodech. *Abstract data types and algorithms*. McMillan Education Ltd., 1988.
- G. Brassard, P. Bratley. *Fundamentos de Algoritmia, 1ª edición*. Prentice Hall, 1997.
- R. Guerequeta, A. Vallecillo. *Técnicas de diseño de algoritmos*. SPICUM. Universidad de Málaga, 1998.
- E. Horowitz, S. Sahni. *Fundamentals of computer Algorithms*. Computer Science Press, 1978.
- R. Sedgewick. *An introduction to Analysis of Algorithms*. Addison-Wesley, 1997.

Lenguaje de programación: –

Resumen: Estos planes de estudios presentan cierta similitud con los ofertados en la FIUM, en cuanto a la distribución de materias y contenidos de las asignaturas. El enfoque utilizado es el imperativo primero, si bien el lenguaje usado es C++, aunque aplicando los principios de la programación modular. La POO es introducida en el segundo cuatrimestre de segundo curso, en una asignatura específica. En cuanto a los contenidos propios de AAED, hay una separación explícita entre dos asignaturas: una con la parte de estructuras, en el primer cuatrimestre, y otra con lo referido al análisis y diseño de algoritmos en el segundo. La primera, llamada "Tipos Abstractos de Datos", hace un énfasis muy especial en las especificaciones formales algebraicas, usando el lenguaje axiomático Maude. Por su parte, la asignatura de algorítmica incluye en primer lugar el análisis de algoritmos y después trata un conjunto de técnicas generales de diseño de algoritmos, que se encuentran generalmente en muchos otros temarios.

Universidad Politécnica de Cataluña
Facultad de Informática

Página web del Centro: `http://www.fib.upc.es/`
Año planes de II: No encontrado
Oferta de plazas: 375
Página web titulación:
`http://www.fib.upc.es/ca/Estudis/guiadocent.html`

Asignaturas de programación

CURSO	Asignatura	Tipo	Dur.	Créd.(T+P)
PRIMERO	Iniciación a la Programación	Tr	C1	9 (4,5+4,5)
	Programación Metódica	Tr	C2	7,5 (4,5+3)
SEGUNDO	**Estructuras de Datos y Algoritmos**	Tr	C1	9 (4,5+4,5)
	Esquemas Algorítmicos	Ob	C2	6 (3+3)
	Proyecto de Programación	Ob	C2	6 (3+3)

Asignatura: Estructuras de Datos y Algoritmos

Datos: Segundo curso; primer cuatrimestre; 9 créditos (4,5T+4,5P)

Contenidos:

1. Conceptos básicos de TAD: concepto, especificación, clases y objetos
2. Análisis de eficiencia de algoritmos: medidas de tiempo y espacio, casos peor y promedio, análisis iterativo y recursivo, notaciones asintóticas
3. Estructuras lineales: listas, pilas, colas
4. Árboles: concepto, tipos y aplicaciones
5. Tablas asociativas. Implementación con listas, listas ordenadas, vectores, ABB, AVL, tablas de dispersión y tries
6. Colas de prioridad
7. Particiones
8. Grafos: concepto, representaciones, recorridos, algoritmos de Dijkstra y Kruskal

Bibliografía:

- M.A. Weiss. *Data Structures and Algorithm Analysis in C++ (2nd edition)*. Addison-Wesley, 1999.
- A.V. Aho, J.D. Ullman. *Foundations of Computer Science (C edition)*. W.H. Freeman & Co., 1995.
- B. Stroustrup. *The C++ Programming Language (3rd edition)*. Addison-Wesley, 1997.

Lenguaje de programación: C/C++

Asignatura: Esquemas Algorítmicos

Datos: Segundo curso; segundo cuatrimestre; 6 créditos (3T+3P)

Contenidos:

1. Fundamentos matemáticos: series, recurrencias, probabilidad
2. Clasificación: quicksort, radix, bucket, selección
3. Estructuras de datos: tablas hash, árboles de búsqueda, árboles rojos-negros, heaps, conjuntos disjuntos
4. Algoritmos probabilistas: primalidad, distancia mínima

Bibliografía:

- T. Cormen, C. Leiserson, R. Rivest. *Introduction to algorithms*. The MIT Press-McGraw Hill, 1990.
- R. Sedgewick. *Algorithms in C*. Addison-Wesley, 1996.
- Diaz, Serna, Spirakis, Toran. *Paradigms for fast parallel approximability*. Cambridge University Press, 1997.

Lenguaje de programación: C/C++

Resumen: El enfoque utilizado en estos planes de estudios es bastante próximo al orientado a objetos primero. Desde primer curso se empieza enseñando el lenguaje OO Java, inicialmente desde una perspectiva de programación modular. Los conceptos de la POO aparecen en el segundo cuatrimestre del primer año, aunque la profundización es fundamentalmente en la parte práctica de la asignatura. En cuanto a la materia propia de AAED, se organiza en torno a dos asignaturas cuatrimestrales de segundo. En primer lugar se estudian las estructuras de datos, incluyendo un tema de análisis de algoritmos iterativos y recursivos. En el segundo cuatrimestre se estudian esquemas algorítmicos, con especial atención a los problemas de clasificación, y a los algoritmos probabilistas. Destacamos la presencia de la asignatura "Proyecto de Programación", en el segundo cuatrimestre de segundo, como una culminación de las asignaturas de introducción a la programación, y que pone énfasis en las etapas del proceso de desarrollo: especificación, análisis, diseño, prueba, etc.

Universidad Politécnica de Madrid
Facultad de Informática

Página web del Centro: `http://www.fi.upm.es/`
Año planes de II: 1996
Oferta de plazas: 375
Página web titulación:
`http://www.fi.upm.es/estudios/segundo_ciclo/Programas_Plan96.pdf`

Asignaturas de programación

CURSO	Asignatura	Tipo	Dur.	Créd.
PRIMERO	Metodología de la Programación	Tr	A	15
SEGUNDO	**Estructuras de Datos I**	Tr	C1	6
	Estructuras de Datos II	Tr	C2	7,5
	Desarrollo Sistemático de Programas	Ob	C2	4,5
TERCERO	Modelo de Desarrollo de Programas	Ob	C2	4,5
	Teoría de Grafos	Op	C2	4,5

Asignatura: Estructuras de Datos I

Datos: Segundo curso; primer cuatrimestre; 6 créditos

Contenidos:

1. Motivación: realización de proyectos grandes
2. Programación modular: encapsulamiento, interfaz/implementación, TAD
3. Estudio y uso de algunos TAD: pilas, colas, listas, secuencias, árboles, conjuntos
4. Especificación formal de un TAD
5. Implementación de TAD
6. Implementación eficiente de TAD, complejidad
7. Diseño de TAD: selección de operaciones y del dominio
8. Especificaciones algebraicas y polimorfismo

Bibliografía:

- N. Dale, H.M. Walker. *Abstract Data Types: Specifications, Implementations and Applications*. D. C. Heath and Company, 1996.
- M.B. Feldman. *Software Construction and Data Structures with Ada 95*. Addison Wesley, 1997.
- N.Dale, S.C. Lilly, J. McCormick. *Ada plus data structures. An object oriented approach*. D. C. Health and Company, 1996.
- S. Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley, 1996.
- J.T. Latham, V.J. Bush, I.D. Cottam. *The Programming Process. An Introduction Using VDM and Pascal, 1st edition*. Addison-Wesley, 1990.

Lenguaje de programación: Ada 95, Haskell 98

Asignatura: Estructuras de Datos II

Datos: Segundo curso; segundo cuatrimestre; 7,5 créditos

Contenidos:

1. Estructuras de datos complejas. Generalidades

2. Estructuras de datos arborescentes avanzadas: árboles ordenados, montículos, tries, AVL, árboles B y enhebrados
3. Estructuras de datos dispersas: conjuntos, matrices
4. Estructuras avanzadas de tabla: tablas arborescentes, hash
5. Grafos: representación, algoritmos sobre grafos: recorridos, recubrimiento, búsquedas
6. Gestión dinámica de memoria
7. Persistencia
8. Ficheros: modos de organización y modos de acceso
9. Implementación de ficheros
10. Bases de datos

Bibliografía:

- N. Dale, H. Walker. *Abstract Data Types*. D.C. Heath and Company, 1996.
- M. J. Folk, B. Zoellick. *File Structures*. Addison-Wesley, 1992.
- M.A. Weiss. *Data Structures and Algorithm Analysis*. Benjamin/Cummings, 1995.
- E. Horowitz, S. Sahni. *Fundamentals of Data Structures in Pascal*. Computer Science Press, 1990.
- T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- N. Wirth. *Algorithms + Data Structures = Programs*. Prentice Hall, 1976.
- D.E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1998.

Lenguaje de programación: Ada 95

Asignatura: Desarrollo Sistemático de Programas

Datos: Segundo curso; segundo cuatrimestre; 4,5 créditos

Contenidos:

1. Programas con orden superior: especificaciones, soluciones, programas con orden superior
2. Técnicas de diseño de algoritmos
   a) Divide y vencerás
   b) Retroceso
   c) Soluciones voraces
   d) Programación dinámica
3. Transformación de programas: técnicas de inmersión, recursividad final, transformación

Bibliografía:

- R. Bird. *Introduction to Functional Programming using Haskell*. Prentice Hall, 1998.
- S. Thompson. *The Craft of Functional Programming*. Addison Wesley, 1999.
- J. Barnes. *Programming in Ada-95*. Addison Wesley, 1996.

- M.B. Feldman, E.B. Koffman. *Ada-95: Problem Solving and Program Design*. Addison-Wesley, 1995.
- G. Brassard, P. Bratley. *Fundamentos de Algoritmia*. Prentice Hall, 1997.
- A.V. Aho, J.E. Hopcroft, J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- E. Horowitz, S. Sahni. *Fundamentals of Computer Algorithms*. Pitman, 1978.
- T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- R. Peña. *Diseño de programas: Formalismo y abstracción. Prentice Hall*, 1994.
- J. Balcázar. *Programación sistemática*. McGraw Hill, 1995.
- ... (Incluye más referencias, que han sido omitidas por razón de espacio).

Lenguaje de programación: Ada 95 y Haskell 98

Asignatura: Teoría de Grafos

Datos: Tercero curso; segundo cuatrimestre; 4,5 créditos; optativa

Contenidos:

1. Nociones básicas: tipos de grafos, isomorfismo, representación
2. Árboles, árboles generadores, árboles generadores mínimos, búsquedas, caminos y distancias, algoritmos de Dijkstra, Ford y Floyd
3. Redes de transporte. Flujos en redes
4. Emparejamientos en grafos bipartidos. Emparejamientos y flujos
5. Grafos eulerianos. Problema del cartero. Digrafos eulerianos: digrafos de De Bruijn
6. Grafos hamiltonianos. Problema del viajante: algoritmos aproximados
7. Planaridad. Algoritmos de detección de la planaridad. Parámetros de planaridad
8. Coloración de grafos. Algoritmos de coloración. Coloración de grafos planos
9. Complejidad. Problemas NP en grafos
10. Visualización y trazado de grafos

Lenguaje de programación: –

Resumen: Este plan de estudios, cuya estructura es similar en las ingenierías técnicas, plantea un enfoque original y poco habitual de la enseñanza de la programación; es lo que el informe CC2001 denomina el acercamiento funcional primero. Realmente se sigue un enfoque mixto, trabajando desde el principio los paradigmas funcional e imperativo, con los lenguajes Haskell y Ada, respectivamente. Esto conduce a la necesidad de retrasar la enseñanza del paradigma orientado a objetos hasta tercer curso. En cuanto a la organización en asignaturas, mientras que en primero existe una asignatura anual, en segundo hay tres asignaturas cuatrimestrales relacionadas con AAED, en las que se siguen usando los lenguajes Ada y Haskell. En una de ellas se enseñan tipos y estructuras de datos básicos,

y análisis de eficiencia; en otra se incluyen estructuras avanzadas y algunos conceptos de bases de datos; y finalmente hay otra asignatura de técnicas generales de diseño de algoritmos. Por otro lado, los contenidos de grafos aparecen en una asignatura optativa de tercer curso. En total las asignaturas correspondientes a AAED sumarían 16,5 créditos.

Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática

Página web del Centro: `http://www.eii.us.es/`
Año planes de II: 1997
Oferta de plazas: 275
Página web titulación:
`http://www.eii.us.es/ingenieria.php`

Asignaturas de programación

CURSO	Asignatura	Tipo	Dur.	Créd.(T+P)
PRIMERO	Introducción a la Programación I	Tr	C1	7,5 (4,5+3)
	Introducción a la Programación II	Tr	C2	7,5 (4,5+3)
PRIMERO	**Análisis y Diseño de Algoritmos**	Ob	C1	7,5 (4,5+3)
	Estructuras de Datos y Algoritmos	Tr	C2	7,5 (4,5+3)
TERCERO	Teoría de Grafos	Op	C2	6 (3+3)

Asignatura: Análisis y Diseño de Algoritmos

Datos: Segundo curso; primer cuatrimestre; 7,5 créditos (4,5T+3P)

Contenidos:

1. Análisis de algoritmos, estudio de complejidad, medidas asintóticas, órdenes de complejidad, clases de problemas
2. Patrones de diseño: introducción, tipos de patrones, catálogo de patrones
3. Técnicas de diseño de algoritmos
   a) Algoritmos voraces
   b) Divide y vencerás
   c) Programación dinámica
   d) Algoritmos de vuelta atrás
   e) Ramificación y acotación

Bibliografía:

- A. Aho, J. Hopcroft, J. Ullman. *Estructuras de Datos y Algoritmos*. Addison-Wesley, 1988.
- K. Arnold, et al. *El Lenguaje de Programación Java*. Addison-Wesley, 2001.
- D. Arnow, G. Weiss. *Introducción a la Programación con Java. Un enfoque Orientado a Objetos*. Addison-Wesley, 2000.

- G. Brassard, P. Bratley. *Fundamentos de algoritmia*. Prentice Hall, 1997.
- T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- E. Gamma, et al. *Design Patterns. Elements of Reusable Object Oriented Software*. Addison-Wesley, 1995.
- M. Grand. *Patterns in Java, Volume 1*. John Wiley & Sons, 1998.
- R. Peña Marí. *Diseño de programas. Formalismo y abstracción*. Prentice Hall, 1998.
- M.A. Weiss. *Data Structures and Algorithm Analysis*. Benjamin/Cummings, 1995.
- N. Wirth. *Algoritmos y Estructuras de Datos*. Prentice Hall, 1987.

Lenguaje de programación: Java

Asignatura: Estructuras de Datos y Algoritmos

Datos: Segundo curso; segundo cuatrimestre; 7,5 créditos (4,5T+3P)

Contenidos:

1. Introducción: abstracción de datos, orientación a objetos, iteradores
2. Tipos Abstractos de Datos. Especificación algebraica de TAD. Implementación de TAD
3. Secuencias. Esquemas de recorrido. Pilas. Colas. Listas. Aplicaciones
4. Árboles. Esquemas de recorrido. Árboles binarios, de búsqueda. Montículos. Árboles multicamino. Árboles B
5. Relaciones y conjuntos. Relaciones. Transformación de claves. Multiconjuntos
6. Grafos. Esquemas de recorrido. Grafos dirigidos. Grafos no dirigidos

Bibliografía:

- A. Aho, J. Hopcroft, J. Ullman. *Estructuras de Datos y Algoritmos*. Addison-Wesley, 1988.
- K. Arnold, et al. *El Lenguaje de Programación Java*. Addison-Wesley, 2001.
- D. Arnow, G. Weiss. *Introducción a la Programación con Java. Un enfoque Orientado a Objetos*. Addison-Wesley, 2000.
- E. Horowitz, S. Sahni. *Fundamentals of Data Structures in Pascal*. Computer Science, 1994.
- L. Joyanes, I. Zahonero. *Estructuras de Datos. Algoritmos, abstracción y objetos*. McGraw-Hill, 1998.
- R. Peña Marí. *Diseño de programas. Formalismo y abstracción*. Prentice Hall, 1998.
- M.A. Weiss. *Data Structures and Algorithm Analysis*. Benjamin/Cummings, 1995.
- N. Wirth. *Algoritmos y Estructuras de Datos*. Prentice Hall, 1987.

Lenguaje de programación: Java

Resumen: El enfoque usado en el diseño de este plan de estudios es similar al denominado objetos primero. La secuencia introductoria empieza presentando la programación imperativa en C en el primer cuatrimestre, pasando en el segundo a la programación OO en C++. El paradigma OO es analizado a fondo en la asignatura correspondiente. En cuanto al segundo curso, existen dos asignaturas cuatrimestrales, la primera dedicada a la parte de algoritmos y la segunda a estructuras de datos. Ambas utilizan el lenguaje de programación Java. Es bastante peculiar la organización de la asignatura "Análisis y Diseño de Algoritmos", que incluye entre sus contenidos un tema de patrones de diseño –algo muy relacionado con el diseño de software OO–. Por otro lado, en la asignatura "Estructuras de Datos y Algoritmos" se presentan las principales estructuras de datos, también desde una clara perspectiva OO. Entre ambas asignaturas suman 15 créditos. Señalamos también la inclusión de una asignatura optativa en tercer curso dedicada a teoría de grafos.

4.3.3. Conclusiones del análisis de planes de estudios

A modo de síntesis, vamos a establecer las principales conclusiones que se pueden extraer tras el análisis de los planes de estudios de Ingeniero en Informática en las universidades españolas. Podemos estructurar las conclusiones en torno a los siguientes puntos:

- **Contenidos.** Existe un consenso bastante amplio y definido sobre los contenidos que deben ser estudiados en las asignaturas de la secuencia introductoria de programación. Este consenso incluye los tipos de datos fundamentales, las técnicas eficientes de representación de datos en memoria, y las técnicas generales de diseño de algoritmos, que son los puntos que aparecen normalmente en la mayoría de los libros de algoritmos y estructuras de datos. Sin embargo, la organización de estos contenidos en asignaturas, su distribución temporal y los matices que se aplican en la enseñanza de los mismos, son particulares y exclusivos de cada centro. Es prácticamente imposible encontrar un programa que pueda ser aplicado directamente en una asignatura de otra titulación, ya que o bien tendrá distinta carga crediticia, o se encontrará en otro lugar del plan de estudios, o las asignaturas previas o posteriores crearán un contexto bien distinto. Por lo tanto, se deben identificar los contenidos propios de la materia, y seleccionar aquellos que no sean cubiertos suficientemente por otras asignaturas del plan.

- **Paradigma de programación.** En la actualidad, las universidades españolas siguen utilizando mayoritariamente el paradigma imperativo en el primer contacto de los alumnos con la programación. Sin embargo, hay una tendencia cada vez más notable a una introducción temprana del paradigma orientado a objetos. El rango de "colocaciones" de la POO varía entre el segundo cuatrimestre de primer curso y el segundo cuatrimestre de tercero. No obstante, lo más habitual es la presentación de los fundamentos de OO en segundo curso. Por otro lado, no hay acuerdo sobre cómo estructurar la enseñanza de este paradigma. Mientras algunos centros incluyen una asignatura específica de POO, otros adoptan la estrategia de mezclarla con la enseñanza de estructuras de datos y algoritmos. La segunda estrategia suele ser más habitual entre las titulaciones que proponen una enseñanza temprana de la orientación a objetos.

- **Organización temporal.** La mayoría de los centros analizados opta por repartir los contenidos propios de AAED entre varias asignaturas, en algunos casos desligando también la parte teórica y la práctica. En la mayoría de los casos, además, las asignaturas de estructuras de datos se sitúan antes que las de algoritmos. Curiosamente, en los planes de la FIUM han coexistido

tres situaciones distintas: los planes antiguos de las Ingenierías Técnicas separaban algoritmos y estructuras de datos en dos asignaturas distintas; los planes antiguos de la Ingeniería superior separaban la parte teórica y la práctica; y los planes nuevos de las tres titulaciones juntan en una asignatura anual todos los contenidos teóricos y prácticos de algoritmos y estructuras de datos. Ya analizamos en el apartado 2.2.3 las numerosas ventajas que, desde nuestro punto de vista, supone la organización en una sola asignatura[6]. La situación de las otras universidades españolas sugiere que los contenidos de estructuras de datos sean planificados antes que los de algoritmos.

- **Carga de créditos.** Si consideramos las asignaturas de los otros centros universitarios cuyos contenidos corresponden con los de AAED, la carga crediticia media por titulación se sitúa por encima de los 16 créditos. Es difícil establecer un cálculo preciso –debido a los aspectos adicionales que son añadidos en esas asignaturas–, pero en cualquier caso la carga total de los contenidos más claramente relacionados con AAED está casi siempre en torno o por encima de los 15 créditos. Frente a esto, la asignatura AED del plan de 2002 de la FIUM tiene asignados únicamente 12 créditos (6 teóricos y 6 prácticos). Inevitablemente, se deberán descartar los temas que no quepan dentro de esta menor asignación de carga[7].

- **Lenguaje de programación.** En lo que se refiere a la parte práctica de la materia AAED, la elección del lenguaje de programación está bastante clara: de las 8 titulaciones estudiadas, 6 utilizan C++, 1 usa Ada y 1 Java. Los datos son bastante reveladores, aunque debemos tener en cuenta que el estudio no es muy representativo en relación al número de centros existentes. En la mayoría de los casos, C++ es presentado como una evolución natural del lenguaje C, permitiendo enseñar conceptos propios de POO en un entorno, en principio, imperativo.

En definitiva, la situación actual en los planes de otras universidades españolas presenta una gran diversidad en la forma, pero una notable serie de similitudes en el fondo. Estas similitudes pueden provenir no sólo de las tendencias más extendidas, sino también de los elementos que constituyen los fundamentos de la materia, de cuya importancia nadie discute. Saber identificarlas es esencial para contrastar en qué grado y sentido los planes y programas actuales deben ser revisados.

4.4. Propuesta de la materia

Una vez expuesta toda la información relevante para la planificación de la asignatura "Ampliación de Algoritmos y Estructura de Datos", afrontamos en esta sección la selección de los contenidos, su organización en actividades teóricas y prácticas, y la distribución temporal de la materia a lo largo del curso. Entendemos que los contenidos *"no son un fin en sí mismos, sino un medio [para alcanzar los objetivos]"*, [Ortín'02]. Por lo tanto, el proceso de selección se basa en criterios de adecuación a los objetivos, representatividad en la materia y encaje dentro del contexto de desarrollo. Empezando por lo más general, justificamos en primer lugar las decisiones más relevantes que se han tomado en el diseño de la asignatura. Después presentamos la organización de la materia teórica

[6]Entre las que podemos mencionar: mejor coordinación de la acción docente, posibilidad de seguir el aprendizaje de los alumnos a lo largo del curso, reparto más flexible de los contenidos, aplicación simultánea de estos conocimientos en las prácticas, etc.

[7]Podemos recordar, también a este respecto, que la aprobación de los planes de 2002 en la FIUM supuso un descenso de 2 créditos en la materia (AAED (8) + "Laboratorio de programación" (6) = 14 créditos). Este "redondeo a la baja" es el que coloca a la FIUM por debajo de la media nacional en esta materia.

y práctica, describiendo los aspectos más importantes a grandes rasgos. Finalmente se describe el material docente recomendado para la asignatura.

La descripción de los temas de teoría y las actividades de prácticas es realizada aquí de forma muy general. En los tres siguientes capítulos se desarrollan en detalle los contenidos de estas unidades, profundizando también en la temporización de los mismos.

En la tabla 4.2 se recuerda la carga de trabajo asignada para las asignaturas a las que está dirigida la propuesta de este proyecto docente, el número de semanas totales y la cantidad de horas por semana.

Planes de II, de 1996

Asignatura	Duración	Créditos		Semanas de clase	Horas/semana	
		Teoría	Prácticas		Teoría	Prácticas
Ampliación de Algoritmos y Estructura de Datos	1er semestre	6	2	20	3	1
Laboratorio de Programación	2º trimestre	0	6	10	0	6

Planes de II, de 2002

Asignatura	Duración	Créditos		Semanas de clase	Horas/semana	
		Teoría	Prácticas		Teoría	Prácticas
Algoritmos y Estructuras de Datos	Anual	6	6	30	2	2

Tabla 4.2: Carga de trabajo de las asignaturas relacionada con la materia AAED, en los planes antiguos y los nuevos.

Recordemos que el contenido teórico propuesto para la asignatura AAED –de plan antiguo– es el mismo que para AED –de plan nuevo–. La diferencia entre ambas se encuentra en la parte práctica. Haremos referencia a las prácticas tanto de una como de la otra.

4.4.1. Decisiones más relevantes

En este apartado exponemos y justificamos las principales decisiones sobre las que se basa la selección y organización de contenidos del presente proyecto docente. Las decisiones se refieren a los siguientes grandes aspectos: el paradigma de programación, los contenidos de la asignatura a gran escala, la organización de los contenidos en temas y bloques, y el lenguaje de programación usado en prácticas. De esta manera seguimos un proceso que, en esencia, nos lleva a "desmontar" el programa actual en los contenidos más elementales, revisarlos a fondo, actualizarlos de forma conveniente, y finalmente volver a "montarlos", es decir, estructurarlos en temas, bloques y partes. Trataremos de ser sintéticos, puesto que todas estas cuestiones han sido ya tratadas de una u otra forma en los apartados anteriores.

Paradigma de programación

La elección del enfoque paradigmático bajo el cual se desarrolla la docencia del ciclo introductorio de programación es uno de los aspectos más trascendentales; y al mismo tiempo uno de los más discutidos, *"que muy a menudo toma el carácter de guerra religiosa"*, [CC'2001]. La enseñanza en

estos cursos introductorios, entre los que podemos situar AAED, se enfrenta a una *"vorágine de rapidísima evolución tecnológica, donde las innovaciones se suceden de forma vertiginosa sin dar apenas tiempo a los entornos académicos a reaccionar"*, [Montoya'01]. Esto implica la necesidad de separar los fundamentos teóricos de las tendencias pasajeras, centrando la docencia en aquellos principios que constituyen la base más sólida y estable de la disciplina de la programación.

Después de más de una década de existencia, la programación orientada a objetos ha demostrado más que suficientemente no ser una de estas modas pasajeras, sino constituir un enfoque imprescindible y bien fundado, cuya influencia en el mundo académico y de la industria es cada vez mayor. De esta manera, el conocimiento del paradigma OO es reconocido mundialmente como un requisito necesario para cualquier ingeniero en informática[8]. Concretando, la cuestión que se plantea es: ¿debe ser introducida la POO en segundo, o se debe seguir usando el esquema imperativo/estructurado/modular? El problema no es sencillo, ya que plantea a su vez otras preguntas: ¿cuándo debe ser enseñada la POO?, ¿cómo debe ser introducida?

Programación orientada a objetos en primero. En el artículo [García'01] se plantea la conveniencia o no de introducir el paradigma OO desde el primer curso de la titulación, en un contexto de planes de estudios asimilables con los de la FIUM. Las conclusiones del autor no dejan lugar a dudas: (1) las aportaciones de la OO no sólo son irrelevantes para los objetivos del primer curso, sino que añaden una complejidad que puede llegar a desbordar a los alumnos; (2) puesto que los alumnos deben conocer tanto el paradigma procedural como el OO, es más lógico ir de lo elemental a lo complejo, de lo imperativo a lo OO. Coincidimos plenamente con estos planteamientos, si bien debemos mencionar los posibles argumentos con los que podrían ser rebatidos: (1) todo depende de los objetivos que se establezcan para el primer curso; (2) el enfoque imperativo primero se basa en la creencia errónea de los profesores de que los estudiantes deben aprender los enfoques en el mismo orden en el que ellos lo hicieron, [Meyer'94]. Sea como fuere, es evidente que ambos acercamientos han sido probados con éxito en distintas universidades, y ninguno puede atribuirse la cualidad de ser el mejor.

Programación orientada a objetos en segundo. Una cuestión bien distinta es la introducción de la POO en segundo curso, al cual llegan los alumnos sabiendo programar[9]. El artículo [García'01], antes citado, es más ambiguo en este sentido, indicando que *"tendría sentido que dicho curso de estructuras de datos utilizase un lenguaje OO para expresar los tipos abstractos de datos, de modo que el alumno al llegar al curso de OO del tercer año ya estuviese familiarizado con el lenguaje OO que se emplease en la parte práctica"*. El planteamiento aquí es hipotético, y la decisión final puede depender mucho del contexto de desarrollo. De manera muy resumida, podemos encontrar dos grupos de factores que apuntan hacia las direcciones opuestas, en nuestro contexto particular:

- Los planes de estudios de la FIUM y la tradición –entendida en el sentido más amplio– apuntan hacia una introducción más tardía de la POO, para la cual existe una asignatura específica en tercer curso. Ya hemos comentado que la elección de uno u otro enfoque no puede concebirse como una decisión adoptada de manera unilateral. No tiene sentido que una asignatura adopte cierto enfoque si éste resulta contradictorio con el resto de asignaturas del plan; y en nuestros planes actuales existe ya una asignatura encargada de la enseñanza del paradigma OO.

[8] Si bien no aparece en la troncalidad de las directrices generales propias, como vimos en el apartado 2.2.1. Pero hay una razón evidente para esto: en el año de su publicación, 1990, la POO estaba aún en sus inicios.

[9] Debemos darlo por hecho, aunque ya vimos que un 20 % de los alumnos de segundo se matriculan sin tener aprobadas las asignaturas de programación de primero.

- Las distintas recomendaciones internacionales, así como la situación actual en la gran mayoría de las universidades españolas, apuntan a una introducción mucho más temprana de la POO. Ya vimos que el informe CC2001, incluso en el acercamiento imperativo primero, propone la introducción de la POO en el segundo semestre del primer año. Este factor no debe ser entendido como una simple tendencia, sino como el resultado de la experiencia de todas las instituciones correspondientes. Lo que de esto se deriva es la necesidad de introducir, de alguna manera, los elementos más básicos de la POO en la asignatura AED.

Algoritmos, estructuras de datos y paradigmas de programación. No dudamos de la trascendencia de la elección del paradigma adecuado para la asignatura, pero al mismo tiempo debemos relativizar la decisión en su justa medida. Si revisamos los objetivos formativos propuestos para la materia de concurso, encontramos que gran parte de los temas son independientes del paradigma de programación: especificaciones formales, algoritmos sobre grafos, análisis de complejidad, técnicas de diseño de algoritmos, etc. Sólo el bloque de estructuras de datos puede resultar influido en parte por la adopción de un enfoque estructurado o uno OO. Pero, aún en estos temas, la asignatura debe incidir en las ideas novedosas que aportan estas técnicas en cuanto a la representación de los datos, como por ejemplo, la necesidad de introducir condiciones de balanceo en las representaciones arbóreas, la importancia de seleccionar buenas funciones de dispersión, o la idea del trie como un árbol de prefijos. Dicho de otro modo, para explicar el algoritmo de Dijkstra o la programación dinámica la POO no aporta una mejor comprensión, sino más bien todo lo contrario. La explicación de estas técnicas desde la perspectiva OO, si bien es factible, resulta en la mayoría de los casos forzada y antinatural; más aún si tenemos en cuenta que lo recomendable es usar un pseudolenguaje lo más sencillo e intuitivo posible. También en este sentido, podemos recordar el hecho de que algunas universidades –aunque no muchas– aplican sobre las mismas técnicas un enfoque funcional o algebraico, lo que vuelve a dar muestra de la independencia entre técnica de AED y paradigma.

Como se indica en [García'01], *"los conceptos de la POO (clase, objeto, mensaje, herencia, polimorfismo y ligadura dinámica) y el modelo computacional subyacente son aplicables cuando es necesario estructurar programas en diferentes módulos"*. De esta manera, sí puede tener sentido la introducción de la POO en las prácticas de la asignatura, sobre todo en las que se refieren a la implementación de tipos abstractos de datos. Si los alumnos deben programar uno o varios de los tipos estudiados en un entorno concreto, lo más razonable y adecuado es que sean implementados mediante clases, como mecanismo que completa y mejora las posibilidades de la programación modular. Mientras tanto, la teoría se debe centrar en los conceptos independientes del paradigma.

Decisión tomada. Después del repaso de factores y condicionantes, llegamos a la toma de decisiones. Creemos que es necesario tender hacia una familiarización progresiva de los estudiantes con la POO desde el segundo curso de la carrera, y en concreto en la asignatura AED. Pero esta introducción de los conceptos OO debe producirse fundamentalmente en la parte práctica de la asignatura, y limitarse a los aspectos más elementales. El desarrollo teórico de la asignatura debe realizarse en un pseudocódigo de alto nivel, flexible, intuitivo, de fácil uso y que abstraiga las cuestiones de sintaxis –e incluso de paradigma, si ello fuera posible–. En la parte práctica, se puede emplear un lenguaje OO pero presentando sólo los conceptos estrictamente necesarios: definición de clases; y creación, uso y eliminación de objetos. Todos los conceptos que surgen de la herencia (jerarquías de clases, polimorfismo, ligadura dinámica, clases abstractas, interfaces, reglas de asignación, métodos diferidos, redefinición, herencia múltiple, privada, virtual, etc.) pueden y deben ser omitidos, ya que son objetivos

de la asignatura de tercero de POO[10]. Los alumnos deberían ser capaces de programar las técnicas estudiadas tanto en entornos OO como en imperativos, valorando las ventajas que supone la utilización de clases y objetos.

Contenidos de la asignatura

Creemos que no tiene sentido y es poco realista plantear la selección de contenidos de la asignatura como un proceso aséptico basado exclusivamente en los fundamentos de la materia. El proceso empieza más bien con una revisión de la situación actual, reflexionando sobre los aspectos que deben ser modificados. Para esto debemos responder a una serie de cuestiones: ¿se cumplen los objetivos educativos con los contenidos actuales?, ¿son razonables los contenidos con la carga teórica y práctica asignada a la asignatura?, ¿existe alguna omisión en los contenidos actuales, en relación a lo que las recomendaciones internacionales establecen?, ¿se alejan sospechosamente los contenidos, en algún tema, de la tendencia predominante en nuestro entorno nacional?, ¿se han detectado carencias en los cursos pasados que aconsejen la revisión de algunos contenidos?

No contestaremos individualmente a estas cuestiones, sino que incidiremos únicamente en lo que, en función de la reflexión llevada a cabo, creemos que debe ser modificado de la programación actual de la asignatura. En concreto, podemos señalar los siguientes aspectos:

- Hasta la fecha, los contenidos teóricos de la asignatura AAED se han podido ajustar de forma exacta a los 6 créditos asignados, cubriéndose el programa de manera completa. Los 2 créditos prácticos han sido utilizados para la realización de prácticas de pizarra, permitiendo realizar numerosos ejercicios de cada tema. Pero recordemos que los nuevos planes de II suponen una **reducción global de 2 créditos**. No está claro si las prácticas de pizarra deben contabilizarse en los créditos teóricos o prácticos. En el primer caso, seguir con una dedicación similar para los ejercicios de clase requeriría reducir muy significativamente los contenidos teóricos, tanto como un 33 %. En el segundo caso, serían las prácticas de programación las que se verían afectadas. Por lo tanto, es necesario saber encontrar la solución adecuada a este problema de reducción de carga.

- En el punto anterior justificamos la decisión de introducir los aspectos más básicos de la **programación orientada a objetos**, por lo que los contenidos necesarios deben ser añadidos de alguna manera al programa. Ya vimos que estos temas deberían desarrollarse fundamentalmente en la parte práctica de la asignatura. No obstante, también es necesario que se haga una introducción teórica, que aunque no muy larga puede descontar tiempo para otros temas. Estimativamente, puede requerirse sobre 1 hora de teoría y por encima de 3 horas de laboratorio.

- Atendiendo a las recomendaciones más autorizadas, podemos detectar una carencia de los planes y programas actuales en los temas de **verificación y pruebas** de software, así como en el **análisis experimental** de algoritmos. Estos contenidos aparecen también en las directrices generales propias pero, como vimos en la tabla 4.1, no hay ninguna asignatura obligatoria en nuestros planes donde se cubran con la suficiente profundidad[11]. Por sus características, se trata de aspectos relacionados fundamentalmente con el trabajo práctico.

[10] Esta variante de la OO suprimiendo lo relativo a la herencia y todo lo que se deriva de la misma, es lo que algunos autores denominan "programación basada en objetos", [Gómez'01].

[11] Podemos señalar la existencia de la asignatura "Herramientas de Programación" de segundo, que incluye estos temas en su programa, pero que tiene carácter optativo y sólo aparece en ITIG.

- La adición de nuevos temas y la reducción de carga conlleva necesariamente la supresión de otros contenidos que han venido formando parte del temario de la asignatura. De esta forma, debemos identificar los temas fundamentales y los **contenidos no prioritarios**. Podemos reconocer los siguientes, ordenados de más a menos prescindibles (en algunos casos porque podrán ser estudiados en asignaturas posteriores y en otros por no estar en el núcleo de conocimientos del CC2001): árboles de juego, estructuras de datos duales, árboles B, programación dinámica, ramificación y poda, y especificaciones algebraicas. Claramente, la lista es mucho más larga de lo que sería razonable suprimir del temario, y en principio se deben intentar impartir todos los temas. Pero las circunstancias de clase pueden obligarnos a reducir el programa. Es más, la futura adaptación al EEES dará lugar previsiblemente a una considerable reducción de las horas de clase disponibles, ante lo cual conviene estar preparados.

Las anteriores indicaciones marcan una guía sobre los principales aspectos en que debe evolucionar el programa de la asignatura, tomando como base la planificación actual. No hemos hecho referencia a aquellas técnicas o conceptos que consideramos necesario mantener, y que fueron presentados en la sección 4.1. Esto no implica, no obstante, que no se deban revisar todos los temas y contenidos a escala más fina, con la consiguiente revisión de los materiales docentes; pero esto será cuestión de los siguientes capítulos. En definitiva, a continuación hacemos explícita la lista de contenidos seleccionados para el diseño de la asignatura –tanto los que se mantienen como los que cambian–. Los presentamos aquí de forma desordenada; en el siguiente punto entraremos en la organización de estos contenidos en temas y bloques, y en la planificación temporal de las actividades docentes.

1. Tipos abstractos de datos, abstracciones, tipos y mecanismos de abstracción.
2. Mecanismos de los lenguajes que dan soporte al uso de abstracciones: módulos y clases.
3. Bases de la programación orientada a objetos: definición de clases; creación, uso y eliminación de objetos.
4. Especificaciones formales: algebraicas; y mediante pre- y postcondiciones.
5. Tipos abstractos de datos básicos: tipos lineales, conjuntos, diccionarios, árboles, y grafos.
6. Especificación de algunos tipos básicos mediante una notación algebraica.
7. Tipos lineales: repaso breve de listas y arrays; estructuras de listas múltiples (u ortogonales).
8. Tablas de dispersión: tipos de dispersión (abierta y cerrada); funciones de dispersión; resolución de colisiones.
9. Árboles trie: representación, algoritmos y aplicaciones.
10. Relaciones de equivalencia en conjuntos: árboles de punteros al padre, optimizaciones.
11. Árboles de búsqueda: árboles binarios de búsqueda; árboles perfectamente balanceados; árboles AVL (operaciones de manipulación); árboles B (árboles de búsqueda n-arios, operaciones de manipulación).
12. Representación de grafos: matrices y listas de adyacencia; representación de grafos *escasos*.
13. Terminología y operaciones elementales sobre grafos: caminos, ciclos, flujos, árboles de expansión, conectividad, etc.; recorridos primero en profundidad y en anchura.

14. Idea de la NP-completitud aplicada sobre problemas clásicos de grafos: ciclo hamiltoniano, coloración, viajante, isomorfismo.

15. Algoritmos clásicos sobre grafos: Prim, Kruskal, Dijkstra, Floyd, y Warshall.

16. Otros algoritmos sobre grafos: búsqueda de puntos de articulación; componentes fuertemente conexos; grafos dirigidos acíclicos; ordenación topológica.

17. Análisis de algoritmos, objetivos e importancia del análisis.

18. Análisis mediante conteo de instrucciones y de memoria.

19. Análisis experimental de algoritmos: técnicas y herramientas matemáticas.

20. Verificación y pruebas de software.

21. Notaciones asintóticas de complejidad algorítmica: O-grande, o-pequeña, Ω, Θ, notaciones condicionadas, notaciones con varios parámetros; significado y uso de las notaciones.

22. Resolución de ecuaciones de recurrencia: recurrencias homogéneas y no homogéneas; método de la ecuación característica; el papel de los casos base.

23. La técnica de diseño de divide y vencerás: idea, esquema algorítmico y ejemplos de aplicación.

24. Mejoras en problemas de cálculo numérico usando divide y vencerás.

25. La técnica de programación dinámica: idea, esquema algorítmico y ejemplos de aplicación.

26. El principio de optimalidad de Bellman: aplicación en algoritmos de programación dinámica.

27. Contraste y comparación de las estrategias de aplicación de descomposición recurrente: técnicas descendentes (divide y vencerás) y ascendentes (programación dinámica).

28. La técnica de avance rápido: idea, esquema algorítmico y ejemplos de aplicación.

29. Demostración de optimalidad en algoritmos de avance rápido.

30. Utilización de avance rápido en resolución heurística de problemas complejos.

31. La técnica de backtracking: idea, esquema algorítmico y ejemplos de aplicación; implementación recursiva e iterativa.

32. Optimizaciones sobre los algoritmos de backtracking.

33. La técnica de ramificación y poda: idea, esquema algorítmico y ejemplos de aplicación.

34. La ramificación y poda como una generalización y mejora de backtracking.

35. El compromiso tiempo-precisión en la estimación de cotas en ramificación y poda.

36. Comparación de soluciones a un mismo problema usando avance rápido, programación dinámica, backtracking y ramificación y poda.

37. Resolución de problemas de juegos: árboles de juego y estrategia minimax.

38. Optimización de la estrategia minimax mediante poda alfa-beta.

En los capítulos 5, 6 y 7 se desarrollan todos estos contenidos, dentro de los temas correspondientes. Se han omitido aquí los ejemplos de aplicación de las diferentes técnicas, que no son parte del programa sino que pueden ser cambiados y actualizados.

Organización y ordenación de contenidos

Los contenidos seleccionados en el punto anterior no constituyen un programa en sí mismos, sino que es necesario organizarlos en temas, bloques y partes, en actividades teóricas y prácticas, buscando una ordenación coherente y adecuada para el mejor aprendizaje de los alumnos. De esta forma, si en el punto anterior hemos desmenuzado el programa, ahora nos toca darle una estructura; e, igual que antes, no podemos obviar la organización dispuesta en el programa actual. Las principales guías que marcan la organización de contenidos son las siguientes.

- **Partes y Bloques de la asignatura.** El primer nivel de organización de los contenidos es la descomposición de la asignatura en partes. En una asignatura como AED, con una temporización anual, conviene distinguir al menos dos grandes partes que marcan el ámbito de cada cuatrimestre. Podemos distinguir las siguientes partes: **Parte I - Estructuras de datos**; **Parte II - Algoritmia**. La diferenciación de los contenidos correspondientes a una u otra parte es directa, como lo demuestra el hecho de que en muchos planes de estudios –por ejemplo, en los planes antiguos de las Ingenierías Técnicas en la FIUM– ambas partes son dispuestas en asignaturas independientes. En la Parte I se incluyen los conceptos relacionados con la abstracción, las especificaciones, los tipos de datos elementales y las estructuras de representación de tipos más frecuentes. La Parte II se puede descomponer a su vez en un **Bloque de Análisis de Algoritmos** y otro **Bloque de Diseño de Algoritmos**. El primero presenta las herramientas necesarias para medir y expresar la eficiencia de los algoritmos, y el segundo contiene esquemas algorítmicos y técnicas generales de diseño de algoritmos.

  A pesar de la separación en dos partes, hay que evitar la percepción frecuente de que ambas no están relacionadas entre sí, y que los algoritmos y las estructuras de datos persiguen objetivos distintos, y manejan principios independientes. Se debe hacer énfasis en la necesidad inevitable de ambos en la resolución de problemas y en los numerosos temas comunes que aparecen tanto en uno como en otro ámbito. Además, en ciertos aspectos la división es más artificial que objetiva; por ejemplo, ¿en qué parte deberían ir los algoritmos clásicos sobre grafos?

  En cuanto a la ordenación de las partes y bloques, se plantean diferentes alternativas: primero algoritmos y luego estructuras; primero estructuras y luego algoritmos; primero análisis y luego diseño; etc. Se deben tener en cuenta las numerosas relaciones de dependencia entre las distintas partes y bloques: (1) el análisis de algoritmos se usa en todas las restantes partes; (2) muchas técnicas de diseño usan estructuras de datos tratadas en la parte correspondiente; y (3) algunos algoritmos de grafos usan técnicas introducidas en el bloque de diseño de algoritmos. Pero considerando otros criterios didácticos, se debería seguir una línea de complejidad creciente, intentando no introducir al principio temas que son tradicionalmente difíciles para los alumnos, con la consiguiente pérdida de motivación que ello supondría. En concreto, resulta poco aconsejable –desde el punto de vista pedagógico– empezar con el análisis de algoritmos, por el fuerte componente matemático que supone, y su mayor dificultad para los alumnos.

  Creemos que la decisión más razonable es comenzar por la parte de Estructuras de Datos, incluyendo un tema inicial de repaso de los conceptos básicos estudiados en primero. Esta parte

empieza con las técnicas de especificación, que pueden ser aplicadas después sobre los distintos tipos estudiados. En segundo lugar se continúa con la parte de Algoritmia, empezando por el análisis (imprescindible para el estudio de ejemplos posteriores) y siguiendo con las técnicas generales de diseño. En definitiva, la organización en partes y bloques de la asignatura sería la siguiente:

- **Parte I - Estructuras de Datos**
  Bloque de Abstracciones y Especificaciones
  Bloque de Tipos y Estructuras
- **Parte II - Algoritmia**
  Bloque de Análisis de Algoritmos
  Bloque de Diseño de Algoritmos

Una organización similar se puede encontrar en las recomendaciones del informe CC2001, así como en los programas de muchas de las universidades españolas analizadas. En relación a las objeciones planteadas por las dependencias antes expuestas, podemos argumentar lo siguiente: (1) el análisis realizado en la Parte I será a un nivel muy básico, para lo cual resulta suficiente con los conocimientos adquiridos por los alumnos en las asignaturas de primer curso; (3) el conocimiento de las técnicas generales no es imprescindible para la comprensión de los algoritmos clásicos sobre grafos, explicados en la primera parte, y simplemente se plantea la visión de "técnica como generalización de algoritmos concretos", frente a "algoritmos como aplicación de una técnica".

- **Organización de la materia práctica.** Evidentemente, debe haber una estrecha coordinación entre la teoría y la práctica de la asignatura. Puesto que el objetivo de las prácticas es poner a prueba las técnicas y conceptos estudiados de forma teórica, la descomposición de las prácticas debe ser acorde con la estructura antes establecida para la teoría. La organización de las prácticas difiere sustancialmente en la asignatura AAED de plan antiguo (con sólo 2 créditos), y en AED de plan nuevo (con 6 créditos). En el primer caso tiene más sentido utilizar las horas disponibles como prácticas de pizarra, en las que se puedan resolver una gran cantidad de ejercicios y solucionar dudas de los alumnos.

  Por su parte, las prácticas de AED están más orientadas a la programación, por lo que se debe incluir un primer módulo de manejo del lenguaje y el entorno de programación seleccionados. El desarrollo de este primer módulo de prácticas coincidiría con el bloque de abstracciones y especificaciones, y con los primeros temas del bloque teórico de tipos y estructuras. A continuación, coincidiendo con el núcleo de la parte de estructuras de datos, tendría lugar el módulo de implementación y manejo de estructuras de datos. Para la parte de algoritmia, proponemos también un desarrollo simultáneo y coordinado de teoría y prácticas, y establecemos una práctica de análisis y otra de diseño de algoritmos. En definitiva, a gran escala la organización de los contenidos prácticos sería la siguiente:

  - **Módulo I - Seminarios de programación**
  - **Módulo II - Implementación y manejo de estructuras de datos**
  - **Módulo III - Análisis de algoritmos**
  - **Módulo IV - Diseño de algoritmos**

  Al mismo tiempo, también se debería disponer el tiempo suficiente para la realización de prácticas de pizarra, que permitan realizar más ejercicios que los que da tiempo a hacer en clases de teoría.

- **Temas de la asignatura.** Los temas constituyen la unidad básica de estructuración de las asignaturas. De alguna manera, un tema agrupa bajo una denominación un conjunto de ideas, conceptos y técnicas relacionadas; en su conjunto, dan a la asignatura una estructura ordenada y coherente. La descomposición en temas es también la base para la construcción mental en los estudiantes de la estructura global de la asignatura. Por lo tanto, una adecuada organización en temas puede mejorar el aprendizaje de los alumnos, aunque los contenidos no cambien. En el apartado anterior se estableció una lista de casi 40 contenidos a ser tratados en mayor o menor profundidad; pero sería muy poco pedagógico fijar un programa con 40 temas. En general, se recomienda definir un número reducido de temas[12]. Además, en cuanto a la organización de los temas se debería intentar cumplir los dos siguientes criterios: (1) en general, el orden debería avanzar en nivel creciente de complejidad, empezando por los conceptos de más fácil asimilación hasta los más elaborados; (2) la ordenación debería respetar lo máximo posible las relaciones de dependencia o uso entre los contenidos tratados en los diferentes temas.

  En el caso particular de la asignatura objeto de concurso, las líneas que han guiado la estructuración en temas son las siguientes:

  - En primer lugar, todos los contenidos relacionados con abstracciones, tipos básicos y especificaciones formales, tiene sentido incluirlos dentro del mismo tema. Además, en cuanto que muchos de los aspectos suponen un repaso de conceptos estudiados en primero, y considerando que las especificaciones son un tema de aplicación en los restantes puntos del temario, resulta adecuado empezar el programa por este tema.
  - Hay una gran cantidad de técnicas de representación de datos, que podrían desbordar el temario: tablas de dispersión, listas múltiples, árboles trie, etc. Sin embargo, hay una idea subyacente en todas ellas: la representación de los TAD conjunto y diccionario. Aunque con características completamente distintas, tanto en las tablas de dispersión como en los árboles trie, el objetivo final es ofrecer un conjunto de operaciones (inserta, suprime, consulta) de forma eficiente. Estas operaciones son las que determinan el tipo abstracto. Las características propias de cada técnica no originan diferentes TAD, sino que son diferentes implementaciones de un mismo TAD. En definitiva, aparecen dos temas: uno dedicado a la definición de los TAD conjunto y diccionario y a las estructuras más sencillas (incluyendo tablas de dispersión y listas múltiples); y otro que trata las representaciones de conjuntos y diccionarios mediante árboles. En ambos casos, la línea conductora que se sigue es la visión de la estructura como una concretización del tipo abstracto.
  - Todos los aspectos relacionados con los grafos están contenidos dentro de un mismo tema, incluyendo representación, problemas y algoritmos. Otra posibilidad sería dejar los algoritmos sobre grafos para la parte de algoritmia de la asignatura. Pero creemos que la primera opción es mejor, por varios motivos: se consigue un reparto más uniforme de los contenidos entre las partes; se concentra más la atención en el modelo y resolución de problemas mediante grafos, en lugar de repartirlas en diferentes temas; el tema de grafos puede servir de puente entre la parte de estructuras y la de algoritmos.
  - En cuanto a las técnicas de diseño de algoritmos, optamos por separarlas en diferentes temas, cada uno dedicado a una técnica específica. Es común en otros programas encontrar un sólo tema de diseño, donde se incluyen todas las técnicas como apartados. Pero el tamaño de este tema sería desproporcionado respecto al resto, por el elevado número de contenidos

[12] En este sentido, algunos autores proponen la regla empírica de que un buen temario es el que cabe en una sola transparencia; que se pueda leer con claridad, claro.

que debe incluir. Por ello consideramos más adecuada la separación de técnicas a nivel de temas, lo cual no debe impedir el planteamiento de las múltiples similitudes y aspectos que aparecen en todos los métodos de forma recurrente.

Además de estas consideraciones, añadimos un primer tema (el Tema 0) de presentación de la asignatura y repaso de los conceptos básicos estudiados en primero. En consecuencia, la estructuración en temas de la asignatura es la siguiente. Para cada tema indicamos los contenidos que incluyen, según la numeración de la página 171.

- **Parte I - Estructuras de Datos**

  Tema 0. Estructuras de datos y algoritmos (1, 2)

  Bloque de Abstracciones y Especificaciones

  Tema 1. Abstracciones y especificaciones (1, 2, 3, 4, 5, 6)

  Bloque de Tipos y Estructuras de Datos

  Tema 2. Conjuntos y diccionarios (5, 7, 8)

  Tema 3. Representación de conjuntos mediante árboles (5, 9, 10, 11)

  Tema 4. Grafos (5, 12, 13, 14, 15, 16)

- **Parte II - Algoritmia**

  Bloque de Análisis de Algoritmos

  Tema 5. Análisis de algoritmos (17, 18, 19, 21)

  Tema 6. Ecuaciones de recurrencia (21, 22)

  Bloque de Diseño de Algoritmos

  Tema 7. Divide y vencerás (23, 24)

  Tema 8. Algoritmos voraces (28, 29, 30)

  Tema 9. Programación dinámica (25, 26, 27)

  Tema 10. Backtracking (31, 32, 36)

  Tema 11. Ramificación y poda (33, 34, 35, 36)

  Tema 12. Resolución de juegos (37, 38)

Si analizamos comparativamente este temario con el programa de la asignatura durante el curso anterior –que estudiamos en el apartado 4.1–, podemos apreciar una estructura de temas bastante parecida. Básicamente no hay grandes modificaciones, pero sí pequeños cambios con el propósito de refinar y depurar el programa actual, y tratar de hacer énfasis en los puntos más débiles. Podemos señalar entre las principales diferencias las siguientes:

- En lo referente a la presentación de la asignatura, se introduce el Tema 0, tomando en parte contenidos del Tema 5 –que desaparece– en lo referente al repaso de conceptos básicos. No obstante, esto no impide que al retomar el segundo cuatrimestre también sea necesario volver a recolocar a los alumnos dentro de la asignatura, con alguna clase de repaso.
- Se ha modificado el nombre de algunos temas, con el objetivo de dar una mejor idea de lo que contienen con una simple lectura del temario. Aunque este cambio pueda parecer superfluo, el nombre puede influir en la percepción del alumno y en el énfasis particular que se pone en cada tema. De esta manera, el tema “Conjuntos” pasa a denominarse “Conjuntos y diccionarios”, y “Métodos avanzados de representación de conjuntos” se cambia por “Representación de conjuntos mediante árboles”, que da una mejor idea del contenido del mismo.

- El Tema 6 del temario antiguo contenía todo lo relativo al bloque de análisis de algoritmos, lo que daba lugar a un exceso de contenidos, bastante desproporcionado con el resto de temas. Por este motivo ha sido descompuesto en dos temas: "Análisis de algoritmos" y "Ecuaciones de recurrencia". Hay que tener en cuenta que esto no supone la introducción de nuevos contenidos, sino una estructuración diferente.
- También se ha establecido una estructuración diferente para las técnicas de resolución de juegos, que antes aparecían en parte en el tema de backtracking, y la otra parte en ramificación y poda. Ahora se ha creado un nuevo tema que incluye todo lo relativo a la resolución de juegos. Este tema queda al final del programa, indicando su carácter menos prioritario para la asignatura.

La inclusión de contenidos en los temas es una primera indicación sobre los aspectos concretos a tratar en cada uno. En los siguientes apartados profundizaremos en los puntos a desarrollar en cada tema, y en el número de horas recomendado para cada uno. En cualquier caso, se pretende ofrecer la suficiente flexibilidad para permitir una adaptación a las diferentes circunstancias que se pueden producir durante el desarrollo de la acción docente.

Lenguaje de programación

Para acabar con la exposición y justificación de decisiones, abordamos en este punto la elección del lenguaje de programación propuesto para las prácticas de la asignatura AED. Debemos hacer hincapié en nuestra convicción de que la explicación teórica debe realizarse haciendo uso de un pseudolenguaje intuitivo y no ligado a ningún lenguaje de programación particular. En algún caso concreto podrá interesar referirse a alguna cuestión específica sobre la implementación, pero no debe ser lo habitual. La docencia teórica se debe centrar más bien en las ideas y conceptos fundamentales, fomentando el pensamiento algorítmico y evitando una asociación errónea de una técnica con un lenguaje o paradigma concretos. Por otro lado, recordamos que la propuesta de prácticas para la asignatura AAED, de los planes antiguos, aconsejaba utilizarlas como prácticas de pizarra, y por lo tanto tampoco deberían estar orientadas a un lenguaje de programación específico.

En el mundo de la docencia universitaria en informática la elección del lenguaje de programación con el que los alumnos deben realizar las prácticas es uno de los aspectos que mayor debate y controversia ha generado. Pero las discusiones se han centrado fundamentalmente en el lenguaje manejado en el primer curso, como uno de los factores que más afectarán sobre la forma de pensar de los estudiantes[13]. La elección del lenguaje usado en el segundo curso y posteriores deben estar más enfocado a los objetivos particulares de la asignatura, ya que en este caso no es tan importante el aprendizaje del lenguaje como la puesta en práctica de los conceptos estudiados en la asignatura. Además, se da por hecho que los alumnos ya conocen algún lenguaje de programación, lo que sin duda facilita el aprendizaje de otros lenguajes. Existen muchas menos sugerencias respecto al lenguaje que debe ser estudiado en segundo curso. En general, la mayoría de las recomendaciones internacionales recomiendan la introducción de un segundo lenguaje de programación, diferente al estudiado en primer curso.

En nuestra opinión, principal característica que debería tener un lenguaje de programación adecuado para las prácticas de la asignatura AED debería ser la coherencia con el planteamiento paradigmático propuesto. En nuestro caso el planteamiento consiste en un enfoque mixto, en el que

[13]Por ejemplo, en palabras de E.W. Dijkstra, *"es prácticamente imposible enseñar buenas prácticas de programación a estudiantes que han estado expuestos a BASIC; como programadores potenciales están mentalmente mutilados más allá de cualquier esperanza de regeneración"*; algo con lo que, dicho sea de paso, no estamos de acuerdo.

tomando las bases del paradigma imperativo se propone la inclusión de los conceptos más elementales de la programación orientada a objetos, descartando todo lo relativo a la herencia. El propósito de este enfoque es la posibilidad de implementar mediante clases los tipos de datos que aparezcan en la resolución de las prácticas, pero dando por hecho programas esencialmente basados en programación modular. Con esto se intenta evitar que el interés en y la dificultad de las prácticas se desvíe hacia cuestiones propias de la programación OO. En consecuencia, debemos escoger un lenguaje OO, y que al mismo tiempo permita una fácil utilización al "estilo imperativo". Por otro lado, el lenguaje debería ser el mismo tanto para el primero como para el segundo cuatrimestre, ya que el aprendizaje de diferentes lenguajes podría significar una reducción innecesaria del tiempo disponible para los aspectos de resolución de problemas de interés.

En función del anterior requisito básico, consideramos que el lenguaje más adecuado para la realización de las prácticas es **C++**. El lenguaje C++ utiliza un enfoque híbrido, que permite una programación bajo un enfoque imperativo o bien usando los conceptos de la OO. El programador tiene completa libertad para decidir qué partes implementa mediante clases y cuáles mediante una descomposición funcional tradicional. Esta característica ha sido duramente criticada cuando se trata de analizar su adecuación para la enseñanza de la POO. Para estos casos se aconseja la utilización de un lenguaje OO de los denominados *puros*, como Eiffel, Smalltalk o Java[14]. Estas críticas no son un obstáculo para la utilización de C++ en una asignatura como AED, donde el uso de clases y objetos no supone una parte central. Además, estamos convencidos de que la presentación de conceptos elementales de OO no es un obstáculo para la comprensión de asignaturas posteriores que profundicen en el paradigma OO, sino más bien todo lo contrario.

Es bien conocido que el lenguaje C++ toma como base el lenguaje C, del cual se puede considerar como un superconjunto. En consecuencia, la enseñanza de C se hace aconsejable como paso previo a la presentación de C++; no es estrictamente necesario, pero sin duda conocer el lenguaje C es beneficioso para la formación –e, incluso nos atreveríamos a decir, para la futura actividad profesional– de los alumnos.

Además de la adecuación de C++ al enfoque propuesto en la asignatura, podemos destacar otras **características adicionales del lenguaje** que apoyan esta elección:

- Los lenguajes C/C++ son actualmente los más extendidos, tanto en el mundo académico como en el de la industria. Podemos referenciar aquí algunos datos a este respecto comentados previamente. Por ejemplo, según el estudio de 25 universidades españolas presentado en el apartado 4.3.1, en 13 titulaciones se usa C o C++ en primer curso. Si nos fijamos en las universidades norteamericanas, el 54 % empiezan introduciendo C++, de acuerdo con el informe citado en [Gómez'01][15]. En lo que atañe más específicamente a la materia AED, 6 de las 8 titulaciones de universidades españolas analizadas en el apartado 4.3.2 utilizan C++ para el desarrollo de las prácticas. También es indudable la extensión de los lenguajes C/C++ en el mundo profesional, lo cual es un aliciente añadido para los alumnos.

- C++ permite la definición de tipos y funciones parametrizados. Si excluimos los lenguajes usados únicamente en entornos académicos, C++ es uno de los pocos lenguajes que ofrecen esta característica. La posibilidad de crear tipos genéricos es realmente muy interesante, fundamentalmente en la implementación de estructuras de datos. De esta forma, en la programación de árboles trie, tablas de dispersión, etc., no se tendrá que repetir el código para cada tipo contenido, sino que los contenedores serán definidos de forma genérica y luego instanciados según las

[14] Aunque según algunos autores tampoco Java puede ser considerado plenamente como OO puro.
[15] Los últimos datos disponibles se refieren al curso 1999/2000.

necesidades de cada aplicación. La cuestión de la genericidad ha sido una de las más discutidas en nuestra Facultad, y ha sido usada como justificación casi exclusiva para propuestas de enseñar en primer curso lenguajes como Modula-3 (versión reciente pero poco extendida de Modula) o Java (aunque paradójicamente no permite genericidad).

- Otra interesante propiedad avanzada que ofrece C++ es la utilización de asertos, como un mecanismo que permite la introducción de pre- y postcondiciones. Estos conceptos son estudiados en teoría, dentro del tema dedicado a las especificaciones, presentando las ideas básicas de la programación por contrato. El uso de C++ posibilita la aplicación en prácticas de los conceptos estudiados en teoría.

- La alta disponibilidad de compiladores y herramientas de programación es un requisito no imprescindible, aunque sí bastante aconsejable. En este aspecto, está claro que los lenguajes C/C++ están muy bien situados. Existen numerosos compiladores y otras herramientas tanto para Windows como para los diferentes entornos Linux, así como en otros sistemas operativos menos habituales. Las herramientas de edición, compilación, pruebas y depuración bajo Linux están bastante extendidas, son muy robustas y además en la mayoría de los casos son gratuitas. En consecuencia, se propone para las prácticas la utilización de C/C++ bajo Linux; la versión concreta a utilizar dependerá de la distribución y versión de Linux instalada en los laboratorios de prácticas.

- El lenguaje C es usado en otras asignaturas de nuestros planes de estudios, que se imparten después o simultáneamente con AED. Para estas asignaturas supone una ventaja que los alumnos lleguen sabiendo manejar el lenguaje. Además, está justificado el aprendizaje de C++ por las numerosas mejoras que ofrece respecto de C; aparte de las ya referidas, podemos mencionar la posibilidad de pasar parámetros por referencia, y la gestión de memoria dinámica mediante operadores del lenguaje.

Por otro lado, debemos reconocer también los inconvenientes de C++, para tratar de minimizar su efecto en la enseñanza. Entre los **puntos débiles** de este lenguaje podemos destacar los siguientes:

- C++ ha sido ampliamente criticado por ser un lenguaje muy complejo, tremendamente amplio y con una flexibilidad tal (debida a la posibilidad de sobrecarga, parámetros por defecto, redefinición de operadores, etc.) que puede causar confusión a los programadores noveles. Por ejemplo, el informe CC2001 advierte que *"muchos lenguajes orientados a objetos usados en la industria –particularmente C++, aunque hasta cierto punto también Java– son significativamente más complejos que los lenguajes clásicos. A menos que los instructores tengan un cuidado especial en introducir el material de modo que se limite esta complejidad, tales detalles pueden fácilmente desbordar a los estudiantes de primer año"*. Aún teniendo en cuenta que en el caso de AED se trata de una asignatura de segundo curso, el problema puede afectar también a los estudiantes. Por esto, conviene reducir el lenguaje, limitando el número de conceptos estudiados de C++. En concreto, nuestra propuesta consiste en eliminar todo lo relativo a la herencia, en consonancia con el planteamiento metodológico realizado.

- Tanto C como C++ son lenguajes de un nivel relativamente bajo, con características como la aritmética de punteros, el tipado débil y aspectos próximos al nivel de la máquina, como la elevada cantidad de tipos numéricos que ofrece. Será necesario incidir en las buenas prácticas de programación, y en las técnicas de depuración y prueba, con el fin de intentar evitar o solventar los errores que se pueden producir por este hecho.

- Al enseñar el lenguaje C y a continuación C++, se corre el riesgo de que los alumnos confundan ambos lenguajes, no sabiendo distinguir las características de C++ que no están disponibles en C. Por este motivo, debe quedar clara la separación entre ambos lenguajes.

De cara a la asignatura POO, de tercero, el conocimiento de los conceptos más básicos del paradigma OO y el uso de un lenguaje como C++ debe suponer, en principio, una buena base de partida. Aunque, posiblemente, el mayor obstáculo para la asimilación plena de la filosofía OO podría ser la experiencia en proyectos que combinan fragmentos de código OO con otros que utilizan una descomposición funcional más tradicional. Para estos alumnos que han usado un enfoque mixto, la idea de que "todo son clases y objetos" puede resultar algo que limita, más que mejorar. Evitar esta visión *escéptica* requerirá una actitud abierta por parte de los alumnos. Pensamos que la asignatura POO tiene suficiente carga lectiva como para garantizar una presentación sólida y bien fundada de la orientación a objetos, y de las ventajas que sin duda ofrece.

4.4.2. Propuesta de materia teórica

Una vez presentadas y justificadas las principales decisiones tomadas en el diseño de la asignatura objeto de concurso, vamos a centrarnos en exponer más detalladamente la propuesta docente. En este apartado trataremos la propuesta para la parte teórica de la asignatura, haciendo un planteamiento inicial de los temas que constituyen el programa, así como su temporización. En el siguiente apartado analizaremos la propuesta para las actividades prácticas, y a continuación la coordinación entre teoría y prácticas.

Como expusimos en el anterior apartado, la estructura que se propone para el programa de la asignatura AAED –y para AED, del plan nuevo– consta de 12 temas, repartidos en dos partes y cuatro bloques de la siguiente manera:

- **Parte I - Estructuras de Datos**

  Bloque de Abstracciones y Especificaciones: Tema 1

  Bloque de Tipos y Estructuras: Temas 2-4

- **Parte II - Algoritmia**

  Bloque de Análisis de Algoritmos: Temas 5-6

  Bloque de Diseño de Algoritmos: Temas 7-12

Puesto que el primer bloque sólo consta de un tema, omitiremos en adelante la distinción entre los bloques de la primera parte de la asignatura. Tanto en la división de la asignatura en partes como en la descomposición en temas, se ha buscado una distribución lo más uniforme posible, de forma que quede clara la importancia relativa de cada unidad. En la tabla 4.3 se muestra el número de horas de clase asignadas a cada tema del programa, sobre el total de 60 horas establecidas en los planes de estudios. En cuanto a la distribución de las partes, se puede ver que la parte de estructuras de datos recibe algo menos de carga que la de algoritmos.

Aunque los 6 créditos de teoría corresponden "teóricamente" –y valga la redundancia– a 60 horas de clase, es muy habitual la pérdida de horas lectivas debido a imprevistos y, fundamentalmente, a las fiestas de naturaleza diversa[16]. Por este motivo es necesario conseguir una planificación flexible, que permita adaptarse a las distintas circunstancias, minimizado los perjuicios de la disminución de

[16]Podemos mencionar, por ejemplo, que el presente curso 2003/04, el número de horas de clase de AED ha quedado reducido a 52, por la coincidencia de fiestas los lunes, lo cual representa una pérdida del 13 % del tiempo disponible.

	Tiempo (horas)	Mínimo (horas)
ALGORITMOS Y ESTRUCTURAS DE DATOS	**60**	**48**
PARTE I: ESTRUCTURAS DE DATOS	**26**	**22**
Tema 0. Estructuras de datos y algoritmos	2	2
Tema 1. Abstracciones y especificaciones	4	3
Tema 2. Conjuntos y diccionarios	5	4
Tema 3. Representación de conjuntos mediante árboles	5	5
Tema 4. Grafos	10	8
PARTE II: ALGORITMICA	**34**	**26**
TECNICAS DE ANALISIS	11	9
Tema 5. Análisis de algoritmos	6	5
Tema 6. Ecuaciones de recurrencia	5	4
TECNICAS DE DISEÑO	23	17
Tema 7. Divide y vencerás	4	3
Tema 8. Algoritmos voraces	4	4
Tema 9. Programación dinámica	4	2
Tema 10. Backtracking	4	4
Tema 11. Ramificación y poda	5	4
Tema 12. Resolución de juegos	2	0

Tabla 4.3: Estructuración de temas de teoría propuesta para la asignatura "Ampliación de Algoritmos y Estructura de Datos". Descomposición en Partes, Bloques y Temas. La propuesta es la misma para la asignatura de plan nuevo. Tiempo: número ideal de horas de clase requerido para cada tema. Mínimo: número mínimo de horas de clase aconsejado.

horas de clase. En consecuencia, en la tabla 4.3 se ha introducido una columna "Mínimo", que indica el tratamiento mínimo en horas de clase que se aconseja para cada tema. Para concretar más sobre la manera en la que se puede realizar esta reducción de horas, en los capítulos 5 y 6 se indica para cada tema los ritmos que deberían seguirse para conseguir una exposición más lenta o más rápida –pero siempre tratando de ser realistas–. El último tema, "Tema 12 - Resolución de juegos", recibe 0 horas en la planificación mínima indicando su carácter opcional y, por lo tanto, la posibilidad de no impartirlo. Aunque el tiempo mínimo total es de 48 horas, no resultaría aconsejable que en todos los temas se siguiera el ritmo más rápido posible.

Dentro del Capítulo 3 se establecieron los objetivos de la asignatura, distinguiendo entre objetivos globales –perseguidos por la asignatura en su conjunto– y puntuales –que pueden ser asignados a temas concretos–. Retomamos aquí los objetivos puntuales, para indicar específicamente los objetivos educativos hacia los que se debe orientar cada tema. De esta manera podemos garantizar que todos los objetivos establecidos son cubiertos por los temas que se proponen. En concreto, la distribución de objetivos entre temas es mostrada en la tabla 4.4.

También entraremos más en detalle sobre los objetivos educativos de cada tema dentro de los siguientes capítulos de este proyecto docente. Pero antes, vamos a presentar de forma breve los temas, mostrando sus contenidos básicos y justificando su situación en el programa de la asignatura.

	OBJETIVOS PUNTUALES
ALGORITMOS Y ESTRUCTURAS DE DATOS	
PARTE I: ESTRUCTURAS DE DATOS	
Tema 0. Estructuras de datos y algoritmos	A1, A2, A4, D1, D2, D3, D4
Tema 1. Abstracciones y especificaciones	A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11
Tema 2. Conjuntos y diccionarios	B1, B2, B3, B4, B5, B6, B7, B8, B13
Tema 3. Representación de conjuntos mediante árboles	B1, B2, B3, B4, B5, B10, B11, B8
Tema 4. Grafos	B1, B2, B3, B4, B5, B9, B10, B11, B12, B13
PARTE II: ALGORITMICA	
TECNICAS DE ANALISIS	
Tema 5. Análisis de algoritmos	C1, C2, C3, C6, C7
Tema 6. Ecuaciones de recurrencia	C4, C5, C6
TECNICAS DE DISEÑO	
Tema 7. Divide y vencerás	D5, D6, D9, D10, D17
Tema 8. Algoritmos voraces	D5, D6, D8, D15, D17
Tema 9. Programación dinámica	D5, D6, D10, D11, D17
Tema 10. Backtracking	D5, D6, D7, D12, D13, D17
Tema 11. Ramificación y poda	D5, D6, D14, D15, D16, D17
Tema 12. Resolución de juegos	D5, D6, D17

Tabla 4.4: Asignación de objetivos educativos puntuales para cada tema de teoría.

Parte I - Estructuras de Datos

La primera parte de la asignatura, que corresponde aproximadamente al primer cuatrimestre del curso, está dedicada al estudio de los tipos de datos, incluyendo su especificación, tipos básicos, y estructuras eficientes de representación de los mismos en memoria. Durante los temas centrales de esta parte, la línea conductora es la implementación de los tipos abstractos conjunto y diccionario, que desemboca en el estudio de las técnicas de dispersión y distintas representaciones arbóreas. Esta parte también incluye el estudio de conceptos y problemas relacionados con los grafos. Aunque por la naturaleza de este tema –centrado en los algoritmos clásicos sobre grafos– podría haber sido incluido dentro de la parte de algoritmos, la inclusión aquí dota al temario de una mayor uniformidad y coherencia de contenidos. Además, como último tema de esta parte, puede verse como un puente hacia la segunda parte, en cuanto que combina planteamientos propios del ámbito de las estructuras de datos con otros más comunes en algoritmia.

Tradicionalmente, como vimos en el apartado 2.3.2, los alumnos muestran normalmente menos problemas de aprendizaje en los temas de estructuras de datos que en los de algoritmos. Esta es una de las razones por las que este bloque es situado en primer lugar, tratando de evitar un comienzo "brusco" de la asignatura, que desanime a muchos alumnos desde el principio. Pero no queremos ser dogmáticos en esta cuestión, ya que llevando un cuidado especial en el seguimiento del aprendizaje de

los alumnos, la asignatura podría también empezar por la parte de algoritmia.

Tema 0. Estructuras de datos y algoritmos. Antes de entrar de lleno en la asignatura, conviene empezar con una clase introductoria que puede considerarse fuera del programa de contenidos propiamente dicho –razón por la cual es enumerado como Tema 0–. Los propósitos de este tema son varios:

- Presentar la asignatura a los alumnos, sus objetivos, temario teórico, programa de prácticas y relaciones con otras asignaturas del plan. Esta presentación debe intentar motivar y despertar el interés de los alumnos por la asignatura.
- Repasar los conceptos más relevantes que los alumnos deben conocer de primer curso, como por ejemplo la definición de algoritmo y tipo abstracto de datos, la diferencia entre problema, programa y algoritmo, la diferencia entre tipo de datos, TAD y estructura de datos, y los principales tipos de datos usados en programación.
- Explicar las normas y modos de trabajo en la asignatura, tanto en la parte teórica como en las prácticas, así como otros aspectos relevantes, como los criterios de evaluación.

Como primer contacto del profesor con los alumnos, y viceversa, la adecuada preparación de este tema es fundamental. Los estudiantes deben ver que lo que estudiaron en primer curso es importante y tiene continuidad, y al mismo tiempo deben apreciar todos los campos potenciales donde pueden ser aplicados los conocimientos adquiridos en la asignatura.

Tema 1. Abstracciones y especificaciones. Este es el primer tema, propiamente dicho, de la asignatura. Ya hemos comentado la relevancia de la abstracción, que es señalada como una de las capacidades básicas de cualquier ingeniero en informática en las recomendaciones curriculares del CC2001. El estudio de los tipos y mecanismos de abstracción que aparecen en programación, que se plantea en este tema, es realmente un repaso de conceptos que los alumnos ya conocen de primero: tipos abstractos de datos, funciones como abstracciones de control, parametrización, genericidad, etc. No está de más insistir en estos conceptos, con el fin de concienciar a los alumnos de su importancia.

El núcleo del tema lo forma el estudio de las notaciones formales de especificación. En concreto, se presentan dos técnicas: las algebraicas, basadas en la definición de un conjunto de axiomas que relacionan operaciones entre sí; y las constructivas, que se crean a través del establecimiento de pre- y postcondiciones, definidas como expresiones lógicas. La técnica de especificación algebraica marca la primera aparición de uno de los temas recurrentes de la asignatura: el razonamiento inductivo. Por la propia naturaleza del método –donde las operaciones no son definidas explícitamente sino a través de las relaciones de unas con otras–, la esencia de las especificaciones algebraicas es el planteamiento del significado de las operaciones de forma recursiva. Un ejemplo sencillo es la definición de los naturales, definidos inductivamente a través de las operaciones *cero* y *sucesor*. Creemos que sería adecuado profundizar en este aspecto en las prácticas de la asignatura, pudiendo llegar a poner a prueba las especificaciones construidas. Por este motivo, proponemos un seminario práctico sobre especificaciones formales –con carácter optativo–, como detallamos más adelante.

Tema 2. Conjuntos y diccionarios. Bajo la denominación de "conjuntos y diccionarios" agrupamos en este tema y en el siguiente una amplia variedad de técnicas de representación, cuyo propósito final es la implementación eficiente de operaciones del tipo: inserta, suprime y elimina; siendo la particularidad del tipo diccionario que las consultas y eliminaciones se realizan por clave. Realmente, estas

técnicas también podrían verse como TAD independientes –por ejemplo, el TAD tabla de dispersión, o el TAD árbol B–, pero el planteamiento que adoptamos es la visión de las mismas como estructuras concretas que implementan un tipo de una abstracción mayor: el tipo conjunto. De esta forma se consigue una uniformidad conceptual entre técnicas tan dispares, y una clarificación de los objetivos comunes en todas ellas: ofrecer al usuario implementaciones eficientes de las operaciones de modificación y consulta, insistiendo en que el usuario debe ser ajeno a las cuestiones de representación de cada estructura particular. Cuando hablamos de eficiencia nos referimos tanto a tiempo de ejecución como al uso de la memoria; en este sentido, la importancia del factor eficiencia es otro de los temas recurrentes que aparece a lo largo de toda la asignatura.

El tema repasa en primer lugar los tipos abstractos conjunto y diccionario, y su representación sencilla con estructuras que los alumnos conocen de primero, como arrays y listas enlazadas, ordenadas o no. A continuación se centra en las tablas de dispersión, estudiando las técnicas básicas de resolución de colisiones. El análisis de los compromisos que se plantean en la implementación es otro de los conceptos que se repetirán de forma recurrente aquí y en adelante. En el caso de las tablas de dispersión, el compromiso surge entre el uso de memoria y el tiempo de ejecución: cuantas más cubetas, operaciones más rápidas pero se gasta más memoria; con menos cubetas, se usa menos memoria pero las operaciones son más lentas. Finalmente se introducen las estructuras de listas múltiples, u ortogonales, como una forma de representar relaciones muchos a muchos. El problema está también relacionado con la representación de matrices *escasas*, y tiene una aplicación directa en la representación de grafos con baja conectividad. Por estos motivos, creemos que su estudio es adecuado y está justificado.

Tema 3. Representación de conjuntos mediante árboles. Continuando en la línea del tema anterior, en este tema se tratan cuatro técnicas más de representación de conjuntos y diccionarios. La particularidad de todas estas técnicas es la utilización de estructuras arbóreas. Los alumnos han estudiado ya el TAD árbol y diversas implementaciones en primer curso, de manera que ahora se plantea la utilización de árboles para representar otro tipo abstracto. La mayoría de estas técnicas son independientes entre sí, por lo que el orden en que sean presentadas no es un factor relevante –excepto en el caso de los árboles B, que se pueden presentar como una evolución de los AVL, y por lo tanto deben ir después de estos–. El orden elegido para la presentación de las técnicas es el siguiente: árboles trie, relaciones de equivalencia, árboles AVL y árboles B.

Los tries, basados en la idea del árbol de prefijos, son estudiados en primer lugar, analizando su aplicación en problemas como los de un corrector ortográfico. Creemos que la discusión de las aplicaciones donde la técnica puede ser usada es una de las mejores formas de motivar el interés por el estudio de la misma. El siguiente punto del tema se sale en cierto modo del ámbito de las restantes técnicas, ya que trata del problema de buscar una representación eficiente de relaciones de equivalencia definidas sobre conjuntos. Esta técnica será usada posteriormente en el tema de grafos, por lo que su estudio está justificado. Tras analizar diferentes alternativas al respecto, se comprueba que la mejor solución consiste en una representación de árboles de punteros al padre. La técnica de los árboles AVL es planteada como una mejora de los árboles binarios de búsqueda, que los alumnos estudiaron en primer curso. Al mismo tiempo, los árboles B son presentados como una generalización de la idea de árbol binario de búsqueda a árbol de búsqueda n-ario. En todas las técnicas de árboles estudiadas aparece como un elemento común la necesidad de que los árboles estén balanceados, como un requisito para conseguir eficiencia. La forma de garantizar el balanceo dependerá de cada tipo de estructura.

Tema 4. Grafos. Aunque situado en la parte de estructuras de datos, este tema está centrado realmente en el estudio de problemas y algoritmos sobre grafos. Ya hemos justificado, no obstante, el interés de incluir este tema dentro de la primera parte de la asignatura. Los grafos son una herramienta

muy útil e interesante en el modelado de una gran cantidad de problemas, que en principio pueden no estar planteados como problemas de grafos. Saber describir estos problemas usando grafos es un paso previo para la aplicación de muchas de las técnicas de diseño de algoritmos, como las que se verán en la segunda parte de la asignatura. Por otro lado, los alumnos ya han estudiado grafos en la asignatura "Álgebra y Matemática Discreta" de primero, como vimos en el contexto curricular; respecto a los contenidos que allí estudiaron, debemos señalar que aquí estamos interesados en los problemas de representación e implementación.

Existe una serie de conceptos, problemas y algoritmos sobre grafos que aparecen de forma más frecuente en los temarios de distintas titulaciones, y que son señalados como básicos en las recomendaciones del informe CC2001. Entre estos tópicos podemos señalar: la representación de grafos; los recorridos (incluyendo búsqueda primero en profundidad y en anchura); el problema del árbol de expansión de coste mínimo (resueltos con los algoritmos de Prim y Kruskal); los problemas de caminos mínimos (resuelto con los algoritmos de Dijkstra y Floyd); y la idea de la ordenación topológica en grafos dirigidos acíclicos. En consecuencia, debemos considerar todos estos contenidos como de inclusión "obligatoria", aparte del indudable interés de los mismos.

Además, las recomendaciones dan margen para la introducción de otros problemas y algoritmos sobre grafos, no considerados de carácter esencial. De entre estos otros problemas, decidimos añadir los siguientes: puntos de articulación, componentes fuertemente conexos, ciclos de Euler, y problemas de flujo en redes. Todos los apartados del tema deben empezar con una ejemplificación de la utilidad de los conceptos presentados, con el objetivo de motivar su estudio. El tema acaba con la definición de otros problemas sobre grafos, como el problema del viajante, la coloración y el isomorfismo de grafos –aunque sin llegar a resolverlos aquí–. Esta definición da pie a la introducción de algunas ideas muy elementales de computabilidad: las clases de problemas P y NP, los problemas NP-completos, la reducción de problemas, el problema de explosión combinatoria, etc. No se pretende un desarrollo teórico extenso de estos conceptos, que serán objeto de la asignatura "Computabilidad".

Parte II - Algoritmia

Normalmente, el comienzo de la segunda parte de la asignatura coincidirá de forma aproximada con el inicio del segundo cuatrimestre. Pero aunque cambie el tipo de contenidos estudiados, es necesario mantener la visión de la asignatura como un todo. De esta manera, se sigue manteniendo el objetivo fundamental: ser capaces de resolver problemas de programación de forma eficiente, para lo cual entran en juego tanto las estructuras de datos como los algoritmos. Esta parte se divide a su vez en dos bloques, dedicados a los dos ámbitos claramente diferenciados de la algoritmia: el análisis y el diseño de algoritmos. Aunque no se incluye en el temario, podría ser adecuado intercalar una lección introductoria de repaso de conceptos fundamentales sobre algoritmos, análisis y diseño, sobre todo si el inicio se produce tras la vuelta del periodo de exámenes.

Bloque: Análisis de algoritmos

Por el volumen de contenidos propios del área de análisis de algoritmos, creemos necesario dividir este bloque en dos temas. En el primero se estudia el proceso de medición de recursos consumidos por un algoritmo, y su expresión usando las notaciones asintóticas de complejidad; en el segundo se presentan las técnicas de resolución de ecuaciones recurrentes, como algo que puede ser necesario en el análisis de algoritmos recursivos. El estudio de estos conceptos dentro de este bloque no es impedimento para que la cuestión de la eficiencia sea también tenida en cuenta dentro de los restantes temas. De hecho, ya hemos comentado cómo la eficiencia es una cuestión recurrente en toda la asignatura, puesto que en cualquier problema –ya sea de la parte de algoritmos o de estructuras de datos– una solución

correcta puede ser considerada como no viable si no garantiza un mínimo de eficiencia.

Tema 5. Análisis de algoritmos. Dentro de este tema se estudian las técnicas más habituales para medir y expresar la complejidad algorítmica. Los contenidos tratados requieren una buena fluidez en el manejo de conceptos matemáticos: sumatorios, series, límites, integrales, etc.; por lo que tradicionalmente suele resultar un tema más *áspero* para muchos estudiantes. No obstante, se supone que los alumnos ya estudiaron las bases del análisis de algoritmos en la asignatura "Metodología y Tecnología de la Programación" de primer curso. En este sentido, el tema debe profundizar en esos conocimientos. La profundización viene principalmente en dos sentidos: ser capaces de analizar algoritmos más complejos, con diferentes casos mejor y peor, y usando probabilidades para el caso promedio; y estudiar las notaciones asintóticas de complejidad de manera más formal, comprendiendo las distintas notaciones existentes, su definición, extensiones y forma de uso.

Para paliar la complejidad relativa de estos contenidos, resulta adecuado intercalar muchos ejercicios y ejemplos de los conceptos según van siendo introducidos, sin dejarlos para después de acabar el tema. De esta manera, la estrategia de enseñanza debe variar un poco con respecto a la utilizada en los restantes temas. También puede ser conveniente hacer un repaso de fórmulas o herramientas matemáticas no triviales, y que sean de utilidad en el posterior análisis de algoritmos.

Además de los contenidos expuestos, las recomendaciones del CC2001 añaden al núcleo básico de conocimientos el estudio de clases estándar de complejidad –es decir, enumerar los órdenes típicos que suelen aparecer, como $O(n)$, $O(n \log n)$, $O(n^2)$– y la capacidad de tomar medidas empíricas del rendimiento de los algoritmos. Ya hemos señalado que este segundo aspecto, desde nuestro punto de vista, debe ser abordado fundamentalmente en las prácticas de la asignatura.

Tema 6. Ecuaciones de recurrencia. Las técnicas de resolución de ecuaciones de recurrencia son herramientas necesarias en el análisis de algoritmos recursivos, y pueden ser de utilidad en el estudio de otros algoritmos no necesariamente recursivos. Refiriéndonos de nuevo al informe CC2001, la capacidad de resolver ecuaciones recurrentes es incluida en la base de conocimientos, indicando las necesidades de saber *"deducir relaciones recurrentes para describir la complejidad temporal de algoritmos definidos recursivamente"* y *"resolver ecuaciones recurrentes elementales"*. No está completamente claro qué tipo de ecuaciones se consideran elementales y cuáles no.

En la asignatura de programación de primer curso, MTP, los alumnos aprenden la resolución de ecuaciones recurrentes mediante la técnica intuitiva de la expansión de recurrencias. Profundizando en este tema, se describe la técnica de la ecuación característica, que permite resolver una amplia variedad de ecuaciones recurrentes lineales, homogéneas y no homogéneas. Consideramos adecuado introducir también otras técnicas que permiten resolver algunas ecuaciones no lineales, como la inducción constructiva o la técnica de transformación de la imagen. Todos estos métodos tienen un carácter netamente matemático, por lo que resulta adecuado no profundizar más de lo necesario; el límite de lo necesario está claro: aquello que realmente sea de utilidad después en el análisis de algoritmos.

Bloque: Diseño de algoritmos

En este bloque, que abarca todos los temas restantes hasta el final del curso, la asignatura aborda una serie de técnicas generales de diseño de algoritmos. En todos estos temas, la explicación es realizada fundamentalmente en base a ejemplos particulares de aplicación de las diferentes técnicas, es decir, algoritmos concretos. Sin embargo, hay que dejar claro que el interés no está tanto en los algoritmos desarrollados en sí mismos, sino en las técnicas generales de diseño, que permiten construir algoritmos para un conjunto muy variado de problemas. Es decir, el objetivo no es tanto aprender

una serie de algoritmos, sino conocer varias técnicas de diseño y adquirir la habilidad de usarlas en la resolución de problemas nuevos. En la práctica, este objetivo resulta mucho más complejo, ya que supone una mayor aportación por parte del alumno, que no es necesaria cuando simplemente debe memorizar una serie de contenidos. Pero el objetivo final de la asignatura no es el sencillo, sino el que persigue la meta de capacitar a los alumnos para resolver problemas de manera autónoma.

La ordenación temporal de los temas en el programa pretende seguir un nivel creciente de complejidad. Empezando con las ideas de más fácil asimilación y comprensión, se va avanzando hacia técnicas cada vez más complejas, algunas de las cuales requieren de las primeras para su resolución (como es, por ejemplo, el caso de ramificación y poda). Un orden similar al que hemos usado es asumido en la mayoría de los programas de las universidades españoles analizadas y en las recomendaciones internacionales, como se puede ver en las secciones anteriores. También la lista de técnicas a estudiar es fruto de un alto consenso, como ya analizamos. Sería posible flexibilizar algo el orden establecido, situando algunas técnicas en un lugar distinto en caso necesario. En particular, si se prevé una falta de horas para desarrollar todo el temario, proponemos que los temas de divide y vencerás y programación dinámica sean impartidos de forma simultánea y al final del programa. Por otro lado, el tema de resolución de juegos podría ser suprimido de forma completa.

Tema 7. Divide y vencerás. De acuerdo con los programas actuales, la técnica de divide y vencerás es estudiada en primer curso, en MTP, como un ejemplo de aplicación de la recursividad. Por lo tanto, se supone un conocimiento previo de los alumnos, que deberá ser ampliado y profundizado en este tema. En principio, los alumnos deben haber visto los métodos de ordenación basados en divide y vencerás (ordenación por mezcla y *quicksort*), aunque normalmente sin profundizar excesivamente en los mismos. Un repaso de estos métodos, como dos ejemplos típicos de aplicación de la técnica, resulta conveniente. Proponemos también la presentación de aplicaciones en algunos problemas de cálculo numérico, como la multiplicación rápida de enteros largos y la multiplicación de matrices.

Ya hemos comentado el carácter esencial del análisis de eficiencia en la asignatura. En este tema, como en los restantes de este bloque, se debe poner un énfasis especial en estudiar el tiempo de ejecución de los algoritmos diseñados. Los algoritmos que distintas técnicas aportan para un mismo problema se pueden contrastar en función de su complejidad computacional. En concreto, aquí la comparación surge entre las técnicas directas de resolución –usadas en los casos base–, y el tiempo de los algoritmos de divide y vencerás. Obviamente, si el algoritmo de divide y vencerás no mejora el tiempo de la solución directa, puede tener poco sentido aplicarlo.

Tema 8. Algoritmos voraces. La técnica de los algoritmos voraces[17] suele resultar fácil de asimilar y aplicar por los alumnos. La dificultad se encuentra, en la mayoría de los casos, en diseñar heurísticas de selección adecuadas que garanticen buenas soluciones, si no las óptimas. Por ello, resulta interesante incluir demostraciones de optimalidad en las explicaciones, siempre que sea posible. Aunque no es un objetivo del tema que los alumnos desarrollen demostraciones para todos los algoritmos que diseñen, se espera que adquieran cierta habilidad en la definición de buenas heurísticas.

En el tema se estudian en primer lugar algunas aplicaciones que dan lugar a algoritmos óptimos. Entre estos, se pueden mostrar a modo de repaso algunos algoritmos presentados en el tema de grafos, como los debidos a Prim, Kruskal y Dijkstra. A continuación se introduce la construcción de algoritmos heurísticos basados en la técnica de avance rápido. En la mayoría de los casos, la obtención de buenas soluciones requiere comprensión del problema y creatividad –aunque esta es inútil si no se tiene una sólida base conceptual para saber exactamente qué es lo que se pide, como ocurre muy a menudo–.

[17] También denominados algoritmos ávidos, codiciosos, o técnica de avance rápido; o en terminología inglesa *greedy*.

Fomentar la creatividad, ser capaz de evaluar mentalmente la adecuación al problema, y traducir las ideas a algoritmos, son objetivos complejos. Sólo mediante la presentación de una amplia variedad de problemas y soluciones se puede aspirar a desarrollar estas capacidades en los alumnos.

Tema 9. Programación dinámica. La programación dinámica es la única técnica general de diseño de algoritmos incluida en la asignatura que no aparece en el núcleo de conocimientos del informe CC2001, aunque sí dentro de las unidades opcionales. A pesar de que su aplicación, en términos generales, suele ser compleja y limitada, creemos que las ideas aportadas por la técnica –y sobre todo el contraste con divide y vencerás– hace interesante su estudio en la asignatura. Por otro lado, a nivel nacional, la mayoría de los programas de las distintas titulaciones españolas incluyen la programación dinámica entre sus contenidos, como se puede observar en el estudio realizado en el apartado 4.3.2. En definitiva, consideramos que su inclusión está suficientemente justificada. Su orden dentro del temario obedece al criterio ya expuesto de ir de menor a mayor complejidad; aunque también sería posible estudiarla después del tema de divide y vencerás.

Si en divide y vencerás los alumnos ven la idea de descomponer un problema en subproblemas, ahora se muestra que pueden existir distintas estrategias para aplicar esa descomposición: de forma ascendente o descendente. Algún ejemplo sencillo, como el cálculo de los números de Fibonacci, se puede usar para ilustrar esta idea. Por otro lado, la programación dinámica introduce la idea de utilizar tablas para almacenar resultados parciales de un problema, lo cual se puede generalizar a otras situaciones. Se presenta también el principio de optimalidad de Bellman, aplicable sobre un gran conjunto de tipos de problemas. Este principio se puede usar como la base para realizar demostraciones de optimalidad de algunos de los algoritmos presentados.

Tema 10. Backtracking. Al igual que con divide y vencerás, el programa de la asignatura MTP de primer curso incluye la técnica de backtracking[18] como un ejemplo de aplicación de la recursividad. Esto supone una ventaja en principio, pero a su vez puede ser un inconveniente: los alumnos asocian ambas ideas y les resulta difícil adaptarse a esquemas de backtracking no recursivos como los vistos en clase. Debe quedar claro que lo importante de backtracking es la idea de la técnica, independientemente del esquema usado, aunque normalmente las versiones iterativas serán más eficientes.

Aunque la idea del backtracking, así como los ejemplos presentados, suelan ser asimilados por los alumnos con más o menos facilidad, nos encontramos nuevamente con grandes dificultades en la aplicación sobre problemas nuevos. La raíz de la dificultad suele encontrarse en no tener claro cómo se representa la solución para determinado problema, lo que da lugar en muchos casos a largas divagaciones –sobre la forma del árbol de backtracking y la manera de generarlo– pero que carecen de sentido para el problema concreto que se pide[19]. Como hemos comentado antes, la exposición de una variedad de aplicaciones es la única manera de intentar fomentar en los estudiantes la habilidad de aplicar la técnica en problemas nuevos.

Tema 11. Ramificación y poda. Aunque la técnica de ramificación y poda[20] aparece en pocos libros de diseño de algoritmos, y es omitida en algunos planes de estudios de universidades españolas, el informe CC2001 la incluye dentro del núcleo de conocimiento básico. Este hecho justifica por sí mismo el estudio de la técnica dentro de la asignatura; es más, es posible y adecuado presentar la ramificación y poda como una generalización de backtracking, en cuanto que también se basa en un recorrido en

[18] Traducida en algunos sitios como "vuelta atrás" o "algoritmos con retroceso". A nuestro parecer, ninguna de ellas expresa exactamente la idea original, por lo que preferimos la denominación "backtracking".

[19] Como, por ejemplo, plantear el uso de un árbol binario en un problema de asignar n tareas a m trabajadores.

[20] También conocida como ramificación y acotación; o en terminología inglesa *branch-and-bound*.

un árbol implícito de soluciones. Este contraste resulta interesante desde el punto de vista docente, puesto que ofrece a los alumnos una base más amplia de técnicas relacionadas, entre las cuales se pueden establecer comparaciones o aplicar ideas de una en la otra.

A lo largo de la carrera, los alumnos volverán a estudiar más técnicas de resolución de problemas de optimización combinatoria, en las asignaturas de inteligencia artificial. En este sentido, el conocimiento de la ramificación y poda –y la visión que aporta sobre el concepto de espacio de soluciones– supone un buen punto de partida para el estudio de esas otras técnicas, basadas en principios similares. En cualquier caso, no se produce ninguna redundancia de contenidos, ya que aquí la ramificación y poda es tratada de forma más general y aplicada a problemas de cualquier tipo.

Tema 12. Resolución de juegos. En este tema se hace una breve introducción a las técnicas de resolución de juegos –específicamente, a los llamados "juegos de tablero"– presentando los conceptos de árbol de juego, la estrategia minimax y la poda alfa-beta. Aunque constituyendo una clase de problemas en una categoría distinta a los estudiados previamente, los problemas de resolución de juegos son asociados frecuentemente con las técnicas generales de diseño de algoritmos y tratados de forma conjunta. De hecho, la estrategia minimax es implementada usando backtracking, por lo que hay una continuidad natural en el desarrollo de los temas del curso.

No obstante, hay que tener en cuenta que la resolución de juegos encaja más propiamente en las asignaturas del área de inteligencia artificial. La profundización en los problemas de juegos requiere la introducción de conceptos que caen fuera del área de interés de la materia de los algoritmos y las estructuras de datos. En definitiva, este tema debe ser considerado como una introducción que permite dar una visión más amplia de los algoritmos, motivando a los alumnos en el estudio de técnicas avanzadas de resolución de problemas en general, y de juegos en particular. Por lo tanto, creemos que su explicación es conveniente pero, en caso de faltar tiempo en la planificación de la asignatura, es posible suprimir este tema. Los alumnos tendrán la oportunidad de estudiar estos contenidos en cursos posteriores.

4.4.3. Propuesta de materia práctica

Las prácticas de una asignatura son un componente fundamental de la misma, ya que dan a los alumnos la posibilidad de poner a prueba los conocimientos adquiridos en la teoría. Al mismo tiempo, las prácticas pueden ampliar o hacer énfasis en otros aspectos de la materia, distintos a los tratados en teoría. Pero las posibles actividades a desarrollar en las prácticas están limitadas, obviamente, por el número de créditos prácticos asignados en los planes de estudios.

A este respecto, recordamos una vez más la diferencia sustancial existente entre los planes vigentes en el momento de la publicación de la plaza objeto de concurso, y los planes actuales en la FIUM. Mientras que en los planes antiguos el número de créditos prácticos asignados a la asignatura AAED era de 2, en los actuales la asignatura AED dispone de 6 créditos prácticos. Por este motivo, distinguimos claramente entre la propuesta de prácticas para la asignatura AAED y la de la nueva asignatura AED.

Prácticas en la asignatura AAED

Como vimos en el apartado 2.2.3, los planes de Ingeniero en Informática de 1996 de la FIUM no sólo asignan 2 créditos prácticos a la asignatura semestral AAED, sino que añaden una asignatura trimestral posterior, "Laboratorio de Programación", de 6 créditos, entendida como el desarrollo práctico de los conocimientos adquiridos en AAED. Este contexto condiciona fuertemente las posibles

propuestas respecto a las prácticas de la asignatura objeto de estudio. Está claro que en la filosofía de los planes subyace la idea de que los créditos de AAED sean dedicados a la realización de prácticas de pizarra, mientras que las prácticas de programación deberían ser responsabilidad de "Laboratorio de Programación".

Independientemente de nuestra opinión al respecto de la conveniencia de hacer independientes teoría y prácticas –que ya expusimos en el apartado 2.2.3–, el programa de una asignatura no puede adoptar unilateralmente un enfoque contrario al resto de las asignaturas del plan en el que está incluida. Por este motivo, la propuesta de prácticas para AAED se basa fundamentalmente en las prácticas de pizarra. Además, proponemos la entrega y corrección de ejercicios en papel, para cada una de las partes de la asignatura.

Para cada tema de teoría, se entregará un **boletín de ejercicios**, que contendrá aproximadamente entre 20 y 30 ejercicios por tema. Estos ejercicios tienen distintos niveles de dificultad, variando entre los que pueden ser resueltos rápidamente en pocas palabras, y los que requieren un planteamiento más detenido. En todos los casos, los ejercicios están diseñados para ser planteados "en papel", mostrando la ejecución de alguna técnica estudiada, aplicando un concepto visto en clase, analizando un algoritmo de forma teórica, o desarrollando en pseudocódigo un algoritmo para algún problema propuesto. En el apéndice C se muestran algunos ejercicios de ejemplo de uno de estos boletines.

Gran parte de las actividades a desarrollar en las prácticas harán uso de estos boletines de ejercicios. En concreto, las actividades propuestas son las siguientes:

- **Prácticas de pizarra.** Al finalizar cada tema –o en los temas más largos, al acabar una parte– se dedicará una o varias horas de clase a la resolución de ejercicios. Los 2 créditos prácticos de la asignatura nos dan un total de 20 horas de problemas, lo cual equivale aproximadamente a unas 2 horas por tema. El objetivo de esta actividad es plantear y resolver en la pizarra algunos ejercicios de cada boletín, de entre los que se consideren más interesantes y novedosos. La solución de los ejercicios debe corresponder fundamentalmente a los alumnos, que se deben convertir en un elemento activo y participativo. Sólo de esa forma es posible comprobar que han comprendido bien la teoría, o por el contrario que existe algún error de aprendizaje que debe ser subsanado. La resolución de los ejercicios por parte del profesor limita la actividad de los alumnos al papel de copiar lo que se expone en la pizarra, reduciendo drásticamente la efectividad de estas prácticas.

  En consecuencia, conseguir la motivación de los alumnos para la participación en la resolución de los ejercicios es fundamental. En general, en una titulación como la nuestra esto es difícil de conseguir, y más aún en una clase masificada donde los alumnos se amparan en el anonimato del grupo y en el miedo a equivocarse para evitar salir a la pizarra. Por esto, proponemos varias medidas para fomentar la motivación: dar notas de clase a los alumnos que participen de forma activa, y señalar con antelación los ejercicios del boletín que pueden resultar más interesantes para las clases de problemas. Realmente, ambas medidas sólo tienen justificación en el intento de mejorar la participación de los alumnos. La nota asignada debe ser un refuerzo positivo, nunca negativo, entendido como una puntuación a añadir sobre la nota final. Explicaremos este aspecto en detalle dentro de la sección 8.4.

  Debemos señalar que la realización de estas prácticas de pizarra no evita la necesaria introducción de numerosos ejemplos y ejercicios resueltos en las clases de teoría. La diferencia es que mientras en las horas de teoría es el profesor el que toma el papel de actor principal en la resolución del problema –aunque posiblemente recibiendo las sugerencias de los alumnos–, en las clases de problemas deben ser los alumnos los que apliquen y expongan sus conocimientos. Analizaremos los objetivos y características de las prácticas de pizarra en la sección 8.2.2.

- **Entrega de ejercicios.** Aunque las prácticas de pizarra tienen la ventaja de la interactividad entre el profesor y el alumno, que permite un mejor guiado del proceso de aprendizaje, el inconveniente es la escasa participación que posibilita. En una clase con más de 80 alumnos, difícilmente intervendrán todos en apenas 20 horas de problemas. Por esto, proponemos la resolución y entrega de ejercicios por parte de los alumnos. Estos ejercicios serán algunos seleccionados por el profesor entre los propuestos en los boletines de cada tema. Los alumnos deberán resolverlos en papel de forma individual, con la posibilidad de consultar al profesor en horas de tutorías. El profesor corregirá los ejercicios, dando indicaciones sobre los posibles fallos cometidos en cada uno y valorando las respuestas con una nota. Finalmente, el ciclo se completará devolviendo a los alumnos los ejercicios corregidos.

  El último punto del proceso es fundamental, ya que el propósito de esta entrega de ejercicios no es tanto dar una nota a los alumnos sino ofrecerles una ayuda y una información sobre su progresión en la asignatura. Los alumnos podrán detectar sus fallos y sabrán dónde deben trabajar más, antes de enfrentarse al examen final. Por este motivo, es importante que la corrección no se limite a una simple nota o a tachar lo que esté mal, sino que se deben escribir las razones del fallo y la manera de corregirlo. Obviamente, esto incrementa enormemente el trabajo del profesor; y, como resultado, limita también el número de ejercicios que se proponen para la entrega, siendo razonables en cuanto a la dedicación que se puede asignar a esta tarea.

  En cursos anteriores, en la asignatura AAED, se establecían dos entregas de ejercicios: una para la parte de estructuras de datos y otra para la de algoritmos. Usualmente cada entrega constaba de unos 10 ejercicios, intentado variar los ejercicios asignados a diferentes alumnos. La valoración de los ejercicios cuenta con un pequeño porcentaje sobre la nota final; pero ya hemos comentado que el verdadero propósito de esta actividad es más bien la realimentación, que obtienen tanto los alumnos como el profesor.

- **Ejercicios opcionales.** Además de las anteriores actividades, proponemos la posibilidad de plantear a los alumnos algunos ejercicios o prácticas opcionales. Estas tareas están orientadas a los alumnos que tengan la intención de profundizar en la materia. Como ha venido ocurriendo otros años, pueden tomar la forma de problemas llamativos, no triviales, pero que pueden ser resueltos e implementados con una dedicación no superior a 15 horas, si se tiene la suficiente soltura. Esta actividad juega el papel de motivar a los alumnos más interesados en la materia, por lo que su realización es considerada como optativa. La carga de trabajo que supone no aconseja introducirla de forma obligatoria en una asignatura con unas características como las de AAED.

Prácticas en la asignatura AED

Recordemos que la asignatura "Algoritmos y Estructuras de Datos", de los planes de II de 2002, tiene asignados 6 créditos prácticos. Esta carga es más acorde con las características de una materia como la estudiada, y permite la realización de numerosas y diversas actividades, frente a lo que ocurría en los planes antiguos. Además, la disposición anual de la asignatura facilita un desarrollo paralelo de teoría y prácticas, lo que sin duda beneficia a ambas.

En el apartado 4.4.1 hemos comentado las principales decisiones sobre el diseño de la propuesta docente para la plaza objeto de concurso. Estas decisiones afectaban tanto a la parte teórica como a las prácticas de la asignatura. Vamos a recordar brevemente las decisiones más relevantes en referencia a la parte práctica de AED.

- **Enfoque mixto: imperativo-orientado a objetos.** Son numerosos los factores externos que aconsejan una introducción temprana de la programación orientada a objetos –cuyos beneficios

son indudables–, apareciendo en la mayoría de los planes de estudios nacionales e internacionales en segundo curso cuanto más tarde. Al mismo tiempo, los condicionantes internos, fundamentalmente los planes de estudios, sugieren un introducción algo más tardía de este paradigma. Coincidimos con las ideas expuestas en [García'01]: los fundamentos de la POO no deben formar parte de los objetivos educativos del primer curso de programación, sino que se deben presentar con el suficiente detalle en la asignatura específica de POO.

De esta forma, la asignatura AED de segundo curso sería un paso intermedio, donde se presentan los conceptos más básicos de la POO pero sin profundizar. En concreto, el enfoque mixto propuesto se caracteriza por lo siguiente: (1) se introducen los mecanismos de definición de clases y de manipulación de objetos, como formas adecuadas de implementar y usar tipos abstractos de datos[21], excluyendo de manera explícita todos los conceptos que surgen de la herencia; (2) la explicación, desarrollo y aplicación de los conceptos OO básicos tiene lugar fundamentalmente en las prácticas de la asignatura, presentando la parte teórica de manera independiente del lenguaje y el paradigma de programación usados en prácticas. En cierto sentido, podríamos hablar de una aplicación de la programación modular pero usando clases y objetos para construir los módulos.

- **Lenguaje de programación: C++.** Derivado principalmente de la anterior decisión, se elige para la realización de las prácticas el lenguaje de programación C++. Efectivamente, C++ permite una fácil *mezcla* de paradigmas y un uso *recortado* del lenguaje que incluya algunos conceptos pero omita lo relativo a la herencia[22]. La amplia utilización de C++ en asignaturas de otras titulaciones similares a la estudiada es un buen soporte para esta decisión. Además, ya hemos señalado algunas otras ventajas interesantes de C++ y que pocos lenguajes pueden igualar: posibilidad de definir tipos y funciones genéricos o parametrizados, utilización de mecanismos de asertos, amplia disponibilidad. Teniendo en cuenta que C++ es, casi, un superconjunto de C, proponemos que se enseñe en primer lugar el lenguaje C y a continuación se introduzcan las nuevas características de C++ que se consideren de interés (gran parte de las no relacionadas con OO, y algunas de las características OO). De esta manera se pretende evitar la complejidad del lenguaje C++, un problema que es advertido de manera bastante generalizada.

- **Distribución en módulos.** En la exposición de las principales decisiones de diseño del presente proyecto, se realizó una primera aproximación a la organización de las actividades prácticas. Estas fueron dispuestas en torno a los 4 siguientes grandes bloques:

  - Módulo I - Seminarios de programación
  - Módulo II - Implementación y manejo de estructuras de datos
  - Módulo III - Análisis de algoritmos
  - Módulo IV - Diseño de algoritmos

  El primero de los módulos se desarrolla de manera independiente de la teoría, y su objetivo es enseñar a los alumnos los lenguajes de programación C/C++, así como el entorno de programación usado, formado por las herramientas de edición y compilación que ofrece Linux. Los tres módulos restantes tienen por objetivo aplicar las técnicas estudiadas en teoría para la creación de estructuras de datos eficientes, para el análisis experimental de la eficiencia algorítmica y

[21] Entiéndase por "adecuada" el respeto de los dos principios básicos del concepto de TAD: encapsulación y ocultamiento de la implementación; que no son garantizados por los lenguajes imperativos.

[22] Algo que ha sido muy criticado para la enseñanza del paradigma OO, pero que consideramos una ventaja en una asignatura como AED.

para el diseño de algoritmos que resuelvan problemas nuevos. Estos módulos se desarrollan de manera simultánea y coordinada con la teoría, por lo que tratan de poner a prueba y mejorar las habilidades de aplicación de los conceptos teóricos. En este apartado detallaremos de forma más precisa las actividades que se plantean para estos módulos.

Además de estas decisiones generales, debemos mencionar las siguientes características sobre la propuesta de prácticas que presentamos:

- **Contenidos prácticos.** La mayoría de los contenidos de la asignatura –que enumeramos en extensión en la página 171– son responsabilidad tanto de la parte teórica como de las prácticas de la asignatura. No obstante, existen algunos tópicos claramente diferenciados que están más relacionados con la parte práctica, en los que deberían incidir de forma especial las actividades que se planifiquen. Los contenidos que consideramos más propios de la parte práctica son los siguientes:

  1. Bases de la programación orientada a objetos: el modelo OO; definición de clases; creación, uso y eliminación de objetos.
  2. Mecanismos de los lenguajes que dan soporte al uso de abstracciones; además de las clases y objetos, podemos señalar: definición de tipos genéricos o parametrizados, programación por contrato mediante pre- y postcondiciones.
  3. Especificación de tipos de datos básicos (como listas, pilas, colas, arrays, conjuntos y árboles) mediante especificaciones formales algebraicas.
  4. El proceso metódico de desarrollo de software: especificación, análisis, diseño, implementación y verificación.
  5. Análisis experimental de algoritmos: técnicas y herramientas matemáticas.
  6. Verificación y pruebas de software.
  7. Comparación de soluciones a un mismo problema usando diferentes técnicas, como avance rápido, programación dinámica, backtracking y ramificación y poda.

  La manera de presentar estos nuevos conceptos en prácticas será a través de los seminarios correspondientes. Todos los puntos pueden y deben ser introducidos de forma teórica, pero será fundamentalmente su aplicación práctica la que aporte un mayor aprendizaje a los alumnos. Por otro lado, está claro que los contenidos de prácticas no se limitan a estos aspectos, sino que incluyen también la mayoría de los tratados en teoría.

- **Trabajo en equipo.** En cualquier práctica de programación se plantea la cuestión de cómo organizar a los alumnos: ¿individualmente o en equipos?, ¿equipos grandes o pequeños? Prácticamente todas las recomendaciones curriculares disponibles señalan la importancia de fomentar las habilidades de trabajo en equipo. Por ejemplo, el informe CC2001 reconoce en sus principios la importancia de este tipo de habilidades, indicando que *"los estudiantes de informática necesitan aprender las mecánicas y dinámicas de la participación efectiva en un equipo, como parte de su educación universitaria"*, teniendo en cuenta que *"pocos profesionales de la informática pueden esperar trabajar de forma aislada por un largo periodo"*. Para conseguir estos objetivos, se recomienda dar oportunidades para el trabajo en equipo desde los primeros años.

  Por su parte, en el informe del consorcio Career Space, [CareerSpace'01] –analizado en el apartado 2.4.3–, la necesidad del trabajo en equipo aparece de forma recurrente en todo el documento. Este requisito es justificado también por el contexto laboral, donde *"los jóvenes graduados deben*

integrarse en equipos trabajando en problemas muy complejos con una estrecha interacción e interdependencia de varios componentes y aspectos de un sistema". El trabajo en equipo es asociado con las habilidades de comunicación, expresión y liderazgo. Se propone la creación de equipos multidisciplinares y multiculturales, algo que obviamente cae fuera de las posibilidades de actuación de las prácticas de AED. Las recomendaciones UNESCO-IFIP, [UNESCO'94], son más específicas sobre el tema e indican que *"en todos los módulos relacionados con software, se debe dar un extenso trabajo de laboratorio, ya sea como trabajo para casa o como proyectos en equipos bajo la supervisión de los profesores*". En concreto, en una asignatura similar a AED, denominada "Algoritmos y Programación Estructurada", se establece que las prácticas deben *"organizar a los estudiantes en equipos de programación"*.

Las principales objeciones al trabajo en equipo están relacionadas con la dificultad de evaluar individualmente a los alumnos y la posibilidad de que algún miembro del equipo se desentienda del trabajo, amparándose en el anonimato del grupo. Esto se puede reducir en parte formando equipos de tamaño reducido. En definitiva, creemos que el tamaño de grupo más adecuado para las prácticas de la asignatura AED es de 2 o 3 personas. Consideramos que los inconvenientes del trabajo en equipo son mucho menos relevantes que los beneficios que aporta respecto al trabajo individual.

- **Trabajo guiado y autónomo.** Otra cuestión fundamental sobre la planificación de las prácticas es el tipo de actividades a realizar. En unas prácticas de programación, la tarea fundamental será la implementación de un programa. Pero existen distintas posibilidades sobre el tipo de programas a realizar: un proyecto de programación de tamaño medio/grande, varios proyectos de tamaño medio para cada parte, o muchos ejercicios de tamaño pequeño; partir de cero en los programas o utilizar algún trozo de código dado por el profesor; trabajar bajo las indicaciones y supervisión directa del profesor o de manera menos guiada.

  A este respecto, podemos encontrar muchas menos sugerencias en las recomendaciones curriculares antes revisadas. Normalmente, la idea más común es la disposición de prácticas cortas y guiadas en el primer curso, y grandes proyectos en equipos en los cursos avanzados. De esta forma, la asignatura AED se situaría en una posición intermedia. Por un lado, es necesario fomentar el trabajo autónomo de los alumnos, que les ofrezca cierta libertad y ocasiones para analizar críticamente los problemas que se presentan, tomar decisiones por su cuenta, y organizar y planificar trabajos que requerirán un plazo de tiempo no inmediato. Por otro lado, la guía y apoyo del profesor es insustituible en el proceso de aprendizaje de los alumnos.

  En definitiva, creemos que en AED no es adecuado ningún extremo en cuanto al tamaño de las prácticas o al carácter guiado o autónomo de las mismas. Descartamos la realización de un gran proyecto anual no supervisado, así como la resolución de problemas pequeños sobre esquemas dados por el profesor. Proponemos una serie de prácticas de tamaño medio, coincidentes con cada bloque de la parte teórica: estructuras de datos, análisis de algoritmos, y técnicas de diseño de algoritmos. La duración de estas prácticas, medida en días naturales, estará sobre los dos meses, con una dedicación media estimada de los alumnos en torno a las 3 o 4 horas semanales[23]. Este tipo de prácticas son las que se conocen como "laboratorio abierto", cuyas características veremos en el Capítulo 8.

- **Proceso metódico de desarrollo de programas.** Aunque en el anterior punto hemos indicado

[23] Desafortunadamente, en muchos casos los alumnos no "sienten la necesidad" de hacer las prácticas hasta que los plazos empiezan a apretar, con lo que el tiempo estimado se acumula la semana antes de la fecha de entrega. Por este motivo sería conveniente introducir algún mecanismo de control o seguimiento de los alumnos.

que la tarea básica de las prácticas es la implementación de varios programas, realmente las actividades propuestas deben incluir todos los aspectos de un proceso metódico de desarrollo de programas: análisis, diseño, implementación, verificación y documentación. Unas prácticas como las de AED son una ocasión excelente para poner en práctica el proceso ingenieril en la construcción de programas. Las prácticas de primer curso están más orientadas a la programación de pequeños fragmentos de código que resuelvan problemas bien definidos, mientras que en los cursos avanzados de ingeniería del software se practicarán los pasos de análisis y diseño pero difícilmente se llegará a la implementación.

La propuesta de realizar prácticas de tamaño medio facilita la aplicación de este proceso metódico, frente a lo que ocurriría en el caso de hacer ejercicios pequeños, donde el análisis y el diseño tienen papeles muy secundarios. Los alumnos deben tener la suficiente libertad para tomar las decisiones que, en función de sus conocimientos, consideren más convenientes en cuanto a los algoritmos y estructuras de datos a utilizar. Las prácticas partirán del enunciado de un problema, intentando restringir lo menos posible la variedad de soluciones que se puedan dar para el mismo. En el diseño del programa se deben aplicar los criterios para la construcción de software de calidad: descomposición modular adecuada, respeto de los principios de encapsulación y ocultamiento de la información, eficiencia, etc.

Igual que con los contenidos propios de OO, se debe hacer una presentación inicial de los conceptos básicos en la teoría, para desarrollarlos después en profundidad en las prácticas. La forma de introducir estos conceptos en prácticas será a través de los seminarios y en las sesiones adicionales de explicación de las prácticas. Además, los requisitos de las prácticas deben incluir una memoria de cómo ha sido llevado a cabo el proyecto, detallando cada etapa de forma separada. Proponemos añadir a la documentación lo que denominamos un **informe de desarrollo de las prácticas**, para el cual se deberán rellenar unas tablas como las mostradas en la tabla 4.5. Esta tabla está basada en las ideas expuestas en [Humphrey'01], donde se sugiere la utilidad de documentar el propio proceso de desarrollo de programas, a través del rellenado y análisis de unas tablas en las que se mide el tiempo dedicado a cada etapa del proceso.

Proyecto:				**Fecha de inicio:**	
Programador:				**Fecha de fin:**	
Día/Mes	**Análisis**	**Diseño**	**Implement.**	**Validación**	**TOTAL**
TOTAL (minutos)					
MEDIAS (porcentaje)					

Tabla 4.5: Tabla de dedicación personal a las distintas etapas del proceso de desarrollo de software, basada en las propuestas en [Humphrey'01], con simplificaciones. La tabla mide tiempos (en horas o minutos).

Estas tablas son una manera conveniente de fomentar y hacer explícito el proceso metódico e

ingenieril de construcción de programas, que de otra forma queda como una idea vaga, muy genérica y de aplicación no directa.

- **Actividades opcionales.** Durante el análisis del contexto personal del alumnado, en la sección 2.3, señalamos la existencia de un porcentaje de alrededor del 15 % de los alumnos que aspiran a la máxima nota posible en la asignatura, frente al 60 % que se conformaría con el aprobado. La asignatura debe garantizar unos conocimientos mínimos para todos los alumnos, a la vez que ofrecer posibilidades de expansión para aquellos con mayor interés por la materia. Pensamos que la mejor manera de incentivar a los alumnos para lograr las más altas cotas, fomentando una implicación y participación *extra* en la asignatura, es a través de la propuesta de actividades prácticas con carácter opcional. La valoración de estas actividades puede realizarse ofreciendo una puntuación adicional en la nota final, o la convalidación por algún ejercicio del examen. Trataremos esta aspecto en la sección 8.4.

  En particular, proponemos para la asignatura AED las siguientes actividades opcionales:

  - **Seminario de especificaciones formales algebraicas.** Resulta muy interesante la posibilidad de poner en práctica el método de especificaciones algebraicas estudiado en la teoría de la asignatura. Existen de hecho herramientas que permiten la ejecución de las especificaciones, descritas en lenguajes algebraicos como OBJ y Maude. Con estas herramientas es posible definir el funcionamiento de los TAD básicos, y simular la ejecución de expresiones de ejemplo. Los alumnos pueden practicar un paradigma de programación distinto al imperativo, y desarrollar la capacidad de razonamiento inductivo, subyacente en la filosofía del método axiomático. Proponemos que esta práctica opcional sea convalidable por los ejercicios de examen correspondientes al Tema 1 de teoría.
  - **Resolución de problemas.** En la segunda parte de la asignatura se estudian muchas técnicas de diseño de algoritmos. Algunas son más complejas, y es difícil que las prácticas de la asignatura incluyan ejemplos de aplicación de todas ellas. Por este motivo, proponemos que se realice una práctica opcional que trate sobre alguna de estas técnicas avanzadas, como la ramificación y poda o las técnicas de resolución de juegos. Igual que antes, esta práctica sería convalidable por algún ejercicio de examen correspondiente a estas técnicas.
  - **Participación en el ACM Contest.** Ya comentamos en la página 62 las características de este concurso, y el interés de fomentar la participación de los alumnos en la fase local murciana y en el curso de promoción educativa de preparación que ofrece la Facultad. Este interés se debe tanto a la coincidencia de objetivos (construir programas que resuelvan problemas) como de métodos (uso de técnicas generales de diseño de algoritmos). Para esta actividad opcional proponemos que la evaluación consista en una puntuación a añadir a la nota final.
  - **Partes opcionales en las prácticas.** Adicionalmente, en las prácticas obligatorias de la asignatura se pueden incluir ciertos requisitos extra, pero no necesarios para superar las prácticas. Estos serían lo que típicamente se denominan "ejercicios para subir nota", y cuya realización puede contar entre el 30 % y el 20 % de la nota final de prácticas.

  Es conveniente señalar dos cuestiones. Primero, por el carácter optativo de estas actividades, no son necesarias para aprobar la asignatura o para aspirar una buena nota. Segundo, el tipo de actividades propuestas puede variar de un año a otro, según las circunstancias externas y los intereses que muestren los alumnos.

- **Clases de problemas.** En la propuesta de prácticas para la asignatura AAED indicábamos los beneficios de las clases de problemas, también conocidas como prácticas de pizarra. Seguimos manteniendo la convicción de que este tipo de actividades es fundamental en una asignatura como AED, puesto que es muy difícil garantizar que las prácticas de programación aborden todos los conceptos estudiados en teoría. Las prácticas de pizarra desempeñan un papel fundamental en la consolidación de los conocimientos teóricos y, por ello, no deben ser omitidas en la asignatura. No obstante, la reducción global de créditos –que ya comentamos con anterioridad en la página 89– ofrece un menor margen para la realización de ejercicios en clase. Aun así, se debe reservar un número suficiente de horas para clases de problemas, que pueden tener lugar en horas de teoría o bien de prácticas, al acabar los seminarios.

  Igual que comentábamos para AAED, se entregará para cada tema de teoría un boletín de ejercicios típicos para ser resueltos en papel, sacados de exámenes anteriores, de algún libro o ideados específicamente por el profesor. Un ejemplo de estos boletines se puede ver en el apéndice C. Las prácticas de pizarra consistirán en el planteamiento y resolución de algunos de estos ejercicios. La participación activa de los alumnos es esencial para que esta actividad pueda conseguir sus objetivos formativos.

En definitiva, la planificación de las actividades prácticas que se proponen para la asignatura AED es mostrada en la tabla 4.6.

	Tiempo presencial (horas)	**Dedicación alumnos (horas aprox.)**	**Créditos equival. (aprox.)**
ALGORITMOS Y ESTRUCTURAS DE DATOS	**28**	**100**	**6**
SEMINARIOS DE PROGRAMACIÓN	**14**		**1,4**
Seminario 1. Programación en C	8		0,8
Seminario 2. Programación en C++	6		0,6
Seminario 3. Especificaciones formales algebraicas	2*	4*	-
PRÁCTICAS DE PROGRAMACIÓN	**6**	**100**	**3,8**
Práctica 1. Análisis y diseño de estructuras de datos	2	35	1,3
Práctica 2. Eficiencia, evaluación y predicción	2	30	1,2
Práctica 3. Resolución de problemas	2	35	1,3
PRÁCTICAS DE PIZARRA	**8**		**0,8**

* No se suman al total, al tratarse de prácticas optativas.

Tabla 4.6: Planificación de las prácticas de AED. Se cuenta aproximadamente 1 crédito por cada 10 horas presenciales y por cada 28 horas de trabajo no presencial (columna "Dedicación alumnos").

En este caso, la estimación del número de créditos es menos precisa ya que hay que tener en cuenta tanto las actividades de carácter presencial como la dedicación de los alumnos a los proyectos de programación, que tienen un carácter no presencial. En general, se admite que para las primeras debe tomarse la equivalencia de 10 horas a 1 crédito práctico –establecida en las directrices generales de planes de estudios–, mientras que para las actividades prácticas no presenciales el número de horas es mayor. No hay un criterio claro y definido para el segundo caso, pudiéndose encontrar en algunos sitios la equivalencia de 1 crédito con 25 horas y en otros sitios con 30. En las estimaciones que aparecen en la tabla 4.6 se ha tomado de forma aproximada la asignación de 1 crédito práctico por cada 28

horas[24]. En cualquier caso, consideramos que el volumen de trabajo es acorde con la carga asignada por los planes de estudio para la asignatura.

A continuación vamos a describir de forma más detallada los aspectos básicos de los seminarios así como de las prácticas de programación. Describiremos de manera más detallada las sesiones que componen los seminarios dentro del Capítulo 7.

Seminarios de programación

Los seminarios de prácticas son una actividad de carácter presencial, desarrollada en los laboratorios de prácticas. El objetivo es presentar a los alumnos un lenguaje, técnica o herramienta concreta, siguiendo un guión de prácticas. En algunos momentos del seminario se deberá realizar una explicación o presentación de los contenidos por parte del profesor, y en otros será la actividad de los alumnos sobre las máquinas lo que prime.

Los seminarios han sido descompuestos en sesiones de 2 horas, de acuerdo con la asignación de laboratorios realizada habitualmente en la FIUM. Estas sesiones se realizan semanalmente. En principio, en una disposición anual de la asignatura, esto daría lugar a tener finalizados todos los seminarios poco antes de las vacaciones de Navidad.

Consideramos que estos seminarios deben ser consideramos como no obligatorios para superar la asignatura, aunque sí bastante aconsejables para aquellos que no conozcan los lenguajes C o C++.

Seminario 1. Programación en C. Según los programas de la asignatura MTP de primer curso, los alumnos que llegan a AED han estudiado y utilizado el año previo el lenguaje Modula-2. Algunos estudiantes de años anteriores pudieron haber aprendido Java, que fue enseñado en algunos grupos de primero en el curso 2001/02 y algunos anteriores. El aprendizaje del lenguaje C también es útil en otras asignaturas de segundo curso y posteriores, como por ejemplo "Sistemas Operativos"; a su vez en esta asignatura se enseñará, simultáneamente, el manejo del sistema operativo Linux, en el que se llevarán a cabo las prácticas de la asignatura.

El seminario de programación en C se compone de cuatro sesiones de laboratorio. En cada una de ellas se introducirán algunos conceptos del lenguaje o de las herramientas disponibles, se dará tiempo a los alumnos para aplicar los conceptos estudiados y se dejarán ejercicios propuestos. Estos ejercicios contendrán pequeños problemas de programación, que deberían ser resueltos por los alumnos antes de la siguiente sesión. Sería muy aconsejable hacer un seguimiento del resultado de estos ejercicios, ya sea a través de la entrega de los mismos, o la corrección en clase antes de comenzar la siguiente sesión. Además del lenguaje y las herramientas, se deberá hacer énfasis en la idea de la modularidad, y la manera de conseguirla en C, observando la imposibilidad de garantizar el principio de ocultación de la implementación.

Como resultado del primer seminario de prácticas, se espera que los alumnos sepan programar en C con más o menos soltura, y que sepan manejar las herramientas de edición y compilación en Linux, así como la herramienta *make* para la automatización del proceso de compilación en proyectos grandes. Suponemos que, al mismo tiempo, los alumnos aprenderán en la asignatura "Sistemas Operativos" el manejo básico de Linux y el uso de herramientas de depuración.

Seminario 2. Programación en C++. Como ya hemos señalado, el objetivo de este seminario no es presentar todas las características del lenguaje C++ sino un conjunto reducido. Por este motivo, creemos que con tres sesiones de laboratorio es suficiente para desarrollar los contenidos seleccionados.

[24]Debemos notar también que la aproximación proviene del hecho de que la dedicación no presencial de los alumnos sólo puede ser estimada de forma subjetiva e imprecisa.

Obviamente, este seminario es impartido después de finalizar el aprendizaje de C. Los alumnos tienen una visión reciente de las posibilidades y limitaciones del lenguaje C, de manera que pueden apreciar las mejoras introducidas en C++.

En la presentación de C++ se distinguen dos tipos de modificaciones sobre C: las relacionadas y las no relacionadas con la programación OO. En la primera sesión se muestran las características de C++ que no implican ningún conocimiento de POO, como los operadores para el manejo de memoria dinámica o el paso de parámetros por referencia. La mayoría de las modificaciones suponen interesantes ventajas frente a los mecanismos alternativos disponibles en C. Las dos restantes sesiones están dedicadas a la introducción de los mecanismos de C++ que permiten la programación OO, aunque a nivel muy básico: definición de clases, creación, eliminación y uso de objetos. También se explican otros conceptos avanzados de C++, como los asertos (aunque también están disponibles en C) y la posibilidad de definir tipos parametrizados. Sería interesante que estos mecanismos fueran aplicados en las prácticas de la asignatura.

Igual que en el primer seminario, se incluye la realización de ejercicios en las horas del seminario, así como la propuesta de otros problemas en los guiones de prácticas. Teniendo en cuenta que estamos en un nivel más avanzado, es necesario incluir muchos ejemplos en los guiones de prácticas. El objetivo final del seminario es que los alumnos sean capaces de programar en C++, entendiendo los conceptos más básicos de la programación OO aunque sin utilizar la herencia. Las prácticas deben servir de base para la asignatura POO de tercer curso. Los alumnos aplicarán de forma práctica los conceptos de clases y objetos, aunque con la posibilidad de usarlos en programas imperativos basados en descomposición funcional.

Seminario 3. Especificaciones formales algebraicas. El objetivo de este seminario es poner en práctica las ideas estudiadas en el primer tema de teoría, en relación al método formal de especificación algebraico o axiomático. Si en clase se estudiaron los fundamentos teóricos del método, basados en establecer la semántica de las operaciones a través de la relación de unas operaciones con otras, este seminario dará la oportunidad de poner en funcionamiento las especificaciones y comprobar la utilidad de la técnica. Esta posibilidad es muy interesante, ya que al contrastar las especificaciones formales con las informales se señala que una de las ventajas de las primeras es la posibilidad de "ejecutar" las especificaciones, y de esa forma crear prototipos antes de llegar a la implementación.

Para la realización de este seminario se propone la utilización del lenguaje algebraico Maude, basado en el lenguaje OBJ. Las principales razones que justifican la elección de este lenguaje son las siguientes:

- El formato utilizado por el lenguaje Maude es muy parecido al estudiado en teoría, lo que facilita la conversión de las especificaciones planteadas en clase. Mientras que la notación presentada en teoría para la especificación de un TAD consta de las partes: nombre del TAD, conjuntos de tipos usados, sintaxis de las operaciones y semántica; el formato usado por Maude tiene los componentes: nombre del módulo y del TAD, módulos y tipos usados, sintaxis de las operaciones y semántica. En ambos casos, la semántica se establece a través de la definición de una serie de axiomas, que indican la igualdad entre dos expresiones del tipo.

- Existen varias herramientas de ejecución de especificaciones en Maude que están disponibles de forma pública y gratuita[25]. Hay un grupo de investigación de cierta relevancia que da soporte a estas herramientas, y trabaja en la mejora y elaboración de nuevas versiones. Como consecuencia

[25]Las herramientas se pueden descargar en la dirección: `http://maude.cs.uiuc.edu/`, donde se puede encontrar más información sobre las investigaciones actuales en relación a las especificaciones algebraicas, y en especial usando Maude.

de esto, existe numerosa documentación disponible y son varias las universidades españolas que utilizan esta herramienta en algunas de sus prácticas (por ejemplo, como vimos en el apartado 4.3.2, en la Universidad de Málaga).

Aunque el intérprete incluye una gran variedad de posibilidades no triviales, como la definición de tipos parametrizados, es posible reducir al máximo los comandos necesarios para realizar un uso básico: escribir especificaciones, simular la ejecución de expresiones de ejemplo sobre los tipos especificados, y almacenar los resultados obtenidos. La explicación de las características básicas de la herramienta cabe de forma sobrada en una sesión de laboratorio.

A pesar de los numerosos beneficios señalados, creemos que la racionalidad de la carga asignada a la asignatura AED no aconseja ni permite un desarrollo demasiado extenso de esta práctica. Es más, proponemos, en principio, que esta práctica tenga carácter optativo; la realización de la misma será convalidable por las preguntas del examen correspondientes al Tema 1 de teoría. De esta forma, aunque la práctica no es obligatoria, es muy aconsejada y está bien valorada para los alumnos[26]. La superación de la práctica de especificaciones algebraicas supone la asistencia obligatoria al seminario (una sola sesión) y posteriormente la entrega de algunos ejercicios resueltos, consistentes en especificar en Maude algunos tipos y operaciones, y probar expresiones de ejemplo. En el apéndice C.3 mostramos los ejercicios propuestos para este curso.

Prácticas de programación

Las prácticas de programación son, sin duda alguna, la parte central de las prácticas de la asignatura AED. Estas consistirán en la realización de proyectos de programación de tamaño medio en los que se apliquen las técnicas de análisis y diseño de algoritmos y estructuras de datos estudiadas en teoría. Los proyectos requerirán la aplicación de todos los pasos del proceso de resolución de problemas, que después se deberán documentar en la memoria entregada. Ya hemos comentado el interés de que en estas prácticas predomine el trabajo autónomo de los estudiantes, desarrollándose en la modalidad conocida como laboratorio abierto. Los alumnos reciben el enunciado de un problema, que deben resolver en horas de prácticas o en el tiempo de que dispongan. No obstante, es importante explicar cuáles son los aspectos más importantes de lo que se pide, dar algunas ideas iniciales y aclarar las dudas que puedan surgir a los alumnos. Por este motivo, en la tabla 4.6 se han asignado dos horas presenciales para estas prácticas.

Las prácticas se realizarán en grupos de dos o tres, creados por los propios alumnos. De esta forma se pretende fomentar el trabajo en equipo, con todas las ventajas que ello implica. Es importante que se documente en la memoria de prácticas la coordinación y el reparto de tareas entre los miembros del grupo, entendiendo que en proyectos como los que se proponen la distribución de tareas y responsabilidades será lo más habitual –frente al trabajo simultáneo de dos personas en la misma máquina–. Tras la entrega de la memoria de prácticas, se deberá realizar una entrevista con los alumnos, en la que tendrán que exponer y defender el trabajo realizado.

La evaluación de las prácticas no sólo se debe basar en el funcionamiento o no de la misma, sino que debe tener en cuenta y valorar la consecución de los criterios de calidad del software:

- **Análisis y diseño.** Se valorará la calidad y adecuación del diseño y el análisis realizados, y la dedicación a estas fases previas a la implementación.
- **Modularidad.** La funcionalidad debe estar bien repartida entre los módulos. Debe estar claro el sentido y la responsabilidad de cada módulo. Se debe respetar el principio de ocultación de la implementación.

[26] Este curso 2003/04, por ejemplo, han realizado la práctica algo más del 50 % de los alumnos matriculados en AED.

- **Abstracciones.** Se deben identificar los tipos abstractos que aparecen, e implementarlos usando clases de forma adecuada.
- **Uso del lenguaje.** El código deber ser claro, legible, robusto y eficiente. No se deben crear procedimientos muy largos y complejos. Se valorará el uso de clases genéricas (plantillas) y precondiciones / postcondiciones (asertos).

Las características de las tres prácticas propuestas son similares, excepto en el caso de la segunda, más centrada en el análisis experimental de algoritmos que en la implementación de programas. Vamos a analizar detalladamente los objetivos y características de cada una de estas prácticas por separado.

Práctica 1. Análisis y diseño de estructuras de datos. El objetivo de esta práctica es aplicar las técnicas de diseño de estructuras de datos estudiadas en la primera parte de la asignatura. Pero pensamos que la labor de los alumnos no debe limitarse exclusivamente a implementar una serie de estructuras de datos establecidas de antemano por el profesor. Los alumnos deben ser capaces de analizar qué necesidades de almacenamiento de información requiere determinada aplicación, diseñar una serie adecuada de tipos de datos capaces de cubrir esas necesidades, y seleccionar la estructura de datos más adecuada para cada tipo.

Analizando los temas de la parte de estructuras de datos, podemos distinguir, a grandes rasgos, dos ámbitos de problemas en los que se pueden realizar las prácticas: técnicas eficientes de representación de conjuntos y diccionarios, y representación y algoritmos sobre grafos. En el primero, encontramos las tablas de dispersión y diversas estructuras arbóreas. Todas ellas comparten el objetivo último de ofrecer implementaciones eficientes de operaciones para la inserción, eliminación y consulta de cierto tipo de elementos contenidos. El segundo ámbito, el de los grafos, está realmente más relacionado con los problemas de diseño de algoritmos que con las cuestiones propias de la estructuración de datos. Esto puede dar lugar, en principio, a dos tipos de prácticas: (1) una aplicación que requiera un uso intensivo de almacenamiento y recuperación de información; (2) un problema que deba ser modelado mediante grafos y resuelto tomando como base los algoritmos de grafos estudiados. También sería posible plantear alguna práctica que implicara ambos componentes.

Podemos poner como ejemplo del tipo de problemas a plantear en estas prácticas, la aplicación propuesta para el presente curso. El enunciado de la práctica se puede encontrar en el apéndice C.4. Básicamente, la idea de la práctica es la implementación del motor de búsqueda para un buscador web. Disponemos de un conjunto muy grande de páginas web, de varios miles o decenas de miles, cada una de las cuales puede contener varios cientos de palabras. El objetivo es ofrecer una implementación eficiente para las operaciones de búsqueda de páginas por palabras. Las operaciones de búsqueda pueden adoptar diversas formas: (BASICA) encontrar las páginas en las que aparezca una palabra; (AND) encontrar las páginas donde aparezcan todas las palabras pasadas como parámetro; (OR) encontrar las páginas donde aparezca alguna de las palabras pasadas como parámetro; (COMILLAS) encontrar las páginas que contengan todas las palabras pasadas como parámetro y además consecutivamente en el mismo orden. Se pueden añadir requisitos adicionales a la práctica: permitir expresiones de búsqueda que combinen distintos operadores, añadir un operador NOT, indicar en los resultados un fragmento de la página donde aparezca la palabra, etc. No obstante, conviene limitar el tamaño de la práctica, y estas partes aparecen como optativas.

Evidentemente, la solución del problema del buscador no se puede basar en recorrer todas las páginas para cada consulta. Es necesario tener precalculadas las consultas de alguna manera, de forma que la operación elemental de buscar una palabra se pueda hacer eficientemente. Esto dará pie a cuestionarse qué información se necesita, cómo organizarla y qué estructuras de datos utilizar. Puede ser necesario implementar más de una estructura de datos, ya que aparecen distintas necesidades.

En el enunciado de la práctica se especifican los apartados que debe contener la memoria de prácticas entregada por los alumnos, que son los siguientes:

- **Portada.** Nombre de los alumnos, y número de prácticas de cada uno.
- **Análisis del problema.** Por análisis se entiende el estudio preliminar del problema e incluye:
  - Estudio de los requisitos propios del problema tratado. Definir las partes de la especificación que sean ambiguas o incompletas. Decidir las partes opcionales que se realizan
  - Estudio de los tipos abstractos necesarios para resolver el problema y las operaciones sobre los mismos. Análisis de las diferentes alternativas que se presentan para la implementación de estos tipos
- **Diseño de la aplicación.** El diseño es la toma de decisiones respecto a la estructura del programa a implementar e incluye:
  - Descomposición modular del programa. Qué módulos existen. Cuál es la responsabilidad de cada uno. Qué relación de uso hay entre los módulos.
  - Especificación de los tipos abstractos. Especificar detalladamente los principales tipos abstractos de datos de la aplicación. Se puede usar un formato propio de especificación (formal o informal).
  - Estructura de datos para cada tipo de datos. Decidir qué estructura de datos se va a utilizar para cada tipo, justificando la decisión. Particularizar las características generales de la estructura para el problema concreto, indicando posibles modificaciones y adaptaciones.
  - Algoritmos sobre los tipos de datos. Descripción abstracta de los algoritmos utilizados, correspondientes a cada operación sobre el buscador.
  - Documentar cualquier otra decisión de diseño que pueda resultar de interés.
- **Listado del código.** Entregar un listado del código, comentado, incluyendo ficheros de cabecera, de implementación y el fichero `makefile` necesario para compilar el programa.
- **Informe de desarrollo.** El informe de desarrollo es un documento sobre el proceso personal y del grupo para la resolución de la práctica. Debe contener lo siguiente:
  - Organización temporal de las fases de resolución (análisis, diseño, implementación, pruebas), no de forma genérica sino para esta práctica en concreto.
  - Cómo ha sido la coordinación y el reparto del trabajo entre los miembros del grupo. Concretar personas y tareas realizadas.
  - Tablas de dedicación personal en las distintas fases del trabajo. Se utilizarán tablas como la mostrada en la tabla 4.5, rellenadas con el mayor rigor posible.
  - Conclusiones y valoraciones personales.

Otras posibles ideas que se podrían usar para el enunciado de estas prácticas serían, por ejemplo, los problemas relacionados con los correctores ortográficos (representación de muchas palabras, gran cantidad de prefijos comunes, posibilidad de hacer sugerencias, auto-completar, añadir palabras, etc.), aplicaciones de planificación de rutas (orientado al tema de grafos, con la posibilidad de diversos tipos de operaciones, pero con posibilidad de incluir información adicional), aplicaciones de estilo base de datos (algún sistema que necesite almacenar y recuperar información de manera eficiente, y según diversos criterios).

Práctica 2. Eficiencia, evaluación y predicción. El objetivo de esta segunda práctica es aplicar las técnicas de análisis de algoritmos sobre programas construidos por los propios alumnos. La práctica hace énfasis especial en un área que recibe poca atención en la teoría de la asignatura, como es el análisis experimental de algoritmos. El estudio experimental tiene sus propios mecanismos, algunos similares a las técnicas aplicadas en el estudio teórico y otros no tanto. Mientras que las clases de teoría de la asignatura tratan las técnicas de análisis a priori de algoritmos, las prácticas profundizan en el estudio empírico a posteriori.

La práctica se desarrolla al mismo tiempo que se explica el bloque de análisis de algoritmos en clase, o con un pequeño retraso. En principio, el análisis experimental de algoritmos se basará en un análisis teórico previo, tras el cual se obtiene una fórmula, o varias, para el tiempo de ejecución y la memoria. Estas fórmulas expresan el crecimiento del tiempo y la memoria en función del tamaño de la entrada, para los posibles casos, y con unas constantes indefinidas. Los alumnos deben saber identificar los parámetros que determinan el tamaño del problema, diferenciar el comportamiento en los casos peor y promedio, y ajustar una función a una serie de datos obtenidos a través de experimentos. El último de los puntos implica el conocimiento y uso de técnicas y herramientas propias de la estadística. Entre las técnicas estadísticas que puede ser interesante aplicar podemos señalar los siguientes:

- Diseño de un conjunto de casos de prueba adecuado, especificando los tamaños y número de pruebas a realizar, de manera que se permita una obtención precisa de los parámetros buscados.
- Técnicas estadísticas de interpolación y ajuste de regresión, para el ajuste de una muestra de puntos a una función con forma conocida.
- Realización de gráficas, en dos y tres dimensiones, que muestren la información relevante.
- Análisis de residuales y tests de aleatoriedad, para la verificación de los ajustes estadísticos realizados sobre datos con formas conocidas o no.

La última categoría de técnicas puede ser precisa cuando se parte de un programa compilado del que no se dispone del código fuente, y por lo tanto no se puede conocer la forma de la función que indica el tiempo. También se puede usar para contrastar si la función obtenida en el estudio teórico era correcta o no.

El programa sobre el cual realizar el análisis es otra cuestión de relativa importancia. El programa podría ser alguno creado específicamente para esta práctica, aplicando alguna de las técnicas estudiadas; aunque en ese caso requeriría una labor de programación adicional. Otra posibilidad interesante aplicar el análisis sobre el programa desarrollado para la primera práctica. Por ejemplo, la práctica para el presente curso 2003/04 consistirá en medir la eficiencia del buscador web en función de diversos parámetros, midiendo el tiempo de búsqueda y la memoria usada en función de: el número de páginas, el número de palabras por página, el número total de palabras distintas, el tamaño de la búsqueda en cuanto al número de palabras, el número de páginas resultantes de la búsqueda.

Para llevar a cabo la práctica puede ser necesario realizar un conjunto considerable de ejecuciones de los programas, anotando tiempos y memoria usada. Por ello, sería conveniente escribir programas que automaticen el proceso de realización de las pruebas y recopilación de resultados (programas que usan otros programas). Hay, en consecuencia, una labor de programación, aunque creemos que no debe ser el núcleo de esta práctica. Existen otras tareas que deberían ser desarrolladas por los alumnos en esta práctica, relacionadas con los propósitos del análisis experimental de algoritmos:

- Estudiar y establecer objetivos de medición. Establecer qué parámetros deben ser medidos; fundamentalmente estaremos interesados en el tiempo y la memoria, pero pueden ser distintos

trozos de los que se mide el tiempo (por ejemplo, en el buscador podría ser el tiempo de búsqueda y el tiempo de inicialización del buscador). Determinar los parámetros relevantes que indican el tamaño del problema y cuáles deben ser mantenidos fijos. Identificar los casos peor y promedio.

- Diseño de casos de prueba. Teniendo en cuenta los objetivos, determinar qué casos pueden ser mas útiles para obtener unos resultados significativos. Saber de dónde recopilar esos casos o crear programas para construirlos. Establecer las condiciones en las que se deben aplicar los casos de prueba.

- Análisis estadístico de resultados. Una vez realizadas las pruebas, usar las técnicas estadísticas antes mencionadas para estudiar los datos: ajuste de regresión, representaciones gráficas, etc.

- Comparación con otros algoritmos. Establecer criterios de contraste y comparación con otros programas. Por ejemplo, en el caso del buscador web se podría dar a cada grupo el programa realizado por otro grupo. Al no tener disponible el código fuente habría que aplicar técnicas para el ajuste de funciones con una forma desconocida.

- Predicción del comportamiento para tamaños grandes. Además de para comparar, el ajuste realizado puede utilizarse para realizar previsiones del tiempo y la memoria en tamaños grandes y difíciles de alcanzar. Por ejemplo, se podrían extrapolar los tiempos del buscador a tamaños equiparables a un buscador como Google, con más de cuatro mil millones de páginas.

- Detección de fallos o problemas inesperados. El análisis experimental de algoritmos también se puede aplicar para detectar algún fallo en la implementación de los algoritmos, o alguna otra situación que hace que el tiempo aumente de forma inesperada para tamaños grandes.

La memoria de prácticas de esta parte debe contener los siguiente apartados:

- **Portada.** Nombre de los alumnos, y número de prácticas de cada uno.

- **Análisis y diseño de pruebas.** Establecimiento del método para la realización de las pruebas, que incluye:

  - Determinación de los objetivos del análisis. Decidir los parámetros a medir (uso de memoria y tiempo, especificar de qué parte). Determinar casos mejor, peor y promedio, en caso de no ser iguales. Identificar los valores que indican el tamaño del problema, e indicar los que se van a medir y cómo.
  - Análisis teórico de la eficiencia. Estudiar la eficiencia, de tiempo y memoria, de los algoritmos usados en el programa objeto de estudio. Obtener las funciones teóricas para cada caso.
  - Diseño de casos de prueba. Describir los casos de prueba que se deben aplicar, número de casos, tamaños a utilizar en cada caso y número de ejecuciones por caso. Indicar las condiciones para la ejecución de los casos (esto es, los factores externos).
  - Diseño del método de ejecución de las pruebas. Ejecución manual o automática. En el segundo caso, describir el programa diseñado para hacer las pruebas. Herramientas matemáticas o estadísticas usadas.

- **Listado del código.** En caso de haber creado algún programa para la ejecución automática o semiautomática de las pruebas, incluir aquí el listado.

- **Resultados y estudio estadístico.** Valores resultantes de la ejecución de los casos y análisis de los mismos. Debe contener los siguientes puntos:
  - Tablas de resultados obtenidos. Tiempos y memoria usada para cada caso de prueba, así como otra información relevante en caso necesario.
  - Representación gráfica de los resultados. Deberían haber gráficos representativos para cada una de las condiciones o parámetros de tamaño relevantes.
  - Ajuste estadístico. Ajuste de regresión de los resultados obtenidos a las funciones teóricas previstas. Estimación de la bondad del ajuste realizado (mediante el error de ajuste y tests de aleatoriedad).
  - Comparación de resultados. Comparar los resultados obtenidos con el programa propio y con el programa dado por el profesor. Comparar los distintos casos de ejecución identificados
  - Predicción. Predecir los tiempos de ejecución y memoria que se usarían en problemas de tamaño muy grande. Compararlos con otras soluciones disponibles, tipo Google.
- **Informe de desarrollo.** Igual que en la primera práctica, el informe incluye la organización y planificación de las distintas fases, la coordinación entre los miembros, las tablas de dedicación personal y las conclusiones personales de los alumnos.

Práctica 3. Resolución de problemas. El desarrollo de esta última práctica coincide con el último bloque de la asignatura, en el que se estudian diversas técnicas generales de diseño de algoritmos. Por lo tanto, el objetivo de la práctica es la aplicación de algunas de estas técnicas sobre problemas concretos. Para esta práctica proponemos el planteamiento de dos o tres problemas de tamaño pequeño/medio, o bien un mismo problema abordado desde diferentes técnicas. La solución no tiene por qué ser única, sino que pueden proponerse diferentes alternativas.

Existen básicamente dos alternativas en el planteamiento de estas prácticas: (1) plantear la resolución de problemas nuevos –esto es, no tratados en clase de teoría– dando el menor número posible de indicaciones sobre la forma de abordarlos o la técnica de diseño a aplicar; o, (2) proponer la implementación de algún algoritmo visto en clase, requiriendo el análisis exhaustivo de las distintas posibilidades que se ofrecen, y comparando cuál es la más adecuada. Ya hemos comentado que, para estas prácticas, creemos más adecuado la propuesta de varios problemas de tamaño pequeño que uno solo de tamaño grande. Los problemas pueden combinar problemas de los dos tipos, aunque consideramos que lo más adecuado aquí serían los del primer tipo.

Otra cuestión a considerar es si debe haber continuidad o no con las prácticas anteriores de la asignatura. Sería posible plantear las prácticas de la asignatura como un proyecto que va evolucionando a lo largo de todo el curso, usando en cada práctica los resultados obtenidos en las anteriores. Es difícil mantener esta continuidad, teniendo en cuenta que la segunda se centra en el análisis de la primera, en cierta forma cerrando el ciclo de análisis, diseño, y validación. La continuidad implicaría proponer problemas que hagan uso de los tipos de datos diseñados en la primera práctica –que, como hemos visto, pueden referirse al ámbito de los tipos conjunto, diccionario o grafo–. Pensamos que no es imprescindible planificar la tercera práctica como una continuación de las anteriores, y no supone ningún inconveniente que esta práctica parta desde cero.

El planteamiento de dos o tres problemas permite la posibilidad de aplicar más de una de las técnicas vistas en teoría. No obstante, los problemas deberían ser abordables en el tiempo disponible para las prácticas. Algunos ejemplos de problemas podrían ser los problemas NP-completos presentados en el tema de grafos, como el problema del viajante, de coloración de grafos o el del ciclo hamiltoniano, o bien problemas donde el modelado deba ser resuelto por los propios alumnos. En el segundo caso es

necesario identificar los datos del problema, encontrar la forma de describirlo con los tipos adecuados, buscar una técnica que sea aplicable y diseñar un algoritmo para el problema concreto. Un ejemplo concreto de este tipo de problemas podría ser el juego de las cifras: tenemos 6 números enteros entre 1 y 100, y debemos encontrar la forma de conseguir otro número entre 1 y 1000, usando las operaciones de suma, resta, multiplicación y división entera, y sin repetir un número más de una vez; si no se dan más indicaciones, los alumnos deberán decidir cómo es la representación de la solución y qué esquema algorítmico aplicar.

Es importante que los alumnos se centren en los aspectos algorítmicos del problema, dejando a un lado la interface de uso de los programas. El enunciado de las prácticas debe ser muy claro en este aspecto, estableciendo que los objetivos de la práctica son la aplicación de las técnicas de diseño y la obtención de buenos algoritmos y programas de calidad. También es conveniente que las prácticas no se queden en la simple implementación de los programas, sino que se incluya un apartado de verificación, pruebas y evaluación de la eficiencia obtenida, si bien no de forma tan exhaustiva como en la anterior práctica.

Está claro que intentar abordar todas las técnicas de diseño en las prácticas de la asignatura puede sobrepasar la carga establecida para la parte práctica. La posibilidad de incluir problemas de los últimos temas del programa –fundamentalmente ramificación y poda, y problemas de resolución de juegos– será menor, teniendo en cuenta que se dispone de menos tiempo entre su explicación teórica y su aplicación práctica. Pensamos que una manera viable de introducir estos temas es a través de problemas opcionales convalidables por algún ejercicio del examen. Estos problemas serían similares a los restantes de la tercera práctica, pero requiriendo la aplicación de técnicas avanzadas de diseño de algoritmos. La posibilidad de convalidar por alguna pregunta del examen, como ocurre en el seminario de especificaciones formales, hace que sea una posibilidad muy interesante para los alumnos.

La memoria a entregar por los alumnos es similar a la propuesta para la primera práctica, pero incluyendo la descripción separada de cada problema. El formato de la memoria sería:

- **Portada.** Nombre de los alumnos, y número de prácticas de cada uno.
- Para cada problema propuesto:
  - **Análisis.** Características del problema, datos disponibles, restricciones, tipo de problema y representación de la solución.
  - **Diseño.** Técnica aplicada, forma de aplicar la técnica, decisiones tomadas y algoritmos abstractos en pseudocódigo.
  - **Listado del código.** Listado para cada programa creado.
  - **Pruebas y eficiencia.** Pruebas realizadas para comprobar la validez del programa y breve estudio experimental de la eficiencia algorítmica obtenida.
- **Informe de desarrollo.** Incluye la organización y planificación de las distintas fases, la coordinación entre los miembros, las tablas de dedicación personal para cada problema y las conclusiones personales de los alumnos.

En esta memoria de prácticas, igual que en las anteriores, los alumnos deben tener la flexibilidad para ampliar la descripción en los aspectos que consideren más adecuados; y en cualquier caso manteniendo la racionalidad de la carga que la propia elaboración de la memoria requiere.

4.4.4. Coordinación entre teoría y prácticas

Ya hemos comentado la importancia de conseguir una estrecha coordinación entre teoría y prácticas en general, y en la asignatura objeto de estudio en particular. Ambas deben seguir un desarrollo paralelo, con el fin de alcanzar unos objetivos educativos comunes y complementarios.

En la asignatura AAED las prácticas han sido planteadas como clases de problemas, por lo que la coordinación entre ambas partes es directa. Cada semana se realizarán ejercicios relativos a los temas que han sido, o están siendo, tratados en teoría. La entrega de los boletines de ejercicios por parte de los alumnos, se realizará a la finalización de cada bloque del programa teórico.

Por su parte, las prácticas de AED implican un mayor número de actividades. La coordinación entre la teoría y la práctica para este caso es mostrada en la tabla 4.2.

Se puede observar que al principio del curso hay un inevitable desfase, debido a la necesidad de enseñar el lenguaje de programación. Esto da lugar a que el tiempo disponible para la primera práctica aparezca como un poco menor. Realmente, sería conveniente entregar el enunciado de la primera práctica antes de comenzar el seminario de C++; de esta forma los alumnos dispondrían de más tiempo para trabajar en esta práctica. En cualquier caso, todas las prácticas tienen lugar después de que los conceptos correspondientes hayan sido presentados en teoría.

En la planificación realizada, el seminario de especificaciones formales ha sido insertado entre los seminarios de C y C++. De esta forma se da algo de tiempo a los alumnos para madurar sus conocimientos de C, y se reduce la diferencia temporal entre la explicación teórica de los contenidos en el Tema 1, y su desarrollo práctico en este seminario. No obstante, también sería posible situar el seminario después del correspondiente a C++.

En la tabla 4.2 se ha hecho una estimación del transcurso de los meses, suponiendo una distribución anual de la asignatura. Hay que tener en cuenta que esta división es aproximativa, ya que el número de clases al mes siempre variará por uno u otro motivo. También es estimativa la fecha de entrega de las prácticas, que podría sufrir retrasos en caso necesario.

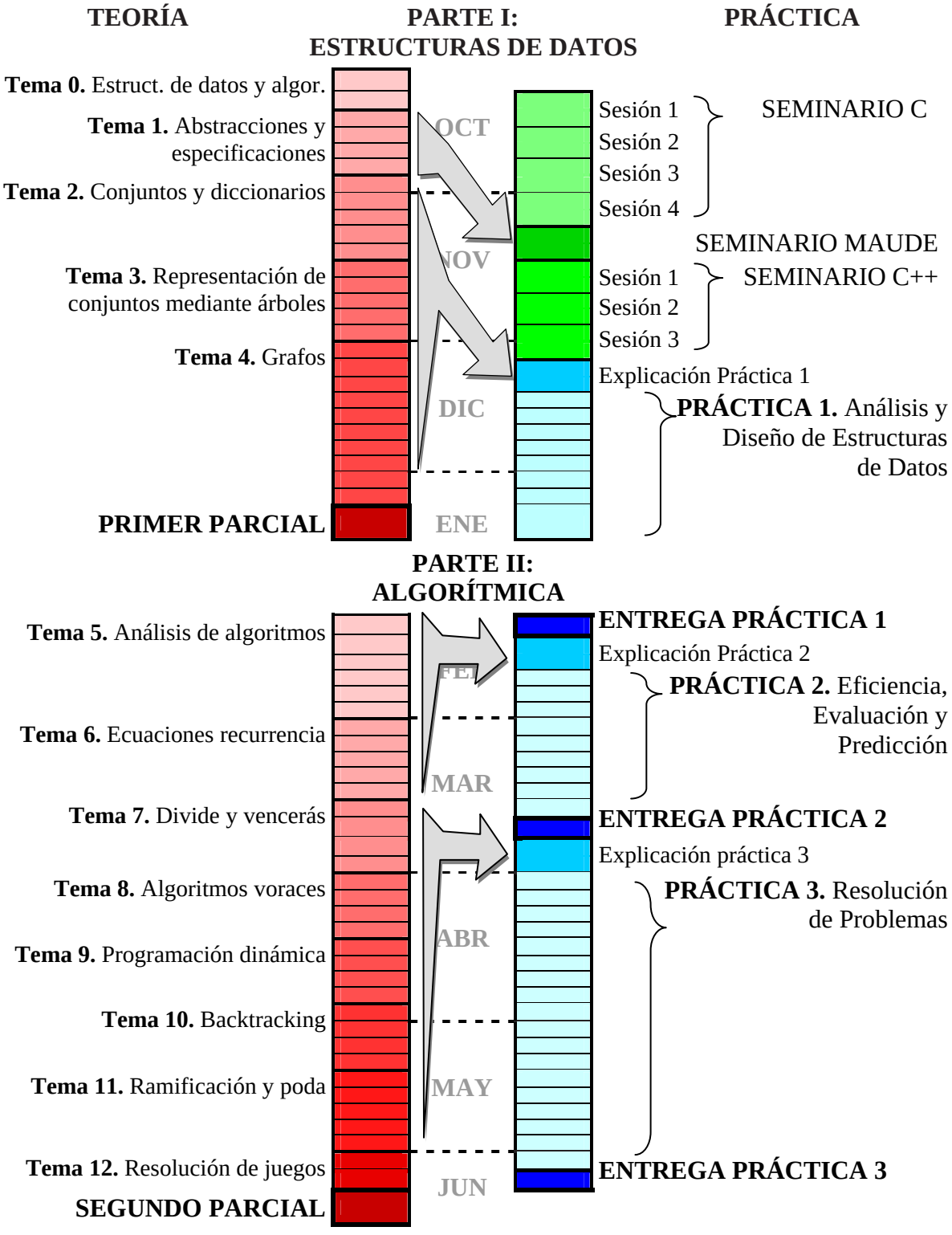


Figura 4.2: Coordinación entre teoría y práctica en la asignatura AED.

Capítulo 5

Programa de Teoría: Parte I - Estructuras de Datos

Cuéntamelo y olvidaré,
explícame y puede que no recuerde,
involúcrame y comprenderé.
Proverbio Nativo Americano

En este Capítulo y en el siguiente se describen los temas del programa teórico de forma muy detallada. La descripción de los temas llega a nivel de técnicas y aplicaciones recomendadas en cada una, dando una guía sobre el desarrollo de las explicaciones. Con esto no se pretende establecer un programa rígido e inflexible de la asignatura, sino más bien dar recomendaciones de utilidad en la preparación de las clases. Obviamente, el desarrollo de la acción docente debe dar respuesta a situaciones que no se pueden predecir a priori, como la pérdida de clases o el diferente grado de seguimiento de la explicación por parte de los estudiantes.

Para cada tema podemos encontrar los siguientes apartados:

- **Objetivos**, entendidos como los conocimientos, habilidades y actitudes que debe propiciar cada tema, y que deben ser una referencia en la evaluación de la asignatura. Algunos objetivos están enunciados desde el punto de vista del alumno ("conocer", "comprender", "ser capaz de") y otros desde el profesor ("presentar", "introducir", "mostrar"). Los primeros aparecen con el símbolo "•" y los segundos con "○".

- **Contenidos**, incluyendo los apartados y subapartados de cada tema.

- **Bibliografía básica**. Se incluye aquí la bibliografía mínima esencial recomendada para el tema, intentando reducir al máximo el número de referencias básicas. Normalmente encontramos el texto guía de la asignatura, y algún otro libro alternativo, que utiliza un enfoque parecido al planteado en el tema.

- **Bibliografía complementaria**. En este apartado se ofrece una mayor variedad de referencias relativas al tema. Se señalan los aspectos principales y más interesantes en los que se puede profundizar, y los sitios donde se puede consultar.

- **Descripción de los contenidos**, llegando a un alto nivel de detalle. Como ya hemos mencionado, esta descripción debe entenderse como una recomendación más que como una imposición de antemano sobre el modo de realizar la explicación.
- **Temporización** de los contenidos del tema. Se indica el número de horas recomendado para cada tema y apartado, incluyendo algunas indicaciones relativas a este respecto. También se han añadido tablas que reflejan el ritmo recomendado de desarrollo de los contenidos del tema. Las tablas ofrecen cierta flexibilidad, permitiendo un margen de actuación entre un ritmo más rápido o más lento. En algunos temas aparece en su tabla una fila "**Prob.**" que indica la posibilidad de realizar problemas en las horas de clase, al final del tema. Estas tablas han sido creadas utilizando la experiencia docente del candidato en la asignatura, por lo que consideramos que son bastante realistas y aplicables.

Tema 0 - Estructuras de datos y algoritmos

Objetivos

- Presentar una visión global de la asignatura, motivando los contenidos que serán desarrollados a lo largo del curso.
- Repasar los conceptos fundamentales relacionados con tipos abstractos de datos, tipos de datos, estructuras de datos y algoritmos, vistos en primer curso.

- Concienciarse de la importancia del análisis y diseño en la resolución de problemas de programación, como pasos previos a la implementación.
- Conocer la necesidad de manejar distintos niveles de abstracción en la resolución de problemas, desde la visión global más abstracta hasta el nivel de programación.

Contenidos

1. Tipos de datos, tipos abstractos y estructuras de datos.
   1.a) El concepto de tipo abstracto de datos (TAD).
   1.b) Tipos de datos y estructuras de datos.
   1.c) Relación entre TAD, tipo de datos y estructura de datos.
2. Tipos de tipos.
   2.a) Tipos escalares, compuestos, primitivos, contenedores.
   2.b) Tipos mutables e inmutables.
   2.c) Algunos tipos contenedores: tipos lineales, árboles, conjuntos y grafos.
3. Algoritmos que resuelven problemas.
   3.a) Definición y propiedades de algoritmo.
   3.b) Relación entre las estructuras de datos y los algoritmos.
   3.c) Procesos en la resolución de problemas.
4. Algorítmica: análisis y diseño.

4.a) Objetivos del análisis de algoritmos. Notaciones de complejidad.
4.b) Algoritmos y esquemas de algoritmos. Clases de problemas.
4.c) Principales esquemas algorítmicos.

5. Consejos para una buena programación.
5.a) Importancia del análisis y diseño.
5.b) Modularidad, encapsulación y ocultamiento de la implementación.
5.c) Reutilización y genericidad.
5.d) Repartir la funcionalidad.

Bibliografía básica

Capítulo 1 de [García'03] y capítulo 1 de [Aho'88].

Bibliografía complementaria

Este primer tema supone un repaso de los principales conceptos que se supone que el alumno conoce de primer curso. Para una revisión más profunda la mejor referencia es, sin duda, la que se usara en las asignaturas correspondientes. Alternativamente, se pueden usar algunos libros propios de AED que incluyen capítulos de repaso de los tipos fundamentales –listas, pilas, colas y árboles– y las principales definiciones relacionadas con estructuras de datos y algoritmos, como los capítulos 2 y 3 de [Aho'88], los capítulos 3 y 4 de [Weiss'95], el capítulo 2 de [Baase'00], el libro [Wirth'80] o el capítulo 10 de [Cormen'90], entre muchos otros.

Descripción de los contenidos

El primer tema de teoría consiste básicamente en un breve repaso de los conocimientos estudiados en la asignatura de programación de primer curso. El alumno debe concienciarse de la importancia de conceptos tales como los tipos abstractos de datos (TAD), los tipos de datos, estructuras de datos y análisis y diseño de algoritmos, que constituyen una base imprescindible para la asignatura de AED.

Por otro lado, hay que recordar la conveniencia de motivar siempre en los estudiantes el interés hacia los temas a desarrollar posteriormente. En este sentido, este tema debe tratar de motivar la asignatura en su conjunto, haciendo hincapié en su carácter básico dentro de una titulación de informática, a la vez que presentando aplicaciones actuales y llamativas que requieren un cuidadoso estudio de los algoritmos y las estructuras de datos usados.

En el punto 1 se empieza repasando los conceptos de TAD, tipo de datos y estructura de datos. Requerir su definición por parte de los alumnos es una buena manera de contrastar su nivel inicial de conocimientos. Aunque la mayoría de ellos conocen el concepto de TAD[1], es importante recalcar la idea de abstracción, por la cual al usuario de un TAD le es indiferente cómo se almacenan los datos o cómo se implementan las operaciones del mismo.

Después de definir los tipos y las estructuras de datos, se analizan las relaciones entre los tres conceptos viendo, a través de algún ejemplo, el tipo de datos como una implementación del TAD y la estructura de datos como la representación en memoria de un valor del tipo. Se repasa la doble perspectiva de un TAD: como usuario del tipo y como programador del mismo.

Algunos conceptos y propiedades sobre tipos de datos son explicados en el punto 2. Se repasan las definiciones de tipo primitivo, enumerado, compuesto, mutable, inmutable, etc. Entre los tipos compuestos se distinguen los llamados contenedores o colecciones, teniendo en cuenta que prácticamente todos los tipos estudiados en la asignatura serán contenedores. Sin entrar en mucho detalle, se

[1] En un test realizado a comienzo del curso –disponible en el apéndice B– el 68 % de los alumnos identificó correctamente el concepto de TAD, mientras que el 14 % respondió que un TAD es la implementación de una estructura de datos.

revisan los tipos estudiados en primer curso (listas, pilas, colas y árboles) y se presentan los principales tipos que serán estudiados en la asignatura (conjuntos, diccionarios y grafos).

El punto 3 trata de los algoritmos, empezando por la definición y propiedades de definibilidad y finitud de un algoritmo. Se muestran ejemplos de cosas que no son algoritmos, aclarando que no todos los problemas de programación son necesariamente de naturaleza algorítmica; el ejemplo típico expuesto es un sistema operativo, cuya ejecución puede no ser finita. Después se analizan las relaciones entre algoritmos y estructuras de datos: una estructura de datos no trivial requiere diseñar algoritmos eficientes para su manejo, y un algoritmo suele requerir el uso de una o más estructuras de datos adaptadas para el problema específico. Se intenta conseguir que los alumnos no disocien ambos conceptos y, por lo tanto, las dos partes de la materia, aunque en la práctica en la enseñanza de la programación suelen aparecer de forma separada[2].

A continuación, se presenta la idea de usar un proceso metodológico e ingenieril en la resolución de problemas de programación, frente a la programación exclusivamente como un arte. Se muestran, por ejemplo, el proceso de refinamientos sucesivos, el proceso propuesto en [Aho'88] basado en abstracciones, y el proceso clásico compuesto por análisis, diseño, implementación y verificación. Se hace referencia al área de la *ingeniería del software*, en cuyas asignaturas tendrán oportunidad de profundizar en los procesos de análisis y diseño.

En el punto 4 se presenta la algorítmica como una disciplina científica compuesta por las áreas de análisis y diseño de algoritmos. Se comentan los objetivos de cada una de ellas, mostrando los conceptos de notación de complejidad y esquema algorítmico. Se hace una clasificación de tipos de problemas (de optimización discreta o continua, de satisfacción de restricciones buscando una solución o todas, de juegos, etc.), aclarando que ciertas técnicas pueden ser más o menos adecuadas para cierta categoría de problemas. Se hace también una enumeración de los esquemas algorítmicos que serán estudiados en la segunda parte de la asignatura. No obstante, este punto debe tratarse muy brevemente ya que convendría hacer una presentación más detenida al empezar la segunda parte de la asignatura.

Finalmente, el punto 5 contiene unos cuantos consejos generales para una buena programación; en su mayoría son ideas que volverán a aparecer a lo largo de los temas restantes. Cuestiones como hacer análisis y diseño antes de empezar a programar, repartir la funcionalidad en módulos de manera adecuada, respetar el ocultamiento de la implementación y reutilizar código, deberían ser tenidas en cuenta por los alumnos al programar. Debido a su carácter práctico, sin embargo, insistir demasiado en estos principios en clase de teoría suele ser bastante poco productivo. Pensamos que resulta más adecuado retomar estas ideas en las prácticas de la asignatura, por ejemplo verificando el nivel de cumplimiento de cada uno de los puntos en la realización de las mismas.

Temporización

Tiempo estimado de clases teóricas: **2 horas**. La planificación es mostrada en la tabla 5.1.

La primera hora de clase incluye la presentación de la asignatura, aunque no aparece en la lista de contenidos ya que no forma parte del temario. En esta presentación se da una visión global de la asignatura, mostrando su relación con otras asignaturas de la carrera y se comenta cómo será el desarrollo de las prácticas y los criterios de evaluación. Aunque los contenidos pueden ser explicados en menos tiempo, la propia experiencia –como profesor y como alumno– aconseja empezar el curso con una presentación distendida de la asignatura, de manera que no se produzca un comienzo traumático tras el final de las vacaciones.

[2]Lo cual no es necesariamente contradictorio: el alumno debe aprender a relacionar conocimientos que aparezcan en temas distintos o, incluso, en asignaturas diferentes.

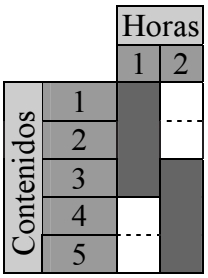


Tabla 5.1: Planificación temporal del Tema 0.

Tema 1 - Abstracciones y especificaciones

Objetivos

- Reconocer la importancia de la abstracción –no sólo en programación, sino como un mecanismo fundamental de la capacidad intelectual humana– y conocer los mecanismos y tipos de abstracciones que aparecen en programación.
- Concienciarse de la necesidad de desarrollar especificaciones del software que sean útiles y precisas, entendiendo la especificación como un punto de acuerdo entre el usuario y el implementador de una abstracción.
- Conocer algún lenguaje de especificación formal de TAD y aplicarlo en la especificación de los TAD estudiados en la asignatura. En particular, conocer el método algebraico, basado en el uso de axiomas, y el constructivo, basado en la definición de pre- y postcondiciones.

◦ Introducir los conceptos de clase y objeto, así como otros mecanismos de los lenguajes de programación que soportan el uso de abstracciones.

Contenidos

1. Las abstracciones en programación.
   - 1.a) Mecanismos de abstracción: especificación y parametrización.
   - 1.b) Tipos de abstracciones: funcionales, de datos y de iteradores.
   - 1.c) Mecanismos de los lenguajes de programación para la abstracción de datos.
   - 1.d) Soporte de tipos parametrizados en los lenguajes de programación.
2. Especificaciones informales.
   - 2.a) De abstracciones funcionales.
   - 2.b) De abstracciones de datos.
   - 2.c) De abstracciones de iteradores.
3. Especificaciones formales algebraicas o axiomáticas.
   - 3.a) Propiedades, notación y ventajas de las especificaciones formales.
   - 3.b) Significado y uso de los axiomas.
   - 3.c) Completitud y corrección de la especificación.
   - 3.d) Reducción de expresiones algebraicas.

4. Especificaciones formales constructivas u operacionales.
   4.a) Precondiciones y postcondiciones.
   4.b) La necesidad del modelo subyacente.
   4.c) Especificación como contrato de una operación.

Bibliografía básica

Capítulo 2 de [García'03] y capítulo 2 de [Collado'87].

Bibliografía complementaria

Aunque los conceptos de abstracción y especificación aparecen en la mayoría de libros de AED (como, por ejemplo, en el capítulo 1 de [Aho'88]), en pocos se estudian en detalle los mecanismos de abstracción y de especificación. Para profundizar en las especificaciones algebraicas se puede utilizar como material complementario [Franch'94]. Este libro sigue una aproximación bastante formal al estudio de los TAD, planteando la especificación algebraica de todos los tipos tratados. Otra referencia interesante es [Peña'98], que introduce las especificaciones algebraicas en el capítulo 5, y las aplica a varios TAD (la mayoría de los incluidos en la asignatura) en el capítulo 6. También en el tema 1 de [Campos'95] se desarrolla la técnica axiomática con cierta extensión. En cuanto a las especificaciones constructivas, la idea de las pre- y postcondiciones como contratos es debida a B. Meyer, y es desarrollada en [Meyer'99] en los capítulos 11 y 12. En este libro se encuentran las bases y los principios de la programación orientada a objetos, aunque en general resulta algo excesivo para el nivel que se requiere en la asignatura.

Descripción de los contenidos

Como punto de partida hacia los temas posteriores, donde se estudian estructuras de datos y algoritmos concretos, en este tema se trata de motivar la necesidad de adquirir una visión global y abstracta de los problemas, particularizando en cada momento los aspectos que sean de interés. El tema está centrado en el estudio de las especificaciones que deberían acompañar cualquier producto software, ya sea un tipo de datos, un programa o un simple procedimiento. Se pretende, por lo tanto, que las diferentes técnicas de especificación expuestas sean aplicadas por los alumnos sobre los tipos de datos y los algoritmos que se verán con posterioridad.

En el punto 1 se empieza estudiando la abstracción como el proceso de eliminar lo irrelevante para quedarse con lo esencial, lo realmente importante. A través de la pregunta retórica "¿qué es lo importante?", se discute cómo la abstracción es una capacidad intelectual humana, muy difícil de emular en una máquina. Se ven algunas ideas estudiadas en primer curso, como la abstracción por especificación y por parametrización, que da lugar a la idea de genericidad. Se repasan los tres tipos fundamentales de abstracciones que el alumno también debe conocer: funcionales, de datos y de iteradores.

Los subapartados 1c) y 1d) relacionan estos tipos de abstracciones con los mecanismos de los lenguajes de programación que les dan soporte. Gran parte de ellos ya son conocidos por los estudiantes; la novedad se encuentra en la definición del concepto de clase como una manera de crear tipos de datos abstractos. Guardando la coherencia con la parte de prácticas, se muestra la definición de una clase en C++, con atributos y métodos públicos y privados. Se presenta el concepto de objeto como valor en el dominio del tipo, y su manipulación a través de la notación punto. Después se ve cómo es posible definir tipos parametrizados en C++, mediante las estructuras *template*. Se excluye intencionadamente

todo lo relativo a la herencia[3].

En el punto 2 se presenta una notación para la especificación informal de abstracciones. La abstracción operacional es especificada mediante una descripción textual, con la cabecera de la operación y las cláusulas *requiere*, *modifica* y *calcula*. Por otro lado, la especificación informal de la abstracción de datos contiene el nombre del tipo abstracto, una descripción textual global del tipo y la lista de operaciones del mismo, cada una de ellas especificada con el formato de especificación informal de abstracciones funcionales.

La anterior notación debe resultar familiar para el alumno, ya que una notación similar es usada en primer curso. A pesar de esta aparente duplicidad de contenidos, creemos conveniente volver a incidir sobre las especificaciones informales, ya que en la práctica las especificaciones formales son poco utilizadas. De esta manera, la especificación informal es aplicable como un método de documentación del software, que el alumno podrá usar, por ejemplo, durante las prácticas de la asignatura.

Antes de introducir la especificación informal de iteradores, se justifica con un ejemplo la necesidad de definir recorridos abstractos sobre los tipos de datos contenedores, permitiendo operaciones del tipo: **PARA** cada elemento **HACER** acción. La especificación informal se hace a través de la definición de un tipo abstracto de datos que indica el estado de la iteración.

El núcleo del tema se encuentra en los puntos 3 y 4, donde se tratan las especificaciones formales. Se motiva su estudio como una forma de eliminar la ambigüedad implícita en las especificaciones informales, y se ven sus principales ventajas: verificación de programas, prototipado a partir de las especificaciones y la posibilidad de reutilizarlas en distintos ámbitos.

En el subapartado 3a) se presenta el formato de las dos notaciones formales que se estudiarán: la algebraica (o axiomática) y la constructiva (u operacional). En ambos casos la definición consta de cuatro apartados, encontrándose la diferencia entre ambas notaciones en el último de ellos:

- **Nombre**. Nombre genérico (y parametrizado, en su caso) del tipo abstracto.
- **Conjuntos**. Conjuntos de datos que intervienen en la definición.
- **Sintaxis**. Signatura de las operaciones, usando la notación funcional matemática.
- **Semántica**. Establece el significado de las operaciones.

Mientras que en la especificación algebraica el significado se establece en base a expresiones que relacionan distintas operaciones entre sí (los axiomas), en la constructiva el significado viene dado por la definición de la precondición y la postcondición de cada operación. De esta manera, se distingue entre una semántica de las operaciones implícita o explícita, respectivamente. El resto del punto 3 está centrado en las especificaciones axiomáticas.

Una de las grandes dificultades, desde el punto de vista del alumno, relacionadas con el uso de especificaciones algebraicas es cómo garantizar que el conjunto de axiomas de la especificación es completo y correcto, es decir que no sobran o faltan axiomas. Una manera de abordar este problema es hacer una clasificación de las operaciones del TAD en operadores de: construcción, modificación y consulta. Entonces, se muestra cómo para garantizar la completitud de la especificación hay que relacionar de forma adecuada todas las operaciones de modificación y consulta con cada uno de los constructores.

Después de plantear algunas especificaciones algebraicas, se estudia el proceso de reducción de expresiones como forma de animar la especificación. Los ejemplos típicos incluyen tipos básicos –

[3]Es importante aclarar que en este punto sólo se hará una introducción breve a los conceptos orientados a objetos mencionados. No se pretende una profundización teórica en orientación a objetos, que será estudiada extensamente en tercer curso. Los alumnos se familiarizarán en la parte práctica con la definición y uso de clases y objetos en C++.

como el tipo entero– y tipos contenedores –como listas, pilas o colas– haciendo uso de especificaciones parametrizadas.

En el punto 4 se estudian las especificaciones formales constructivas, donde cada operación es definida de manera separada mediante una precondición y una postcondición. Estos conceptos ya son conocidos por los alumnos, por lo que el interés aquí es que se use una notación matemática precisa para definir ambos predicados, lo cual requerirá normalmente un modelo subyacente apropiado.

Con el estudio de un ejemplo sencillo se muestra cómo cierto requisito de una operación puede ser incluido en la precondición o bien ser tratado como caso de error en la postcondición. Entonces se plantea "¿cuál es la mejor opción?" Para el alumno ambas opciones parecen igualmente válidas. Viendo superficialmente la idea de la programación por contrato, [Meyer'99], se da la visión de la especificación como un contrato entre el usuario y el programador de la operación. En consecuencia, lo más adecuado es incluir el requisito en la precondición y no hacer comprobación de errores en la postcondición.

Por último, se desarrollan varias especificaciones constructivas de tipos contenedores siguiendo estas ideas y se muestra la ejecución de algunas operaciones de los tipos especificados.

Temporización

Tiempo estimado de clases teóricas: **4 horas**. La planificación es mostrada en la tabla 5.2.

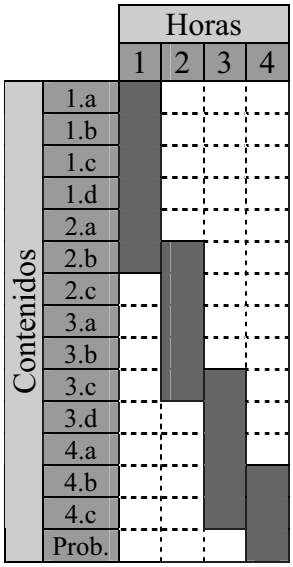


Tabla 5.2: Planificación temporal del Tema 1, Parte I.

Los dos primeros puntos pueden considerarse como un repaso de conceptos vistos, en su mayoría, en primer curso. Claramente, el núcleo del tema son los puntos 3 y 4, donde se estudian las especificaciones formales algebraicas y constructivas, respectivamente. Cada una de ellas puede requerir algo más de una hora de clase, incluyendo el tiempo necesario para realizar algunos ejercicios de ejemplo. También hay tiempo para realizar algunos ejercicios al final del tema.

Tema 2 - Conjuntos y diccionarios

Objetivos

- Familiarizarse con la notación y terminología de los tipos de datos abstractos conjunto, diccionario y bolsa, siendo consciente de la importancia y ubicuidad de estos tipos en el desarrollo de programas.
- Ser capaz de diseñar, implementar y analizar estructuras de representación no arbóreas para los tipos conjunto y diccionario, adaptadas a las necesidades específicas de cada aplicación.
- Conocer la estructura de datos de tablas de dispersión, sus distintas variantes y los factores que influyen en su eficiencia.
- Concienciarse de la importancia del factor eficiencia, en cuanto a tiempo de ejecución y uso de memoria, en el diseño e implementación de estructuras de datos.
- Comprender la independencia entre la idea de datos almacenados y estructura de acceso a esos datos, a través del estudio de estructuras duales o múltiples.

Contenidos

1. Notación e implementaciones básicas de conjuntos.
   - 1.a) El concepto matemático y los conceptos informáticos.
   - 1.b) Definición del TAD conjunto.
   - 1.c) Representación mediante vectores de bits.
   - 1.d) Representación mediante listas enlazadas, ordenadas y no ordenadas.
2. El TAD diccionario.
   - 2.a) Definición de los tipos asociación y diccionario.
   - 2.b) Similitud y diferencias con el TAD conjunto.
   - 2.c) Representación sencilla mediante una tabla.
3. Tablas de dispersión.
   - 3.a) La dispersión y los sinónimos.
   - 3.b) Dispersión abierta.
   - 3.c) Dispersión cerrada. La redispersión.
   - 3.d) Funciones de dispersión y redispersión.
   - 3.e) Eficiencia de las tablas de dispersión.
   - 3.f) El algoritmo de Brent.
4. Asociaciones muchos a muchos y estructuras duales.
   - 4.a) La relación muchos a muchos.
   - 4.b) Representaciones inmediatas con matrices y listas.
   - 4.c) Representación mediante listas múltiples.
   - 4.d) Combinación de estructuras de datos.
   - 4.e) Ejemplo de combinación de tabla de dispersión y tabla ordenada.

Bibliografía básica

Capítulo 3 de [García'03] y capítulo 4 de [Aho'88].

Bibliografía complementaria

Las tablas de dispersión son el aspecto más interesante en el que profundizar de este tema. Se trata de técnicas de representación ampliamente utilizadas y, por lo tanto, son estudiadas en la mayoría de los libros de estructuras de datos. El nivel de detalle con el que se tratan en la asignatura es similar al de [Aho'88] en el capítulo 4, o al de [Weiss'95] en el capítulo 5. En esta última referencia se puede encontrar otra técnica de redispersión, conocida como *dispersión extensible*, que se suele usar en aplicaciones de bases de datos. Otro punto interesante en el que profundizar es el estudio de las conocidas como funciones de *dispersión perfectas*, que se pueden consultar en el capítulo 11 de [Cormen'90], o en el capítulo 10 de [Drozdek'01].

Por otro lado, las ideas de las estructuras de datos duales y las listas múltiples están sacadas de [Aho'88] y son analizadas en el apartado 4.12. La misma idea, bajo el nombre de listas ortogonales o matrices *escasas*, se puede encontrar en el apartado 2.2.6 de [Knuth'97], el apartado 3.6 de [Drozdek'01] y la lección 25 de [Campos'95]. Ideas similares, como la combinación de estructuras individuales, se pueden consultar en el capítulo 14 de [Cormen'90].

Descripción de los contenidos

Este tema trata la definición de los tipos abstractos conjunto y diccionario, y su representación utilizando estructuras de datos no arbóreas. En particular, el núcleo de interés del tema es el estudio de las tablas de dispersión, vistas como una forma eficiente de implementar el tipo abstracto conjunto (o diccionario). De hecho, esta visión de la estructura de datos como una implementación particular del tipo abstracto es el nexo de unión entre las diferentes técnicas de representación estudiadas en este tema y en el siguiente. Así, el alumno debe conocer que el objetivo final de todas ellas es siempre el mismo: ofrecer un conjunto de operaciones (*insertar*, *eliminar*, *miembro*, etc.) cuyo funcionamiento sea coherente con la semántica del tipo abstracto.

El punto 1 empieza con un breve repaso de teoría de conjuntos, que los alumnos ya deben conocer de las asignaturas de matemáticas. Se definen los conjuntos como colecciones no ordenadas de elementos distintos, viendo cómo en programación interesa normalmente que los elementos sean del mismo tipo; las operaciones del tipo abstracto conjunto son enumeradas, repasando la notación y significado. Se presenta también el tipo bolsa.

Después se ven los dos métodos más elementales para representar conjuntos: mediante vectores de valores booleanos y mediante listas de elementos. Aunque su programación para un alumno de segundo curso no debería presentar ninguna dificultad, el interés de presentar estas dos estructuras es mostrar dos ideas clave –y en cierto sentido contrapuestas– en cuanto a representación de datos: indexar cada elemento del conjunto universal en una posición fija de una tabla, o almacenar exclusivamente los elementos de interés en una estructura enlazada. Esto da pie a una discusión de ventajas e inconvenientes entre estas representaciones, y su implementación usando estructuras estáticas y dinámicas; también se discuten los beneficios de usar listas ordenadas o sin ordenar.

Antes de estudiar implementaciones más complejas de conjuntos, se introduce en el punto 2 el tipo abstracto diccionario. Se define el concepto de asociación, como un TAD formado por el par $(clave, valor)$, y el TAD diccionario como un conjunto de elementos de tipo asociación, donde la inserción requiere ambos elementos del par, pero las operaciones de consulta y eliminación se hacen por clave. Para tantear el nivel de entendimiento de los estudiantes puede ser interesante plantear la cuestión: ¿tiene sentido definir "diccionario" como un TAD distinto o es equivalente al TAD conjunto? La pregunta es respondida atendiendo a la definición de TAD: siempre que cambie la sintaxis o la semántica de las operaciones tenemos un TAD distinto. No obstante, se considera la similitud entre ambos tipos, de manera que para todas las estructuras vistas en adelante (en este tema y en el siguiente)

el alumno debe tener en mente su aplicación para representar tanto conjuntos como diccionarios.

El punto 2 acaba con el estudio de una representación muy sencilla de diccionarios, a través de una tabla de registros de tipo asociación. Los elementos no están ordenados, por lo que los nuevos valores son añadidos simplemente al final de la tabla y una consulta requiere recorrer toda la tabla en caso de no pertenencia.

A continuación se estudia la técnica de dispersión, o *hashing*[4], en el punto 3. Se destaca la gran eficiencia de la representación con vectores de bits para el acceso a cada elemento particular, pero que tiene como contrapartida su limitación en el tamaño del conjunto universal. Entonces, surge de manera natural la idea de la función de dispersión h, como una función que dado un elemento devuelve una posición en una tabla de tamaño definido.

En los diccionarios la función de dispersión tiene la forma: $h : Claves \rightarrow [0, ..., B-1]$, donde $Claves$ es el dominio de la clave, o de una parte de la clave, y B es el tamaño de la tabla de dispersión. Se plantea el problema de los sinónimos –dos elementos x e y distintos con $h(x) = h(y)$– para el cual se estudiarán dos formas de tratarlos: mediante dispersión abierta y dispersión cerrada.

En primer lugar se trata la dispersión abierta o externa, cuya comprensión suele resultar más sencilla para los alumnos. Se definen los conceptos de cubeta y clase, de manera que cada posición de la tabla es una lista de elementos (una cubeta) donde se meten todos los que tienen ese valor de dispersión (los de la misma clase).

En segundo lugar se estudia la dispersión cerrada o interna, en el subapartado 3c). Aparece el problema de las colisiones y se muestra cómo es resuelto mediante la redispersión. En concreto, se presenta la variante más sencilla: la redispersión lineal. Se plantean las operaciones de inserción y la eliminación usando marcas de eliminado. Se muestra el problema de agrupamiento provocado por la redispersión lineal, lo cual justifica el uso de mejores funciones.

El subapartado 3d) contiene este estudio teórico de cómo deben ser unas buenas funciones de dispersión y de redispersión. Primero son planteadas las propiedades que deben tener unas funciones para ser consideradas buenas, y luego se estudian algunas de las más comunes. Los alumnos ven con algunos ejemplos cómo la idea de función de dispersión puede aplicarse sobre enteros, reales, cadenas o cualquier otro tipo.

Una vez vistas la dispersión abierta y la cerrada, se hace un estudio teórico de la eficiencia de las operaciones de inserción y búsqueda, y de la ocupación de memoria en ambas estructuras; se discuten los casos donde puede ser mejor utilizar una u otra. Al estudiar el tiempo, se comprueba la degradación de la eficiencia a medida que se llena la tabla, planteando la posibilidad de reestructurar la tabla, es decir, disponer otra con mayor tamaño. Para acabar el punto 3, se explica el algoritmo de Brent para mejorar la eficiencia de las búsquedas en dispersión cerrada cuando las inserciones y eliminaciones son poco frecuentes. La idea del algoritmo es mostrada a través de un ejemplo práctico, donde la inserción de un elemento produce colisiones.

Finalmente, en el punto 4 se presentan dos ejemplos de aplicaciones que usan el tipo conjunto o diccionario. Aunque no resultan trascendentales en sí mismos, ambos tienen el interés de mostrar la posibilidad de combinar distintas estructuras de datos en una sola estructura, con el fin de mejorar la eficiencia de ciertas operaciones. De esta manera, se pretende independizar los conceptos de datos almacenados y estructura de acceso a los mismos, que hasta el momento los alumnos asocian de manera unívoca.

Primero se consideran las relaciones muchos a muchos, con el ejemplo "un alumno está matriculado en muchos cursos y un curso tiene matrículas de muchos alumnos". Se estudia la estructura de listas múltiples (en un sentido enlazando las matrículas de un alumno y en el otro enlazando las

[4]En general, consideramos preferible utilizar la terminología en castellano, siempre que exista y esté ampliamente aceptada. No obstante, también es importante que los alumnos conozcan la denominación en inglés.

matrículas de un curso) como una forma eficiente de representar una matriz escasa, es decir con la mayoría de sus valores nulos.

El segundo ejemplo hace referencia a la combinación de dos estructuras de diferente naturaleza para mejorar la eficiencia de los accesos. Por ejemplo, si representamos empleados de una empresa y queremos acceso rápido por nombre y por categoría, podemos combinar una estructura de tabla de dispersión indexada por nombres y una lista ordenada por categoría. Se discuten las ventajas e inconvenientes que surgen de manera genérica usando este tipo de estructuras duales.

Temporización

Tiempo estimado de clases teóricas: 5 horas. La planificación es mostrada en la tabla 5.3.

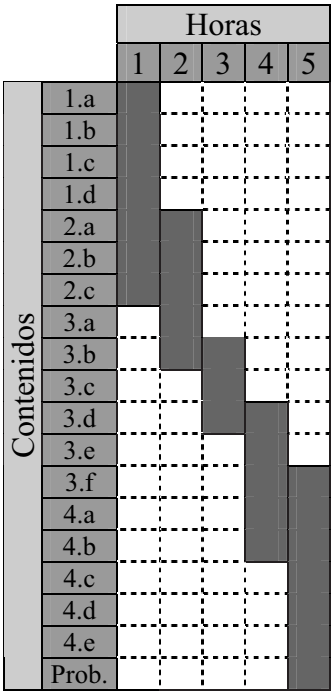


Tabla 5.3: Planificación temporal del Tema 2, Parte I.

El núcleo de interés del tema está claramente en el punto 3, donde se estudian las tablas de dispersión. Su explicación puede requerir en torno a 3 horas, incluyendo la realización de algunos ejercicios de ejemplo. Los restantes puntos podrían reducirse, en caso necesario, para ajustarse al tiempo total estimado para este tema.

Tema 3 - Representación de conjuntos mediante árboles

Objetivos

- Entender y conocer un conjunto de técnicas eficientes de representación de conjuntos y diccionarios mediante estructuras arbóreas, como los árboles trie, AVL y B.

- Comprender la necesidad de usar mecanismos de equilibrado o balanceo para conseguir eficiencia en las distintas representaciones arbóreas.
- Adquirir la habilidad de evaluar las necesidades de representación de una aplicación específica, tomando decisiones justificadas sobre las estructuras de representación más adecuadas.
- Ser capaz de diseñar e implementar tipos de datos usando las estructuras estudiadas en este tema, realizando las adaptaciones que cada aplicación concreta requiera.

Contenidos

1. Árboles trie.
   - 1.a) Los trie como árboles de prefijos.
   - 1.b) Representación de nodos trie mediante tablas.
   - 1.c) Representación hijo izquierdo, hermano derecho.
   - 1.d) Eficiencia y comparación entre estructuras.
2. Relaciones binarias de equivalencia.
   - 2.a) Relaciones y clases de equivalencia.
   - 2.b) Representaciones sencillas de relaciones de equivalencia.
   - 2.c) Representación mediante punteros al padre.
   - 2.d) Equilibrado y compresión de caminos.
3. Árboles de búsqueda balanceados.
   - 3.a) Árboles binarios de búsqueda, perfectamente balanceados y AVL.
   - 3.b) El peor caso de árbol AVL.
   - 3.c) Rotaciones simples y dobles sobre árboles AVL.
   - 3.d) Inserción en un árbol AVL.
   - 3.e) Eliminación en un árbol AVL.
4. Árboles B.
   - 4.a) Definición de árbol B de orden p.
   - 4.b) Inserción en un árbol B.
   - 4.c) Eliminación en un árbol B.
   - 4.d) Análisis de eficiencia de los árboles B.

Bibliografía básica

Capítulo 4 de [García'03] y capítulos 5 y 11 de [Aho'88].

Bibliografía complementaria

Para profundizar en la técnica de los árboles trie se puede consultar el capítulo 7 de [Drozdek'01], donde se desarrolla una implementación en Java, incluyendo un completo estudio de su aplicación a un corrector ortográfico interactivo. El problema de representar relaciones de equivalencia se puede encontrar en varias referencias aunque con distintos nombres; por ejemplo, en [Weiss'95] en el capítulo 8, en [Cormen'01] en el capítulo 21 y en [Baase'00] en el apartado 6.6. En cuanto a los árboles binarios de búsqueda balanceados y árboles AVL, las referencias más interesantes son [Weiss'95] en el apartado 4.4, [Wirth'80] en el capítulo 4, el apartado 6.3 de [Levitin'03], y el capítulo 5 de [Hernández'01]. Los alumnos interesados en este tipo de problemas pueden profundizar estudiando los árboles denominados *rojos-negros*, por ejemplo en el capítulo 13 de [Cormen'01]. Los árboles B son analizados en muchas referencias de bases de datos, además de las propias de AED. Podemos destacar [Hernández'01] aparta-

do 6.2, [Wirth'80] apartado 4.5, [Cormen'01] capítulo 18, y el capítulo 7 de [Drozdek'01], donde se analizan los árboles B y las variantes B+ y B*.

Descripción de los contenidos

Siguiendo con la implementación de conjuntos y diccionarios, este tema presenta una variedad de técnicas de representación que tienen todas ellas la particularidad de usar estructuras arbóreas. Desde esta perspectiva, tanto los árboles trie, como los AVL y los árboles B son considerados, en última instancia, como implementaciones de una misma abstracción que incluye operaciones de inserción, consulta y eliminación.

Por otro lado, además del interés de las estructuras en sí mismas, todas las técnicas expuestas pueden aportar ideas útiles en cuanto a la organización de los datos y la consecución de eficiencia en las operaciones del tipo. Así, por ejemplo, conociendo la idea del trie como árbol de prefijos, el alumno se puede plantear la posibilidad de definir árboles de sufijos; o en el caso de los árboles AVL, podrían proponerse otras condiciones similares de balanceo alternativas y estudiar el efecto.

En general, las cuatro técnicas de representación estudiadas en este tema son bastante independientes entre sí, por lo que el orden de presentación no es un factor relevante; en cierto sentido el orden propuesto intenta seguir un nivel de dificultad creciente. Se supone, además, cierta familiaridad de los alumnos con el tipo árbol y con la programación de estructuras arbóreas, estudiadas extensamente en primer curso.

La primera técnica tratada en este tema es la de los árboles trie. Se analiza el problema de representar las palabras de un corrector ortográfico, motivando la necesidad de definir una estructura donde no haya redundancia debido al gran número de prefijos comunes existentes. De manera más o menos intuitiva surge la idea de los árboles de prefijos. Entonces, se define la estructura de árbol trie, donde los nodos son prefijos y las palabras son los caminos desde la raíz hasta una hoja etiquetada con marca de fin. Se plantea el uso de los árboles trie en un corrector ortográfico interactivo, consistente en ir recorriendo el árbol a la vez que se introduce el texto.

Entrando en cuestiones de implementación, se estudian las dos formas básicas de representar los nodos del árbol trie: mediante tablas o mediante listas[5]. Para cada una de ellas se propone una definición de los tipos de datos "nodo trie" y "árbol trie", y la implementación de las operaciones sobre nodos, para insertar, eliminar o consultar valores. Se plantea también la operación de inserción de una palabra en el árbol trie, que usa las operaciones sobre los nodos. La implementación de esta última es, y debe ser, independientemente de que se use una representación u otra en los nodos.

Una vez vistas las dos posibilidades, se hace un estudio comparativo del uso de memoria y de la eficiencia de la operación de consulta o inserción de una palabra nueva. En consecuencia, se analiza cuando una representación es más adecuada que la otra.

El punto 2 plantea el problema de representar relaciones binarias de equivalencia, definidas sobre conjuntos de tamaño conocido. Por lo tanto, no es estrictamente un problema de representación de conjuntos, aunque está estrechamente relacionado. Se empieza definiendo el concepto matemático de relación binaria de equivalencia, como una relación con las propiedades reflexiva, simétrica y transitiva. Se ven algunos ejemplos, como la relación de ciudades de la misma provincia, y se listan las operaciones del TAD relación de equivalencia: crear una relación vacía, unir dos clases de equivalencia y buscar la clase de un elemento.

En cuanto a la implementación, se comprueba cómo las dos soluciones sencillas (una lista por cada clase o una tabla con el número de clase de cada elemento) logran poca eficiencia (en la operación de búsqueda o en la de unión, respectivamente). Se propone entonces una representación de árboles

[5]Este segundo tipo es equivalente a una representación hijo-izquierdo hermano-derecho.

mediante punteros al padre, donde cada árbol es una clase de equivalencia. Se muestra una posible implementación de las operaciones, representando los datos mediante una tabla.

Aunque esta técnica consigue una buena eficiencia para las operaciones de búsqueda y unión, estudiando el peor caso se comprueba que la eficiencia conseguida puede no mejorar la representación con listas. Para solucionarlo, se introducen dos sencillas modificaciones en el funcionamiento de las operaciones, conocidas como equilibrado de árboles y compresión de caminos.

En el punto 3 se estudian los árboles binarios de búsqueda balanceados o AVL. Se parte de que los alumnos ya conocen los árboles binarios de búsqueda (ABB), estudiados en primer curso. A modo de recordatorio, se repasa la definición de ABB y se comprueba el peor caso de ABB, donde el funcionamiento es equivalente a una lista ordenada; esto justifica la necesidad de introducir una condición de balanceo. Primero se propone la condición basada en número de nodos de los subárboles, que da lugar a los ABB perfectamente balanceados. Luego se ve la condición menos restrictiva basada en diferencias de altura, que da lugar a los árboles AVL. Se discute qué significa y qué implicaciones tiene el que una condición sea más o menos restrictiva.

Para comprobar que con su condición de balanceo los árboles AVL tienen un buen comportamiento, se estudia el peor caso de árbol AVL –en cuanto a máximo desequilibrio permitido– resultando una fórmula recurrente similar a la definición de los números de Fibonacci. El número mínimo de nodos respecto a la altura resulta ser una función exponencial, y por lo tanto la profundidad máxima para un número dado de nodos es logarítmica.

A continuación, se plantean las operaciones de inserción y eliminación, que pueden requerir el uso de las rotaciones simples y dobles sobre AVL. Así pues, se explica el significado de estas rotaciones y se muestra su implementación. Después se explican las operaciones de inserción y eliminación, a través de un análisis de casos de las distintas posibilidades que pueden ocurrir, y se resuelven algunos ejemplos. Normalmente, el análisis de casos se hace únicamente sobre inserciones y eliminaciones en un subárbol (por ejemplo, el izquierdo), dejando como ejercicio plantear los casos simétricos en el otro subárbol. Aunque el problema no parece excesivamente complejo, en la práctica son muchos los alumnos que no saben exactamente cómo aplicar la simetría, por lo que conviene dar alguna idea o dejar disponible la solución.

Finalmente, el punto 4 contiene una introducción a los árboles B. Aunque se comentan las distintas variantes existentes (los árboles B, B+ y B*), no se profundiza en su estudio ya que serán tratadas en la asignatura de bases de datos, de tercer curso. De esta manera, se explican los árboles B como una extensión de la idea de ABB a árboles de búsqueda no binarios. Se estudia la definición y propiedades de los árboles B, y se analizan las operaciones de inserción y eliminación.

Un error frecuente en los alumnos es usar el proceso de "préstamo de entradas", definido para la eliminación en árboles B, también en la operación de inserción de elementos. Aunque el resultado puede ser un árbol B correcto, no es lo que se habría obtenido con la implementación estudiada. Es más, en muchos casos el error sólo se comete en las inserciones que habrían provocado que el árbol aumentara de altura, lo cual no ocurre al hacer el préstamo de una entrada. Esto indica que el alumno no es capaz de razonar sobre lo que ocurre al aumentar la altura del árbol en una unidad, por lo cual es importante incidir en ese aspecto.

El estudio de los árboles B acaba con un análisis de la eficiencia de la operación de búsqueda. Se comprueba cómo el número de accesos a nodos del árbol se reduce en una constante, respecto a los ABB, aunque el número total de comparaciones no se reduce.

Temporización

Tiempo estimado de clases teóricas: **5 horas**. La planificación es mostrada en la tabla 5.4.

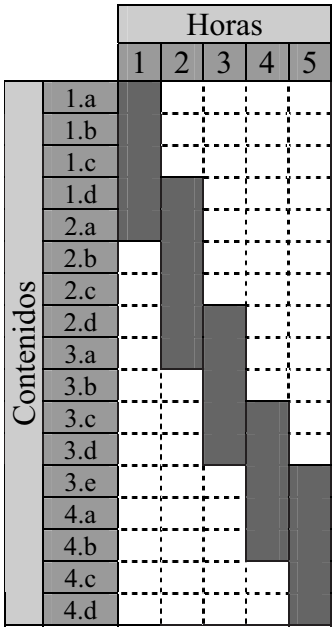


Tabla 5.4: Planificación temporal del Tema 3, Parte I.

Todos los puntos de este tema tienen una importancia similar y cada uno de ellos requiere aproximadamente un poco más de una hora de clase. Es importante hacer algunos ejemplos de cada apartado. En la planificación mostrada estos ejemplos son realizados al acabar cada uno de los puntos, en lugar de hacerlo al final del tema.

Tema 4 - Grafos

Objetivos

- Familiarizarse con la notación y terminología usada en teoría de grafos, incluyendo tipos de grafos, propiedades, conceptos y problemas típicos sobre grafos.
- Ser capaz de diseñar e implementar una estructura para el tipo de datos grafo –en sus distintas variantes–, usando listas y matrices de adyacencia.
- Analizar las ventajas e inconvenientes de las representaciones de grafos mediante listas y matrices de adyacencia, y su influencia en la eficiencia de los algoritmos sobre grafos estudiados.
- Conocer y comprender el funcionamiento de una amplia variedad de algoritmos clásicos sobre grafos (tales como los algoritmos de Prim, Kruskal, Dijkstra, Floyd y Warshall), razonando sobre las ideas subyacentes y analizando su complejidad computacional.
- Adquirir la habilidad de usar los algoritmos estudiados como herramientas para la resolución de problemas en un contexto genérico, a través de la transformación de un problema de interés en un problema sobre grafos.

- Ser capaz de diseñar algoritmos para resolver nuevas clases de problemas sobre grafos.

- Introducir, de manera informal, los conceptos de problemas P y NP a través de algunos ejemplos de problemas NP sobre grafos.

Contenidos

1. Definiciones, terminología y notación.
   - 1.a) Definición de grafo y tipos de grafos.
   - 1.b) Grado, caminos y ciclos.
   - 1.c) Conectividad y subgrafos.
   - 1.d) El tipo abstracto de datos grafo.
2. Estructuras de representación.
   - 2.a) Representación mediante matrices de adyacencia.
   - 2.b) Representación mediante listas de adyacencia.
   - 2.c) Comparación entre las estructuras básicas.
3. Recorridos sobre grafos.
   - 3.a) El concepto de recorrido. Árboles y bosques de expansión.
   - 3.b) Búsqueda primero en profundidad.
   - 3.c) Búsqueda primero en anchura.
   - 3.d) Arcos que no son del árbol de expansión.
4. Árboles de expansión de coste mínimo.
   - 4.a) Definición. Problema del árbol de expansión de coste mínimo.
   - 4.b) Algoritmo de Prim.
   - 4.c) Algoritmo de Kruskal.
5. Problemas de caminos mínimos.
   - 5.a) Caminos mínimos desde un origen y entre todos los pares.
   - 5.b) Algoritmo de Dijkstra.
   - 5.c) Optimalidad del algoritmo de Dijkstra.
   - 5.d) Algoritmo de Floyd.
   - 5.e) Cierre transitivo. Algoritmo de Warshall.
6. Algoritmos sobre grafos dirigidos.
   - 6.a) Componentes conexos y fuertemente conexos.
   - 6.b) Algoritmo para el cálculo de los componentes fuertemente conexos.
   - 6.c) Grafo reducido de un grafo dirigido.
   - 6.d) Grafos dirigidos acíclicos (GDA).
   - 6.e) Ordenación topológica de GDA.
7. Puntos de articulación y componentes biconexos.
   - 7.a) Definición de punto de articulación y componente biconexo.
   - 7.b) Algoritmo para encontrar los puntos de articulación.
8. Otros problemas sobre grafos.
   - 8.a) Flujo máximo en redes.
   - 8.b) Circuitos de Euler.
   - 8.c) Ciclos hamiltonianos.
   - 8.d) Problema del viajante.
   - 8.e) Coloración de grafos.
   - 8.f) Isomorfismo de grafos.

Bibliografía básica

Capítulo 5 de [García'03] y capítulos 6 y 7 de [Aho'88].

Bibliografía complementaria

El nivel de detalle con el que se tratan los distintos contenidos es similar al que se puede encontrar en [Aho'88], en el capítulo 9 de [Weiss'95] o en los capítulos del 7 al 9 de [Baase'00]. En el último caso, también se incluyen algunos otros problemas en el resto de capítulos del libro; algo parecido ocurre en [Brassard'90] y [Brassard'97], donde los problemas sobre grafos están repartidos a lo largo de todo el texto. Para profundizar en otros problemas no estudiados en el tema se puede utilizar bibliografía adicional. Por ejemplo, en los capítulos del 22 al 26 de [Cormen'01] se tratan los grafos de forma exhaustiva. El capítulo 26 está dedicado a los problemas de flujo en redes, sobre los cuales este tema sólo hace una pequeña introducción. En el capítulo 8 de [Drozdek'01] se tratan algunos otros problemas interesantes, como el isomorfismo de grafos (en inglés *matching*) y los ciclos hamiltonianos y de Euler.

Descripción de los contenidos

En este último tema de la Parte I se estudian los grafos: terminología, representación y problemas sobre grafos. Aunque incluido en la parte de estructuras de datos, la mayoría de los aspectos tratados en el tema hacen referencia a la resolución de problemas con grafos, y sólo el segundo punto trata explícitamente las cuestiones de representación. No obstante, ya hemos justificado el interés de que estos problemas sean tratados aquí y no dispersos entre los temas de algorítmica. En cierto sentido, se puede decir que el tema sirve de transición hacia la parte de algorítmica de la asignatura.

Se debe suponer cierta familiaridad de los alumnos con la teoría de grafos, que es estudiada en la asignatura Matemática Discreta de primer curso. No hay, sin embargo, ninguna redundancia de contenidos, ya que aquí se verán los grafos desde una perspectiva muy distinta, teniendo en cuenta los aspectos de programación; cuestiones como el uso de memoria, la implementación y el análisis de eficiencia de los algoritmos son en esta asignatura los puntos relevantes a considerar. El tema es claramente el más largo del temario e incluye el estudio de una gran variedad de algoritmos clásicos sobre grafos (como los debidos a Prim, Kruskal, Dijkstra, Floyd y Warshall) además de otros problemas que pueden resultar de interés en la formación de los alumnos.

El punto 1 empieza con un necesario repaso de definiciones y terminología de la teoría de grafos. A través de algunos ejemplos, se muestra la gran variedad de aplicaciones en las que se requiere el uso de grafos, desde el caso típico de una red de carreteras hasta las redes de conocimiento usadas en inteligencia artificial. Estos ejemplos servirán para ir mostrando sobre ellos las definiciones de los conceptos que se van exponiendo.

En particular, se definen los conceptos de vértice, arista, grafo dirigido y no dirigido, grafo etiquetado, grafo con pesos, adyacencia entre dos vértices, camino en el grafo, camino simple, longitud de un camino, ciclo, ciclo simple, grado de un vértice, subgrafos, subgrafos conexos o fuertemente conexos y grafo completo. Después se plantea la visión de los grafos como TAD, mostrando la necesidad de definir un TAD distinto por cada tipo de grafo; se enumeran las operaciones que forman parte de la especificación de los TAD correspondientes. No se aborda la especificación formal del tipo, ya que no se considera suficientemente interesante su desarrollo en clase de teoría. Sin embargo, sí podría resultar adecuado plantearlo como un ejercicio a resolver por los alumnos.

Las técnicas de representación de grafos son estudiadas en el punto 2. Se explican los dos esquemas básicos –mediante matrices y listas de adyacencia– considerando cómo serían las definiciones de las estructuras para los casos de grafos dirigidos o no dirigidos, y etiquetados o no etiquetados.

Después se comparan ambas estructuras, en cuanto a uso de memoria y en cuanto a eficiencia de algunas operaciones. Las operaciones analizadas son muy sencillas y son del tipo: calcular el grado de entrada y de salida de un nodo o contar el número total de aristas. Como en la mayoría de los problemas, se llega a la conclusión de que cierta estructura será mejor para determinadas aplicaciones y menos adecuada para otras. Se plantea también la posibilidad de usar la estructura de listas múltiples, estudiada en el Tema 2, para mejorar la eficiencia de algunas operaciones.

En los puntos del 3 al 8 se estudia una variedad de problemas sobre grafos, desde los más básicos hasta otros más complejos para los que sólo se define el objetivo del problema. Para empezar, en el punto 3 se tratan los recorridos sobre grafos. Se expone primero la idea de recorrido de un grafo y las posibles estrategias, en profundidad y en anchura; estos recorridos dan lugar a los árboles y bosques de expansión (o abarcadores) en profundidad o en anchura, respectivamente. Después se plantea la implementación de los dos tipos de recorrido: en profundidad usando recursividad y en anchura usando una cola. Se analiza la complejidad de ambas operaciones –que obviamente depende del tipo de representación utilizado– y se explican algunas aplicaciones sencillas de los recorridos, por ejemplo, contar el número de componentes conexos de un grafo no dirigido.

Es importante que los alumnos vean las operaciones de recorrido como herramientas flexibles que se pueden aplicar para resolver muchos problemas sobre grafos. En este sentido, en algunos casos puede resultar interesante usar los arcos que no pertenecen al árbol de expansión asociado a un recorrido (los llamados arcos de avance, retroceso o cruce), por lo que también es estudiada la forma de encontrarlos.

El problema de los árboles de expansión de coste mínimo es tratado en el punto 4. Se define el coste de un árbol de expansión y se justifica mediante algún ejemplo el interés de calcular, para un grafo con pesos dado, su árbol de expansión con mínimo coste. Se presentan los dos algoritmos clásicos para resolverlo, debidos a Prim y Kruskal. Ambos utilizan la idea de los algoritmos voraces, que serán estudiados más adelante; aun así, su comprensión no suele resultar difícil para los alumnos, ya que conceptualmente el funcionamiento de ambos no es complejo. De los dos algoritmos se muestran los esquemas en pseudocódigo, se realiza algún ejemplo de aplicación y se analiza a grandes rasgos su complejidad computacional. Se discuten las similitudes de fondo entre ambos algoritmos, con lo cual se empieza a sugerir la idea de esquema algorítmico, aunque sin profundizar en ella por el momento.

El punto 5 expone los problemas de caminos mínimos, para los cuales se estudian los algoritmos también clásicos de Dijkstra (entre un origen y todos los demás vértices) y de Floyd (entre todos los pares de vértices). Igual que antes, no consideramos un inconveniente el no haber estudiado los algoritmos voraces para asimilar el funcionamiento del algoritmo de Dijkstra; lo mismo ocurre con el algoritmo de Floyd, basado en los principios de la programación dinámica. Para ambos algoritmos, se empieza explicando el fundamento de los mismos, después se muestra una implementación en pseudocódigo, se realizan algunos ejemplos y finalmente se analiza la eficiencia según el tipo de representación usado. Además, en el algoritmo de Dijkstra se demuestra que los resultados obtenidos son siempre los caminos mínimos, es decir que el algoritmo es óptimo. Puede ser interesante proponer a los alumnos modificar los algoritmos para que resuelvan problemas parecidos, como por ejemplo determinar si los caminos mínimos son únicos o no.

En el subapartado 5e) se presenta el algoritmo de Warshall, como una modificación del algoritmo de Floyd para resolver el problema del cierre transitivo de una matriz de adyacencia[6]. La explicación de este algoritmo tiene el interés de mostrar a los alumnos cómo muchas veces dado un problema sobre grafos con pesos, se puede definir un problema similar o equivalente sobre grafos no etiquetados, y viceversa. En tales casos, una simple adaptación puede servir para resolver los dos problemas.

[6]Aunque históricamente no se puede considerar el algoritmo de Warshall como una modificación del algoritmo de Floyd, sino más bien todo lo contario.

A continuación, se tratan en el punto 6 dos cuestiones relacionadas con grafos dirigidos: los componentes fuertemente conexos y los grafos dirigidos acíclicos (GDA). Lo primero es estudiado en los subapartados 6a), 6b) y 6c). Se empieza repasando la definición de componente fuertemente conexo, justificando el interés de calcularlos. Mientras que para el caso de los grafos no dirigidos ya se vio la forma de calcular los componentes conexos, con una simple aplicación de la búsqueda primero en profundidad, se comprueba cómo es posible utilizarla también en el caso de los grafos dirigidos, pero de forma no tan directa. Se justifica el algoritmo, basado en dos búsquedas en profundidad, donde la primera indica un orden de recorrido para la segunda. Como aplicación sencilla, se muestra la definición del grafo reducido asociado a un grafo dirigido, donde cada componente conexo del segundo es sustituido por un simple nodo en el primero.

Los GDA son estudiados en los subapartados 6d) y 6e). Se motiva su interés viendo una variedad de aplicaciones que utilizan grafos dirigidos, donde las propias restricciones del problema implican necesariamente que no pueden existir ciclos. Se muestra el concepto matemático subyacente: la representación de órdenes parciales sobre conjuntos. Después se estudia la ordenación topológica, como característica peculiar de los GDA; se da su definición formal y se interpreta el significado de una ordenación topológica aplicada sobre los grafos vistos en las aplicaciones de ejemplo. Por ejemplo, la ordenación topológica en un grafo de tareas indica un orden válido en el que realizar las subtareas. Se razona sobre cómo proceder para calcular una ordenación topológica, llegando al diseño de un algoritmo. Para completar el estudio se analiza la complejidad del algoritmo.

Los conceptos de punto de articulación y componente biconexo –asociados a grafos no dirigidos– son desarrollados en el punto 7. Ambos conceptos están estrechamente relacionados entre sí y también con la conectividad de un grafo no dirigido. Igual que antes, se introduce primero la definición y después se aplica sobre ejemplos concretos; de esta forma la idea general adquiere un significado concreto y más fácil de comprender. Se muestra el algoritmo para calcular los puntos de articulación de un grafo no dirigido, que hace uso de la búsqueda primero en profundidad. En este caso, para facilitar la comprensión del algoritmo, creemos que lo más adecuado es mostrar la ejecución sobre un ejemplo y justificar su funcionamiento sobre ese ejemplo. De esta manera, la idea es asimilada con más facilidad, puesto que la base teórica no es tan evidente.

Aparte de los anteriores algoritmos, es adecuado que los alumnos adquieran una visión amplia de las aplicaciones y problemas más habituales sobre grafos, siendo capaces de transformar un problema particular en uno sobre grafos, siempre que sea posible. Este repaso de problemas sobre grafos es realizado en el último punto, el punto 8. Así, en el subapartado 8a) se presenta el uso de grafos para representar flujos. En concreto, se estudia el problema del flujo máximo, para el cual se da un algoritmo voraz sencillo. Se comprueba que este algoritmo no es óptimo, introduciendo algunas modificaciones en su funcionamiento para obtener un algoritmo óptimo.

Después se ve el problema de los circuitos de Euler en el subapartado 8b). Se exponen las condiciones necesarias y suficientes para que exista un circuito de Euler en un grafo no dirigido. Se presenta un algoritmo para calcular un circuito de Euler, que hace uso del recorrido en profundidad.

Para los cuatro problemas restantes simplemente se define el objetivo de cada uno y se da una visión de las aplicaciones donde pueden ser útiles. En particular, se enuncian los problemas del ciclo hamiltoniano, del viajante, de la coloración y del isomorfismo de grafos. No se plantea en este tema ningún algoritmo para resolverlos. De hecho, todos ellos resultan ser problemas NP-completos, por lo que no se conocen algoritmos eficientes para resolverlos. Su presentación en este punto es un buen momento para introducir informalmente la clasificación de problemas P y NP, y la necesidad de usar heurísticas para resolver problemas computacionalmente costosos. Cuando se estudien las técnicas de diseño de algoritmos, los alumnos deberán ser capaces de abordar estos problemas, diseñando algoritmos que los resuelvan de forma exacta o inexacta pero eficiente.

Temporización

Tiempo estimado de clases teóricas: **10 horas**. La planificación es mostrada en la tabla 5.5.

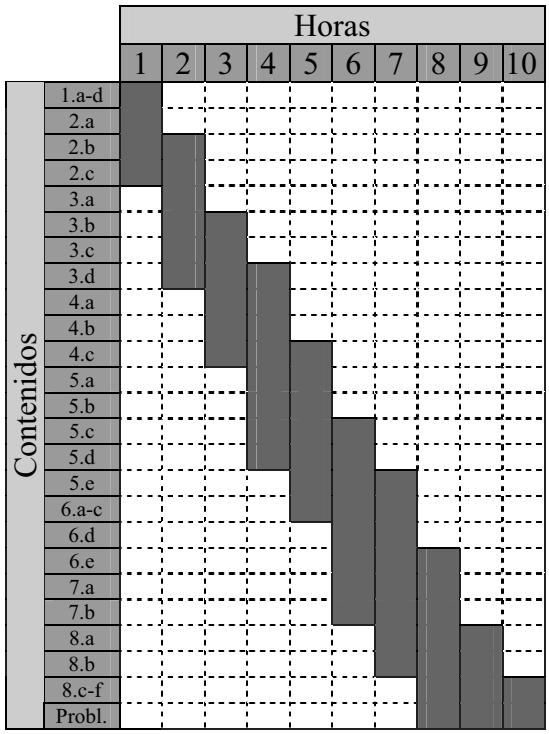


Tabla 5.5: Planificación temporal del Tema 4, Parte I.

Este tema es, con diferencia, el que más horas de teoría requiere. Debido a ello, las diferencias en la planificación entre seguir un ritmo más rápido o más pausado se hacen más notables, como puede verse en la tabla 5.5. En el mejor caso puede ser suficiente con unas 8 horas, quedando tiempo para la realización de ejercicios. En el más lento, 10 horas de clase deberían ser suficientes para completar todos los contenidos. De cualquier forma, es adecuado realizar ejemplos junto con las explicaciones y no únicamente al final del tema.

Capítulo 6

Programa de Teoría: Parte II - Algoritmos

"Sólo existen 10 tipos de personas:
los que entienden el sistema binario y los que no."
A.M. Bronstein

Tema 5 - Análisis de algoritmos

Objetivos

- Concienciarse de la importancia del estudio de los recursos consumidos por un algoritmo y del interés de hacerlo antes de la implementación, entendiendo que para cada aplicación puede variar lo que sea considerado un recurso crítico.
- Saber determinar en función de qué parámetros está dado el tamaño de un problema y qué otros factores afectan al consumo de recursos.
- Comprender, distinguir y relacionar los conceptos de eficiencia, complejidad computacional y tiempo de ejecución de un algoritmo, en los casos mejor, peor y promedio.
- Conocer y usar correctamente y con soltura las notaciones de complejidad O, Ω, Θ y o-pequeña, comprendiendo su significado, utilidad y limitaciones.
- Conocer las cotas de complejidad que aparecen más frecuentemente en el estudio de algoritmos, así como la relación entre los órdenes de estas funciones.
- Saber calcular el tiempo de ejecución, el orden de complejidad y el uso de memoria de una gran variedad de tipos de algoritmos, usando las técnicas básicas de conteo de instrucciones y de memoria.

Contenidos

1. Consumo de recursos de un algoritmo.
   - 1.a) Recursos consumidos por un algoritmo y otros criterios.
   - 1.b) La eficiencia como relación recursos consumidos / resultados obtenidos.
   - 1.c) Factores externos e internos en la medición de recursos.
   - 1.d) La notación $t(n)$. Caso más favorable, más desfavorable y promedio.
2. Tipos de análisis de algoritmos.
   - 2.a) Análisis a priori y a posteriori.
   - 2.b) Análisis teórico y experimental.
   - 2.c) El proceso de conteo de instrucciones y de memoria.
3. Notaciones asintóticas.
   - 3.a) Necesidad y aplicación de las notaciones asintóticas.
   - 3.b) La notación O. Definición formal e interpretación. Relación de orden.
   - 3.c) La notación Ω. Definición formal e interpretación. Relación de orden.
   - 3.d) La notación Θ. Definición formal e interpretación.
   - 3.e) La notación o-pequeña. Definición formal e interpretación.
4. Extensiones y propiedades de las notaciones asintóticas.
   - 4.a) Propiedades de las notaciones asintóticas.
   - 4.b) Notaciones con varios parámetros.
   - 4.c) Notaciones condicionales.
   - 4.d) Cotas de complejidad más frecuentes.
   - 4.e) Repaso de fórmulas para el cálculo de series.

Bibliografía básica

Capítulos 6 y 7 de [Giménez'03] y capítulos 2 y 3 de [Brassard'97].

Bibliografía complementaria

La mayoría de los libros clásicos de programación y algoritmos empiezan con capítulos dedicados a las nociones básicas del análisis de algoritmos, como [Aho'74], [Aho'88], [Brassard'90], [Cormen'90], [Horowitz'78], [Wirth'80] y [Wirth'87]. El acercamiento usado en este tema es similar al propuesto en [Brassard'97]. Antes de empezar el tema puede ser adecuado repasar algunas herramientas matemáticas necesarias para hacer el tipo de estudios que aparece en este tema: series, límites, integrales, etc. Algunos de los libros anteriores también incluyen capítulos de repaso de matemáticas, como el libro clásico [Knuth'85] y los más recientes [Cormen'01], [Brassard'97], [Heileman'98] y [Baase'00]. Posiblemente, el aspecto más interesante para profundizar en este tema son los métodos alternativos de análisis de algoritmos, como el análisis probabilístico y el análisis amortizado.

Descripción de los contenidos

Este es el primer tema de la parte de algorítmica de la asignatura. En particular, este tema junto con el siguiente forman un bloque de análisis de algoritmos. Aunque hay un cambio notable en el tipo de cuestiones tratadas, hay que tener en cuenta que ya se han realizado análisis –de tiempo y de uso de memoria– en algunos de los temas de la parte de estructuras de datos. Para llevarlos a cabo se supone que era suficiente con los conocimientos básicos que los alumnos adquirieron en primer curso.

En consecuencia, en este tema se trata de profundizar en el análisis de algoritmos, desde un punto de vista matemáticamente más riguroso, pero a la vez que sea aplicable al estudio de los

problemas que nos interesan. El tema está centrado en el estudio de las notaciones asintóticas, sus propiedades y extensiones. Se intenta desarrollar en los alumnos la habilidad de manejar con soltura las distintas notaciones existentes, que de hecho serán utilizadas ampliamente en los sucesivos temas de la asignatura. Todos los conceptos que se introducen son definidos formalmente primero y después se interpreta su significado de manera más intuitiva.

El tema empieza motivando, en el punto 1, el interés de profundizar en el análisis de algoritmos. Se hace un repaso de la definición de algoritmo, incidiendo ahora en el hecho de que su ejecución consume recursos; se discuten cuáles son los tipos de recursos consumidos, sin perder de vista otros criterios subjetivos que pueden hacer un algoritmo más o menos adecuado (como la facilidad de programarlo). Entonces, se define la eficiencia como el criterio "empresarial" de maximizar la relación entre productos obtenidos y recursos consumidos. Es básico que los alumnos tengan en mente no sólo el tiempo y la memoria que requiere un algoritmo, sino también qué es lo que resuelve, es decir, si encuentra siempre solución o si garantiza la solución óptima, en su caso.

Se muestra que no tiene sentido estudiar los recursos de manera absoluta, ya que dependen de factores externos e internos, de los cuales el análisis debería ser independiente. De esta forma, se llega a la definición de la notación $t(n)$, del tiempo de ejecución. Aunque la notación ya es conocida por los alumnos, conviene aclarar algunos aspectos de su significado, como por ejemplo ¿qué unidades mide $t(n)$? ó ¿qué ocurre si hay otros factores que influyen en el tiempo además de n? Esto da pie a la introducción de los casos más favorable, más desfavorable y promedio. Es adecuado advertir a los alumnos sobre el error frecuente de pensar que el caso más favorable equivale a un tamaño de entrada pequeño.

En el punto 2 se tratan las distintas alternativas en el análisis de algoritmos: hacer el análisis a priori o a posteriori, y hacer un estudio teórico o experimental. Aunque gran parte de lo estudiado en la asignatura está orientado al análisis teórico a priori, es interesante que los alumnos conozcan la posibilidad de hacer estudios a posteriori, ya sean teóricos o experimentales. Es más, en algunas ocasiones el estudio experimental puede ser la única alternativa viable. Se trata primero la distinción entre análisis a priori y a posteriori, viendo en qué situaciones tiene utilidad cada tipo. Después se diferencia entre estudio teórico y experimental, y se realizan algunas consideraciones sobre el análisis experimental, basado en mediciones de tiempos, para su representación gráfica o análisis usando técnicas de regresión y ajuste estadístico. En el subapartado 2c) se explica el proceso básico para hacer el estudio teórico de un algoritmo, conocido como conteo de instrucciones o de memoria. Este proceso es aplicado sobre algunos ejemplos de algoritmos.

El estudio formal de las notaciones asintóticas se encuentra en el punto 3. En primer lugar se justifica el uso de estas funciones como una manera de simplificar notación, lo cual es visto a la vez como una ventaja y una limitación. El alumno ya conoce de manera informal la notación O (o-grande), que ha trabajado hasta el momento con ejemplos sencillos. La asimilación de la definición formal no suele ser sencilla, aunque es esencial para avanzar en el estudio de las notaciones de complejidad; para ello, una interpretación gráfica de la idea subyacente en la definición es muy orientativa. Se define también la relación de orden dentro de la notación O, dada por la relación de contenido entre conjuntos.

A continuación se estudia la notación Ω (o de cota inferior) y su relación de orden, la notación Θ (o de orden exacto) y la o-pequeña. A medida que se introducen nuevas notaciones se va analizando su relación con las ya conocidas, de manera que no son tratadas como conceptos aislados. Igual que antes, se presenta primero la definición matemática y luego se interpreta gráficamente o mediante ejemplos. Por otro lado, conviene realizar ejercicios sobre el manejo de las notaciones según se explican, sin esperar necesariamente al final del tema para hacerlos.

Siguiendo con las notaciones asintóticas, en el punto 4 se estudian algunas de las propiedades y extensiones más importantes. Las propiedades incluyen: transitividad, relaciones pertenencia/inclusión, la propiedad del máximo, relaciones entre límites y órdenes, y relaciones entre notaciones. Estas propiedades ayudan a conocer mejor el significado de las notaciones y a facilitar su manejo. Es interesante realizar la demostración formal de alguna de las propiedades expuestas; el número de las que se realicen depende del ritmo de ejecución de la planificación temporal, ya que no es un contenido prioritario.

A continuación, se plantean dos extensiones de las notaciones de complejidad. La primera de ellas consiste en usar funciones con varios parámetros, y resulta de utilidad cuando son varios los valores que indican el tamaño del problema. Por ejemplo, si el tiempo depende de dos parámetros, la función del tiempo tendrá la forma $t(n, m)$. Todas las notaciones –y, consecuentemente, las propiedades asociadas– son extendidas para usar funciones de este tipo. La segunda extensión permite definir órdenes de complejidad condicionados, del tipo $O(t(n)|p(n))$; el significado es que sólo es necesario comprobar la propiedad del orden cuando la proposición $p(n)$ sea cierta. Un caso típico es el condicionamiento a que el tamaño de entrada sea potencia de 2, expresado de forma simplificada como: $O(t(n)|n = 2^k)$. Un ejercicio interesante para los alumnos es plantear la cuestión: ¿es cierto que $O(t(n)) \subset O(t(n)|p(n))$ o que $O(t(n)) \supset O(t(n)|p(n))$?

Para acabar el punto 4, se enumeran algunas de las cotas de complejidad que suelen aparecer más frecuentemente, como por ejemplo $O(log_2 n)$, $O(n)$, $O(n^2)$, etc. Se discute y establece una ordenación de menor a mayor entre las mismas. Se repasan también algunas fórmulas matemáticas –como las de las series aritméticas y geométricas– y la técnica de aproximación a un sumatorio mediante integrales. Se da por supuesto que los alumnos han adquirido estos conocimientos en las asignaturas de matemáticas, si bien su falta de práctica hace necesario el rápido repaso que aquí se incluye. Obviamente, todas estas fórmulas están orientadas a la resolución de los sumatorios que aparecen en el análisis de algoritmos iterativos.

Temporización

Tiempo estimado de clases teóricas: **6 horas**. La planificación es mostrada en la tabla 6.1.

La mayor parte de la carga lectiva del tema se encuentra en el punto 3 y en el subapartado 4a), donde se estudian las propiedades de las notaciones asintóticas. Como se ha comentado anteriormente, la duración de este subapartado puede ajustarse incluyendo o no las demostraciones de las propiedades; el tiempo necesario para incluirlas todas es alrededor de una hora y media. Al final del tema se dispone de tiempo para hacer algunos ejercicios adicionales.

Tema 6 - Ecuaciones de recurrencia

Objetivos

- Repasar el concepto de recursión y las cuestiones ligadas a la construcción de algoritmos recursivos.
- Ser capaz de analizar algoritmos recursivos, aplicando los resultados del análisis en comparación de algoritmos, evaluación de resultados experimentales y predicción de tiempos de ejecución.
- Conocer y saber aplicar la técnica del polinomio característico para la resolución de ecuaciones recurrentes lineales y homogéneas, y su extensión a algunos casos no homogéneos o no lineales.

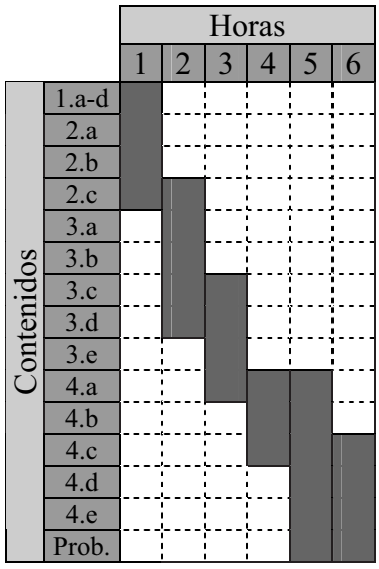


Tabla 6.1: Planificación temporal del Tema 5, Parte II.

- Conocer algunas técnicas alternativas para la resolución de ecuaciones de recurrencia no lineales.
- Comprender el papel de las condiciones iniciales en la resolución de las ecuaciones de recurrencia, determinando justificadamente cuáles deben ser usadas en cada caso.
- Identificar las ecuaciones recurrentes más frecuentes en el estudio de algoritmos y los órdenes de complejidad a los que dan lugar.

Contenidos

1. Ecuaciones de recurrencia lineales.
   - 1.a) Algoritmos recursivos.
   - 1.b) Ecuaciones de recurrencia en análisis de algoritmos.
   - 1.c) Formato general de las ecuaciones de recurrencia.
   - 1.d) Resolución de ecuaciones recurrentes lineales y homogéneas.
     - Transformación de la ecuación recurrente a una ecuación característica.
     - Caso de soluciones con multiplicidad uno.
     - Caso de soluciones con multiplicidad mayor que uno.
     - Cálculo de las constantes.
   - 1.e) Resolución de ecuaciones recurrentes lineales y no homogéneas.
     - Eliminación de la no homogeneidad.
     - Resumen del método general del polinomio característico.
   - 1.f) Cambio de variable. Eliminación de la condición de cambio.
2. Ecuaciones de recurrencia no lineales.
   - 2.a) Método de expansión de recurrencias.
   - 2.b) Método de transformación de la imagen.
   - 2.c) Método de inducción constructiva.
3. Condiciones iniciales en ecuaciones recurrentes.

3.a) Significado y aplicación de las condiciones iniciales.
3.b) Previsión del tiempo de ejecución.

Bibliografía básica

Capítulo 8 de [Giménez'03] y capítulo 4 de [Brassard'97].

Bibliografía complementaria

Aunque las ecuaciones de recurrencia aparecen en muchos libros de algoritmos y estructuras de datos, la aproximación que se propone en este tema es más formal y completa que la planteada en gran parte de ellos, como ocurre con [Aho'88] en el capítulo 9, o [Weiss'95] en el capítulo 2. La referencia más interesante es la sección 4.7 de [Brassard'97]. También se puede consultar el capítulo 4 de [Cormen'01] y el capítulo 3 de [Baase'00]. Teniendo en cuenta la naturaleza netamente matemática de este tema, para encontrar más información sobre las ecuaciones de recurrencia se pueden usar libros de matemáticas, aunque con los libros antes referenciados resulta suficiente.

Descripción de los contenidos

En este segundo tema del bloque de análisis de algoritmos se trata la resolución de ecuaciones de recurrencia, las cuales están estrechamente ligadas al estudio de algoritmos recursivos. El núcleo de interés del tema es la técnica conocida como técnica del polinomio característico, que resulta muy útil para resolver una amplia variedad de ecuaciones recurrentes. La técnica es presentada con el rigor matemático necesario, pero buscando en última instancia la utilidad práctica del método para el análisis de algoritmos recursivos. De esta forma, se realizan demostraciones en las partes que consideramos adecuadas, pero finalmente se llega a un proceso de resolución cuya aplicación resulta relativamente sencilla y casi automática.

Aunque la mayoría de los contenidos presentados son de carácter netamente matemático, su estudio es necesario para resolver los tiempos de ejecución de muchos de los algoritmos recursivos que se verán en los temas sucesivos. Por lo tanto, el hecho de que no se incluya en las asignaturas de matemáticas obliga a estudiarlo aquí.

Se empieza, en el punto 1, haciendo un rápido repaso del concepto de recursividad y su aplicación en la construcción de algoritmos –cuestiones extensamente estudiadas en primer curso–. A continuación se realiza el conteo de instrucciones de un algoritmo recursivo sencillo –como el de las torres de Hanoi– resultando que el tiempo viene dado en función del mismo tiempo para tamaños menores, es decir, tenemos una ecuación recurrente. De esta forma se motiva la necesidad de estudiar las ecuaciones recurrentes y su resolución. Se muestra el formato general de las ecuaciones de recurrencia, de las cuales se empezarán estudiando los casos más sencillos. En primer lugar se trata el caso de las recurrencias lineales y homogéneas, es decir de la forma:

$$a_0 t(n) + a_1 t(n-1) + ... + a_k t(n-k) = 0 \tag{6.1}$$

A través del ejemplo sencillo $t(n) = xt(n-1)$, se llega al resultado $t(n) = x^n$. Entonces se propone sustituir en la ecuación general $t(n)$ por x^n, llegando a la ecuación característica asociada a la ecuación recurrente:

$$a_0 x^k + a_1 x^{k-1} + ... + a_k = 0 \tag{6.2}$$

La resolución de esta ecuación y la expansión de $t(n)$ según las soluciones de x, son la base de la técnica del polinomio característico. Se trata primero el caso más sencillo, donde todas las soluciones

para x son distintas, es decir, el caso de multiplicidad 1. En esta situación, la expansión de $t(n)$ viene dada por una suma de cada solución, elevada a n, y multiplicada por una constante distinta. Tenemos algo del tipo:

$$t(n) = c_1 s_1^n + c_2 s_2^n + ... + c_k s_k^n \tag{6.3}$$

Donde $\{s_1, s_2, ..., s_k\}$ son las soluciones de x, y $c_1, c_2, ..., c_k$ son las constantes, cuyo valor es obtenido aplicando las condiciones iniciales o casos base.

Después se ve que si la multiplicidad de una solución es mayor que 1 aparece otro tipo de términos, lo cual es demostrado aplicando derivadas en el polinomio característico. Se estudia cuáles son esos términos y se resuelven algunos ejemplos.

En el subapartado 1e) se tratan las recurrencias lineales y no homogéneas. A través de un ejemplo, se ve un modo de eliminar la no homogeneidad de la ecuación recurrente; con esto, llegamos a una ecuación del tipo tratado en el subapartado anterior y podemos aplicar la técnica del polinomio característico. Aunque este proceso es fácil de entender por los alumnos, es largo de aplicar y puede ser sustituido por una sencilla regla general, que transforma cada término no homogéneo en un conjunto de soluciones de la ecuación característica. De esta forma, se puede definir el método general del polinomio característico, aplicado sobre ecuaciones lineales de recurrencia, ya sean homogéneas o no. Se vuelven a hacer algunos ejemplos, cada vez con ecuaciones más complejas.

En la práctica, hay muchos casos donde la aplicación del método requerirá previamente aplicar un cambio de variable. Esto ocurre cuando $t(n)$ no está definido sobre $t(n-1)$, $t(n-2)$, ..., sino sobre términos de la forma $t(n/b)$, $t(n/b^2)$, ... El proceso de cambio de variable es estudiado en el subapartado 1f), a través de dos ejemplos típicos. Se razona que, puesto que se ha aplicado un cambio de variable, los resultados serán válidos siempre y cuando se cumpla la condición del cambio. Entonces se enuncia un teorema que permite eliminar la condición dentro de un orden condicionado, si se cumplen una serie de propiedades sobre las funciones implicadas.

En el punto 2 se estudian tres métodos alternativos para resolver ecuaciones de recurrencia, que pueden ser usados cuando la recurrencia es no lineal. El más básico es el método de la expansión de recurrencias. Esta técnica fue estudiada por los alumnos en primer curso, de acuerdo con los programas actuales, por lo que se trata simplemente de hacer un repaso. A continuación se ve mediante un ejemplo la técnica de transformación de la imagen, consistente en aplicar alguna transformación sobre ambos lados de la ecuación recurrente, para intentar eliminar la no linealidad y aplicar el método del polinomio característico. Después se ve el método de la inducción constructiva, donde se propone un orden de complejidad, cuya validez es luego demostrada por inducción. Decidir cuál es el orden propuesto queda a la intuición de la persona que aplica el método.

Hay que tener en cuenta que estas dos últimas técnicas son complejas –y la mayoría de las veces de muy difícil aplicación–, por lo que no interesa profundizar en ellas más allá de la resolución de los ejemplos planteados.

Finalmente, en el punto 3 se realizan algunas consideraciones sobre las condiciones iniciales usadas para resolver las constantes en las ecuaciones de recurrencia. Aunque la cuestión ya ha sido planteada en los puntos anteriores, es importante aclarar el significado y aplicación de las condiciones iniciales, ya que normalmente en los casos no triviales los alumnos tienen dificultad para justificar qué valores deben ser usados[1]. Se realizan algunos ejemplos, interpretando el cálculo de las constantes como el ajuste de una función a una muestra de datos. Relacionado con esta idea, se muestra la posibilidad de usar el análisis teórico para realizar previsiones del tiempo de ejecución. En este caso, el ajuste puede ser entendido como un ajuste estadístico y la previsión se convierte en la extrapolación

[1] Según las condiciones aplicadas se pueden obtener resultados distintos, pero sólo uno de ellos es el correcto.

de una función. Esta aplicación es desarrollada más extensamente en la parte de prácticas de la asignatura.

Temporización

Tiempo estimado de clases teóricas: **5 horas**. La planificación es mostrada en la tabla 6.2.

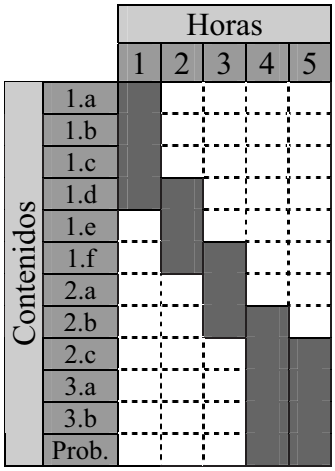


Tabla 6.2: Planificación temporal del Tema 6, Parte II.

El estudio de la técnica del polinomio característico, en el punto 1, es la parte central del tema. Su explicación completa puede requerir en torno a 2 horas y media. En todos los puntos se debe incluir la realización de numerosos ejemplos. Además, se han dispuesto las horas suficientes al final del tema para realizar algunos ejercicios más que incluyan el proceso completo de análisis, empezando por el conteo de instrucciones de un algoritmo dado.

Tema 7 - Divide y vencerás

Objetivos

- Comprender los fundamentos de la técnica de diseño de algoritmos basada en divide y vencerás, identificando su estructura en algunos algoritmos ya conocidos.
- Conocer tanto las ventajas como las limitaciones en la aplicación de la técnica de divide y vencerás.
- Ser capaz de valorar la posibilidad y conveniencia de aplicar divide y vencerás sobre cierto problema, entendiendo que puede existir más de una forma en que la técnica sea aplicada.
- Analizar el tiempo de ejecución de los algoritmos de divide y vencerás, al menos para el peor caso.
- Conocer la influencia del tamaño del caso base en el tiempo de ejecución, resolviendo problemas de obtención de tamaños óptimos del caso base.

Contenidos

1. Método general de divide y vencerás.
   - 1.a) La idea de divide y vencerás.
   - 1.b) Esquemas recursivo y no recursivo. Ejemplos sencillos.
   - 1.c) Condiciones para poder aplicar divide y vencerás.
   - 1.d) La técnica de reducción o simplificación.
2. Análisis de tiempos de ejecución.
   - 2.a) División en 2 subproblemas de igual tamaño.
   - 2.b) División en a subproblemas de tamaño n/b y combinación en $O(n)$.
3. Ordenación por mezcla y ordenación rápida.
   - 3.a) Interpretación divide y vencerás de la ordenación por mezcla.
   - 3.b) Complejidad de la ordenación por mezcla.
   - 3.c) Estudio teórico del tamaño óptimo del caso base.
   - 3.d) Interpretación divide y vencerás de la ordenación rápida.
   - 3.e) Complejidad de la ordenación rápida.
4. Multiplicación rápida de enteros largos.
   - 4.a) Representación de enteros largos. Algoritmo clásico de multiplicación.
   - 4.b) Aplicación de divide y vencerás. Algoritmo sencillo.
   - 4.c) Multiplicación rápida de Karatsuba y Ofman. Complejidad.
   - 4.d) Estudio teórico del tamaño óptimo del caso base.
5. Multiplicación rápida de matrices.
   - 5.a) Algoritmo clásico de multiplicación matricial.
   - 5.b) Aplicación de divide y vencerás. Algoritmo sencillo.
   - 5.c) Multiplicación rápida de Strassen. Complejidad.
6. El problema de selección.
   - 6.a) Algoritmo directo mediante ordenación.
   - 6.b) Algoritmo de reducción usando el procedimiento pivote.

Bibliografía básica

Capítulo 9 de [Giménez'03] y capítulo 7 de [Brassard'97].

Bibliografía complementaria

Aparte de las referencias incluidas en la bibliografía básica, existen varios libros donde la técnica de divide y vencerás es estudiada en capítulos o apartados separados. Por ejemplo, se puede consultar el apartado 10.1 de [Aho'88], el apartado 10.2 de [Weiss'95], el capítulo 4 de [Levitin'03], y en el apartado 4.3 de [Baase'00]. Por otro lado, hay libros que incluyen algunas ideas y aplicaciones de divide y vencerás dentro de capítulos dedicados a algún tipo de problemas, como [Kruse'89], [Manber'89], [Cormen'01], [Harel'92]. El problema de ordenación, al ser uno de los más básicos, se puede encontrar en la mayoría de los libros de programación. Además de los antes mencionados, se pueden usar [Aho'74], [Wirth'80], [Horowitz'82], [Baase'83], [Wirth'87], [Gonnet'91]. Normalmente estos algoritmos aparecen en capítulos dedicados a problemas de ordenación junto con otros métodos directos. Los problemas de multiplicación rápida de enteros largos, de matrices y otros problemas aritméticos son también una posibilidad donde profundizar, y se pueden encontrar en muchos de los libros anteriores. Otro tipo interesante de aplicaciones de divide y vencerás, que no son incluidas en este tema, son los problemas geométricos y gráficos, como los que se tratan en las secciones 10.2.2 de [Weiss'95], el apartado 3.3 de [Wirth'80] y el capítulo 33 de [Cormen'01].

Descripción de los contenidos

Con este tema comienza el bloque de diseño de algoritmos, dentro de la parte de algorítmica de la asignatura. La técnica de divide y vencerás (en adelante, DV) ya es conocida por los alumnos, que la estudiaron en primer curso como un ejemplo de uso de la recursividad. Este hecho, unido a la sencillez conceptual de su funcionamiento, hace de este método un buen candidato para ser estudiado en primer lugar. En el tema los alumnos profundizarán en la técnica de DV, en cuestiones tales como el uso de descomposiciones no triviales de los problemas o el estudio del tamaño más adecuado del caso base.

A lo largo del tema se desarrollan una serie de ejemplos de aplicación, donde los algoritmos de DV mejoran los órdenes de complejidad de las variantes directas[2]. En todos ellos, además, la manera de aplicar DV no es única, por lo que se deduce que la aplicación de una técnica de diseño no es un proceso automático; aunque se intenta que sea un proceso lo más sistemático posible. Está claro, no obstante, que el objetivo último no es aprenderse el conjunto de ejemplos planteado en clase, sino saber aplicar la técnica de diseño de DV; los alumnos deben desarrollar la habilidad de encontrar formas de descomponer un problema en subproblemas y –quizás más importante– deben saber determinar si la aplicación de DV es posible y adecuada.

La idea general de la técnica de diseño de DV es repasada en el punto 1. Se muestra que esta idea aparece no sólo en programación, sino también en muchos otros ámbitos. Introduciendo un esquema genérico no recursivo, se explica que los algoritmos de DV no deben ser necesariamente recursivos; es más, puede que sólo se aplique la descomposición en subproblemas a un nivel. En la práctica, todas las aplicaciones que se verán de DV son recursivas, así que se presenta otro esquema de DV, usando recursividad, con descomposición de un problema en dos subproblemas y datos almacenados en una tabla. Aunque este esquema sólo contempla un reducido conjunto de casos, tiene el interés de mostrar las funciones básicas de todo algoritmo de DV: *dividir*, *combinar*, *solución directa* y *tamaño base*. Se ven algunos ejemplos sencillos, que posiblemente son conocidos por los alumnos, como el problema de las torres de Hanoi y el cálculo de los números de Fibonacci.

Analizando el esquema general se deducen algunas características que debe satisfacer un problema para que se pueda aplicar DV. Por ejemplo, centrándose en la función *dividir*, se concluye que tiene que existir una forma de descomponer el problema en varios subproblemas disjuntos. Hay que reconocer, de todas formas, que normalmente la mejor manera de aprender a discriminar cuándo es posible o no aplicar DV es la propia experiencia. Por otro lado, se indica que en el caso de hacer una descomposición en un solo subproblema hablamos más propiamente de la técnica de reducción o simplificación, de la cual también se verá un ejemplo.

En el punto 2 se hace un estudio genérico de los tiempos de ejecución de los algoritmos de DV. Se usa el método del polinomio característico –explicado en el tema anterior– para analizar los casos de descomposición de un problema en dos subproblemas de la mitad de tamaño, y en a subproblemas de tamaño n/b y combinación en $O(n)$. Se llega a un conjunto de fórmulas generales, que serán aplicadas sobre los ejemplos posteriores.

El primer ejemplo de aplicación de DV, analizado en el punto 3, es el de la ordenación, dando como resultado las técnicas de ordenación por mezcla (o *mergesort*) y ordenación rápida (o *quicksort*). Ambas son estudiadas detalladamente en primer curso, por lo que el interés aquí es considerar su interpretación como algoritmos de DV y realizar un estudio de la complejidad más minucioso que el realizado en primero. En particular, se hace un estudio de tiempo y memoria para los dos algoritmos, distinguiendo en la ordenación rápida los casos peor, mejor y promedio. Para la ordenación por mezcla se hace un análisis teórico del tamaño óptimo del caso base, comprobando su influencia en el tiempo

[2]Llamamos aquí "solución directa" a un algoritmo que no use DV, si bien el algoritmo puede ser no trivial.

de ejecución.

En el punto 4 se aborda el problema de la multiplicación de enteros largos, es decir, enteros de tamaño no restringido. Este ejemplo permite comparar la eficiencia conseguida con una solución clásica y directa al problema, con un algoritmo sencillo de DV y con una aplicación no tan inmediata de DV. El punto empieza haciendo una introducción a la definición y uso de enteros de longitud arbitraria, para lo cual se requiere una estructura de representación mediante listas de dígitos. Sobre valores de este tipo se define la operación de multiplicación, resuelta mediante el procedimiento clásico de multiplicar todos los dígitos de un entero por los del otro.

Se plantea la aplicación de DV sobre este problema, que en su variante sencilla requiere 4 multiplicaciones, cada una de la mitad de tamaño del original; esto da lugar a un algoritmo con la misma complejidad que el método clásico. Entonces se propone la descomposición de Karatsuba y Ofman, que sólo necesita resolver de 3 subproblemas. Se analiza cómo aunque el orden de complejidad obtenido es menor, en la práctica sólo se logra mejora en el tiempo de ejecución para tamaños muy grandes. Para profundizar en esta cuestión, se hace un estudio teórico del tamaño óptimo del caso base.

Seguidamente, se plantea en el punto 5 la aplicación de DV a la multiplicación de matrices. Igual que antes, se empieza viendo las estructuras de datos que intervienen en el problema, se repasa y analiza el algoritmo clásico de multiplicación matricial, y se aplica DV usando una descomposición sencilla del problema. Esta descomposición resulta tener el mismo orden de complejidad que el método clásico, es decir $O(n^3)$. Se propone una descomposición un poco más compleja –debida a Strassen– pero que necesita un menor número de subproblemas, con lo que se logra reducir el orden de complejidad.

Con estos dos ejemplos se intenta hacer ver a los alumnos cómo a veces la solución más sencilla puede no ser la más eficiente y, a la vez, la solución más eficiente puede ser más costosa para tamaños pequeños. Por otro lado, el estudio del ejemplo de la multiplicación de matrices es un buen momento para presentar informalmente el concepto de cota inferior de un problema, como el menor orden de complejidad entre todos los algoritmos que resuelven ese problema.

Por último, en el punto 6 se estudia una aplicación de la técnica de reducción (o simplificación) al problema de selección. Se define el objetivo del problema –que es una generalización del problema de encontrar la mediana de un array – y se plantea el método directo para resolverlo, ordenando toda la tabla. Entonces se discute cómo usando el procedimiento *pivote*, de la ordenación rápida, es posible reducir el problema a un problema de menor tamaño; de esta forma se llega a un algoritmo de reducción, que además resulta no recursivo. Se analiza la complejidad conseguida, concluyendo que en el caso medio se obtiene una mejora significativa de la eficiencia.

Temporización

Tiempo estimado de clases teóricas: **4 horas**. La planificación es mostrada en la tabla 6.3.

El desarrollo de los puntos 1 y 3 puede ser relativamente rápido, teniendo en cuenta que son, en su mayor parte, un repaso de cuestiones vistas en primer curso. De esta forma, al final del tema debe quedar tiempo suficiente para plantear algunos ejercicios en los que sean los alumnos los que discutan la aplicación de DV.

Tema 8 - Algoritmos voraces

Objetivos

- Comprender los fundamentos de la técnica de diseño de algoritmos voraces o de avance rápido,

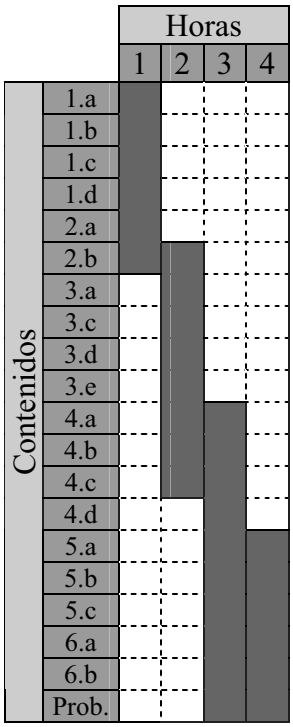


Tabla 6.3: Planificación temporal del Tema 7, Parte II.

identificando su estructura en varios de los algoritmos sobre grafos ya estudiados.

- Conocer una variedad de algoritmos voraces, adquiriendo la capacidad de aplicar avance rápido para problemas nuevos, determinando si su aplicación es posible y adecuada.
- Valorar la importancia de diseñar heurísticas de selección adecuadas, como un medio para obtener buenos algoritmos voraces.
- ◦ Demostrar formalmente la optimalidad de algunos algoritmos voraces, usando reducción al absurdo y análisis de casos.
- Conocer la utilidad de la técnica de avance rápido en la resolución no óptima o aproximada de problemas complejos, analizando el compromiso exactitud/complejidad que se plantea.

Contenidos

1. Método general de los algoritmos voraces.
   - 1.a) La idea del avance rápido.
   - 1.b) Esquema general de avance rápido. Datos y funciones usados.
   - 1.c) La importancia del criterio de selección.
   - 1.d) Ejemplos de algoritmos voraces conocidos: Dijkstra, Prim y Kruskal.
   - 1.e) Estudio del problema del cambio de monedas.
2. Análisis de tiempos de ejecución.
   - 2.a) Análisis general del peor caso. Generalidades.

2.b) El tiempo de algunos algoritmos conocidos.
3. Problema de la mochila.
   3.a) Definición y formulación matemática del problema.
   3.b) Aplicación de la técnica de avance rápido.
   3.c) Estudio de la función de selección. Demostración de optimalidad.
4. Problemas de planificación de tareas.
   4.a) Definición del problema de planificación con plazo fijo.
   4.b) Aplicación de la técnica de avance rápido.
   4.c) Estudio de la función de selección y de factibilidad.
   4.d) Demostración de optimalidad y orden de complejidad.
5. Problema del paseo del caballo.
   5.a) Definición del problema del paseo del caballo.
   5.b) Aplicación de la técnica de avance rápido. Función de selección.
   5.c) Posibilidades en la representación del recorrido.
6. Heurísticas voraces.
   6.a) El compromiso solución buena/ejecución rápida en problemas NP.
   6.b) Solución heurística para el problema del viajante.
   6.c) Solución heurística para la coloración de grafos.

Bibliografía básica

Capítulo 10 de [Giménez'03] y capítulo 6 de [Brassard'97].

Bibliografía complementaria

No son muchos los libros que dedican un capítulo exclusivamente a los algoritmos voraces, además de los mencionados en la bibliografía básica. Entre ellos podemos destacar el texto docente [Campos'95] en la lección 22, el capítulo 2 de [Gonzalo'98], el capítulo 16 de [Cormen'01], y el capítulo 4 de [Harel'92]. Un problema interesante en el que se puede profundizar es la creación de códigos de Huffman. En algunos libros los algoritmos voraces aparecen como apartados en capítulos dedicados a técnicas de diseño, como el apartado 10.3 de [Aho'88], el apartado 10.1 de [Weiss'95], y el apartado II.3 de [Troya'84],. Por otro lado, teniendo en cuenta que muchos de los algoritmos clásicos sobre grafos usan la técnica de avance rápido (como los algoritmos de Dijkstra, Prim y Kruskal), son numerosos los libros que incluyen ideas y ejemplos de avance rápido en capítulos dedicados a grafos, como [Aho'74], [Horowitz'82], [Baase'83], [Aho'88], [Brassard'90], [Cormen'90], [Brassard'97], [Baase'00]. No obstante, puesto que estos algoritmos ya fueron estudiados en el Tema 4, su interés en el presente tema es muy secundario.

Descripción de los contenidos

A lo largo de este tema se trata de dar una visión amplia y flexible de la técnica de avance rápido o de algoritmos voraces (en inglés *greedy*). Algunos algoritmos voraces han sido ya vistos en el Tema 4 de grafos; se retoman aquí estos algoritmos, para ver cómo todos ellos están basados en la idea de construir una solución por pasos, sin deshacer ninguna decisión ya tomada. La idea es conceptualmente muy simple y su comprensión no suele presentar dificultad por parte de los alumnos. Como en todos los temas del bloque de diseño de algoritmos, la explicación es realizada fundamentalmente en base a ejemplos de aplicación. Se intenta que estos ejemplos sean variados en cuanto a representación de la solución, restricciones del problema y tipo de función de selección. Aun así, es posible definir un esquema genérico de funcionamiento.

Siempre que se pueda demostrar fácilmente la optimalidad de alguno de los algoritmos presentados, se explicará en clase la demostración. El alumno debe tener en mente que un buen algoritmo voraz puede resolver ciertos problemas de forma óptima y eficiente, pero si no ha sido bien diseñado las soluciones obtenidas pueden ser malas. Por otro lado, debe conocer también la existencia de problemas para los cuales no hay algoritmos voraces óptimos. En esos casos, la heurística de selección juega un papel esencial para determinar la bondad del algoritmo.

El punto 1 empieza introduciendo la idea del avance rápido desde un punto de vista genérico. Se da la visión de un algoritmo voraz como un proceso por pasos, en el que en cada uno se toma un elemento de entre un conjunto de candidatos. Esto da pie a introducir un esquema general, donde se manejan los conjuntos de candidatos, de seleccionados y de elementos añadidos a la solución. Las funciones que aparecen en este esquema serán los aspectos a determinar para cada problema particular a resolver: función de selección, de factibilidad, de añadir un elemento a la solución parcial y comprobar si tenemos una solución. Se incide en la importancia del criterio de selección, como una medida heurística para discriminar entre los elementos del conjunto de candidatos.

A continuación, se hace un repaso de algunos algoritmos sobre grafos conocidos, viendo su interpretación como algoritmos voraces. En el caso de los algoritmos de Prim y Kruskal, se comprueba que pueden existir distintas posibilidades para elegir cuál es el conjunto de candidatos (en este caso los vértices o las aristas, respectivamente). En el algoritmo de Dijkstra, se ve que al añadir un elemento a la solución puede ser necesario hacer otros cálculos adicionales.

En el subapartado 1e) se desarrolla un ejemplo sencillo: el problema del cambio de moneda. Se muestra la aplicación paso a paso de la técnica de avance rápido, discutiendo con los alumnos la función de selección más adecuada. Se llega hasta un algoritmo en pseudocódigo y se explica que –dependiendo del sistema monetario– puede garantizar solución óptima o no.

Igual que en los restantes temas de diseño de algoritmos, en el punto 2 se tratan algunas generalidades en cuanto a los tiempos de ejecución de los algoritmos voraces. En este caso, el análisis genérico es poco práctico, ya que requiere realizar muchas suposiciones sobre el tamaño del conjunto de candidatos, de una solución final, el orden de complejidad de las funciones básicas, etc. Se repasan los tiempos de algunos algoritmos ya conocidos, concluyendo que normalmente obtendremos órdenes de complejidad polinomiales.

Los puntos 3, 4 y 5 incluyen tres problemas para los cuales los algoritmos voraces consiguen siempre soluciones óptimas. Otros dos problemas, para los cuales no se puede garantizar la solución óptima usando avance rápido, son estudiados en el punto 6.

El problema de la mochila[3] es tratado en primer lugar, en el punto 3. Se define el objetivo, describiéndolo formalmente como un problema de optimización con satisfacción de restricciones. Entonces, se va discutiendo y desarrollando el algoritmo, aplicando las ideas vistas en el punto 1. En particular, el interés está en la elección de la función de selección y en la demostración –por reducción al absurdo– de su optimalidad. Normalmente, los alumnos ven de forma intuitiva que el criterio de máximo beneficio por unidad de peso garantizará una solución óptima. Se comprueba que en el caso donde los objetos no se puedan partir el algoritmo no garantiza solución óptima.

La siguiente aplicación tratada es la planificación de tareas con plazo fijo. El principal interés de este problema es el estudio de la función de factibilidad, debida a la restricción de plazo máximo impuesta sobre las tareas. Además, a diferencia del ejemplo anterior, el orden de los elementos incluidos en la solución es un factor relevante. Se enuncia y se demuestra un lema, que afirma que esta ordenación debe ser por orden creciente de plazo máximo de las tareas. Una vez estudiadas todas las funciones, se muestra el esquema del algoritmo, se realiza algún ejemplo y se estudia el orden de complejidad. Es

[3]Entiéndase mochila no 0/1, es decir, el caso donde se pueden partir los objetos. En otro caso hablamos de mochila 0/1 o binaria.

posible, también, realizar la demostración de optimalidad del algoritmo obtenido, siempre que quepa en la temporización del tema.

En el punto 5 se aplica avance rápido sobre el problema del paseo del caballo. Se define el objetivo, mostrando cómo puede ser transformado en un problema de ciclo hamiltoniano sobre un grafo de posiciones del tablero. Después se analiza la aplicación de avance rápido, estudiando la definición de una función de selección adecuada; no se demuestra la optimalidad de este criterio de selección. Para acabar, se consideran las distintas posibilidades para representar el recorrido, como las representaciones mediante pares $(fila, columna)$ o mediante movimientos, estudiando su influencia en la generación de los candidatos.

Por último, en el punto 6 se tratan dos problemas NP-completos sobre grafos, para los cuales las soluciones voraces son aproximaciones que pueden resultar más o menos buenas. Se motiva la necesidad de diseñar algoritmos rápidos para resolver problemas computacionalmente costosos, aun a costa de no encontrar la solución óptima –o incluso de no encontrar ninguna solución– y se analiza el compromiso que ello implica.

El primer problema estudiado es el del viajante (o agente de comercio), cuya definición ya fue vista en el tema de grafos. Se plantean dos posibilidades en cuanto a cuál es el conjunto de candidatos: o bien los candidatos son los vértices del grafo o bien las aristas. Cada una de estas posibilidades da lugar a un algoritmo voraz distinto, surgiendo posibles heurísticas de selección de manera más o menos inmediata. En cualquier caso, se comprueba que no garantizan la solución óptima.

El segundo ejemplo tratado en el punto 6 es la coloración de grafos. Se plantea el algoritmo voraz como un proceso por fases, donde en cada fase se selecciona un color distinto y se asigna a todos los vértices posibles que estén sin colorear. En este caso, la función de selección toma los elementos en un orden aleatorio. Igual que antes, se muestra con un ejemplo que el algoritmo no es óptimo.

Normalmente, la explicación de estos dos últimos ejemplos sólo llega a nivel de describir textualmente el funcionamiento de los algoritmos. Se supone que los alumnos son capaces de trasladar las ideas a esquemas en pseudocódigo y, en última instancia, de llegar a programarlos. Así pues, puede resultar un buen ejercicio proponer la escritura de los algoritmos en pseudocódigo.

Temporización

Tiempo estimado de clases teóricas: **4 horas**. La planificación es mostrada en la tabla 6.4.

La mayor parte de la carga lectiva de este tema se encuentra en el estudio de los ejemplos de aplicación. Cada uno de ellos requiere, como mínimo, alrededor de 40 minutos, incluyendo la realización de algunos ejercicios. Aunque es posible que sobre algo de tiempo para hacer más problemas, según la planificación no es previsible que sea más de media hora de clase.

Tema 9 - Programación dinámica

Objetivos

- Comprender los fundamentos de la técnica de diseño de algoritmos basada en programación dinámica y el principio de optimalidad de Bellman en el que se sustenta.
- Relacionar las técnicas de divide y vencerás y programación dinámica, entendiéndolas como diferentes estrategias para aplicar la misma idea de descomponer un problema en subproblemas.
- Conocer una variedad de algoritmos de programación dinámica, adquiriendo la capacidad de usar la técnica en problemas nuevos, determinando si su aplicación es posible y adecuada.

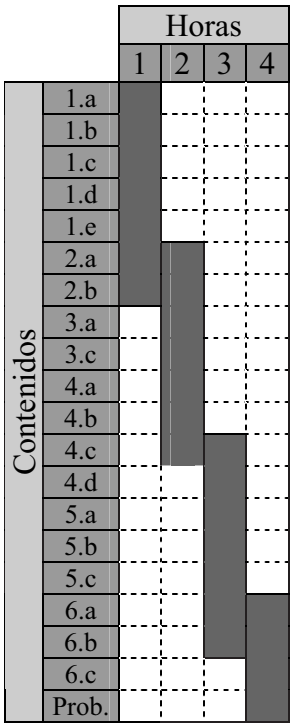


Tabla 6.4: Planificación temporal del Tema 8, Parte II.

- Demostrar la optimalidad de algunos algoritmos de programación dinámica, comprobando que se cumple el principio de optimalidad de Bellman.

- Reconocer y analizar los factores de tiempo de ejecución y memoria en la resolución de un problema usando programación dinámica, planteando diferentes alternativas de descomposición y representación.

Contenidos

1. Método general de programación dinámica.
   - 1.a) La idea de la técnica de programación dinámica.
   - 1.b) Métodos de descomposición ascendentes y descendentes.
   - 1.c) Ejemplos de algoritmos conocidos: números de Fibonacci y Floyd.
   - 1.d) El principio de optimalidad de Bellman.
   - 1.e) Pasos para aplicar la técnica de programación dinámica.
2. Análisis de tiempos de ejecución.
   - 2.a) Factores a considerar: tamaño de la tabla y tiempo en cada elemento.
   - 2.b) Importancia del factor memoria.
3. Problema de la mochila 0/1.
   - 3.a) Diseño de la solución con programación dinámica.
   - 3.b) Reconstrucción de la solución a partir de las tablas.
   - 3.c) Orden de complejidad y principio de optimalidad.
   - 3.d) Otras posibilidades de representación de la tabla.

4. Problema del cambio de monedas.
    4.a) Diseño de la solución con programación dinámica.
    4.b) Reconstrucción de la solución a partir de las tablas.
    4.c) Orden de complejidad y principio de optimalidad.
    4.d) Otras posibles definiciones recurrentes.
5. Multiplicación encadenada de matrices.
    5.a) Definición y objetivo del problema.
    5.b) Definición de la ecuación recurrente y los casos base.
    5.c) Algoritmo de programación dinámica y reconstrucción de la solución.
    5.d) Orden de complejidad.

Bibliografía básica

Capítulo 11 de [Giménez'03] y capítulo 8 de [Brassard'97].

Bibliografía complementaria

Para profundizar en aspectos de la programación dinámica no estudiados en este tema se pueden usar algunas referencias que incluyen capítulos enteros dedicados a esta técnica. Entre estas referencias podemos destacar el capítulo 15 de [Cormen'01], el capítulo 9 de [Rabhi'99], el capítulo 10 de [Baase'00], el capítulo II.4 de [Troya'84] y el capítulo 5 de [Brassard'90]. En otros libros la técnica es estudiada junto con otras técnicas de diseño, de manera más o menos detallada, como en el apartado 10.3 de [Weiss'95], el apartado 10.2 de [Aho'88], y el apartado 7.5 de [Heileman'98]. Los dos aspectos más interesantes para profundizar en este tema son la utilización de representaciones alternativas a las tablas y el estudio de otros ejemplos de aplicación, especialmente en problemas que no sean de optimización. Por ejemplo, se puede estudiar el problema de construir árboles binarios de búsqueda óptimos o varios problemas relacionados con la búsqueda de subcadenas.

Descripción de los contenidos

Este tema presenta la técnica general de diseño de algoritmos de programación dinámica. Aunque ya se estudiaron en el Tema 4, de grafos, los algoritmos de Floyd y Warshall –basados en programación dinámica–, es difícil que los alumnos asocien los anteriores algoritmos con un esquema de funcionamiento claramente definido. La idea de la programación dinámica es presentada como una estrategia ascendente para aplicar la descomposición de un problema en subproblemas. Por lo tanto, es relacionada con la técnica de divide y vencerás y, en cierto sentido, se pueden considerar como los extremos opuestos en la forma de aplicar una misma definición recursiva.

Encontrar buenos ejemplos para explicar programación dinámica no es sencillo; no suele ser fácil diseñar descomposiciones adecuadas para todos los problemas, que se puedan implementar mediante el uso de tablas. Pero el mayor inconveniente es que la descomposición aplicada para un problema puede aportar poca información sobre cómo hacerlo para otros problemas, siendo en muchos casos difícil de encontrar una fórmula adecuada.

En cada uno de los ejemplos desarrollados se va viendo paso a paso la aplicación de la técnica de programación dinámica, hasta llegar al diseño de un algoritmo en pseudocódigo y al análisis de complejidad. El principio de optimalidad de Bellman es introducido como una forma de asegurar el funcionamiento óptimo de los algoritmos de programación dinámica, y es aplicado sobre los ejemplos desarrollados.

En el punto 1 se empieza introduciendo la idea general de la programación dinámica, relacionándola con divide y vencerás: ambas se basan en resolver un problema combinando las soluciones

de subproblemas de menor tamaño. Se muestra la idea de usar una tabla para almacenar los resultados de los subproblemas. Esta estrategia, en contraste con divide y vencerás, da lugar a la distinción entre dos estrategias contrapuestas en la aplicación de la descomposición recurrente: descendente (desde el problema original hacia los subproblemas) y ascendente (desde los casos base hacia problemas de mayor tamaño). Con el ejemplo sencillo del cálculo de los números de Fibonacci –para el cual ambas posibilidades resultan en algoritmos muy simples–, los alumnos pueden apreciar claramente la diferencia entre ambas estrategias. Además, este problema sirve para comprobar cómo la programación dinámica puede evitar en muchos casos la realización de cálculos repetidos. Otro ejemplo analizado es el algoritmo de Floyd, visto en el tema de grafos e interpretado ahora como un método de descomposición ascendente.

A continuación, se presenta el principio de optimalidad de Bellman, y es aplicado sobre el problema de los caminos mínimos en un grafo. Con esto se justifica el funcionamiento óptimo del algoritmo de Floyd, previamente repasado. Para comprobar que no todos los problemas cumplen este principio se analiza el contraejemplo de los caminos simples más largos, donde una subsecuencia de un camino simple máximo no es necesariamente un camino máximo. Es más, en muchos casos la aplicación del principio de optimalidad no depende del problema sino de la descomposición que se haga.

Resumiendo el método de programación dinámica, se muestran los pasos necesarios para aplicarlo a un problema: definir una ecuación recurrente con los casos base, definir las tablas y la forma de rellenarlas, y establecer la forma de obtener la solución a través de las tablas. Es adecuado que los alumnos conozcan y utilicen un proceso metódico de este tipo, de manera que el diseño de algoritmos usando programación dinámica no se convierta en un problema de idea feliz.

En el punto 2 se hacen algunas consideraciones sobre la eficiencia y el uso de memoria de los algoritmos de programación dinámica. Se hace hincapié en el factor memoria, que puede convertirse fácilmente en un recurso crítico debido a la necesidad de usar tablas de gran tamaño. En cuanto al tiempo de ejecución, se puede considerar como la suma de los tiempos requeridos para cada elemento de la tabla. Se muestra que, en el caso de que estos tiempos sean iguales, se podrá obtener el orden de complejidad multiplicando el tamaño de la tabla por el tiempo en cada elemento.

A partir del punto 3 se empiezan a estudiar los ejemplos de aplicación, comenzando por el problema de la mochila 0/1. Se repasa la definición del problema de la mochila, introduciendo ahora la restricción de no poder partir los objetos. Se van aplicando sucesivamente los pasos del proceso propuesto; la primera cuestión a considerar –y quizá la que plantea mayores dificultades para los alumnos– es determinar en función de qué parámetros está dado el tamaño del problema: un parámetro es el número de objetos a usar y el otro –no tan evidente– es el tamaño de la mochila.

Lo siguiente es definir una ecuación recurrente, con sus casos base. Estudiando el problema $Mochila(i, j)$, consistente en resolver el problema de la mochila 0/1 con los i primeros objetos y capacidad de la mochila j, se plantea la decisión "usar o no usar el objeto i-ésimo". Esto da lugar a dos subproblemas, que son combinados con una operación máximo. Después se discuten los casos base, las tablas utilizadas para almacenar los subproblemas y la forma de rellenarlas, llegando a un algoritmo en pseudocódigo.

Tras aplicar el algoritmo sobre un ejemplo, se ve la necesidad de recomponer la solución a partir de las tablas, y se estudia el modo de hacerlo. Luego se analiza la complejidad del algoritmo diseñado, que se obtiene de manera casi directa. Se compara con el orden de complejidad del algoritmo voraz, pero recordando que este no resuelve el problema de forma óptima. Sin embargo, puesto que se puede aplicar el principio de optimalidad, el algoritmo de programación dinámica sí es capaz de hacerlo. Por último, se ven otras posibilidades en la representación de la tabla, que evitan el desperdicio de memoria.

El siguiente ejemplo, estudiado en el punto 4, es el problema del cambio de monedas. La aplicación de la técnica presenta varias similitudes con el caso anterior; igual que para la mochila 0/1, se ha estudiado ya un algoritmo voraz pero que no siempre garantiza una solución óptima. En este caso, el tamaño del problema viene determinado por los tipos de monedas a utilizar y la cantidad a devolver. La decisión a adoptar es "usar al menos una moneda de tipo i o no", y es combinada con un operador de mínimo, ya que estamos en un problema de minimización. Se definen las tablas, el algoritmo para rellenarlas y la forma de recomponer la solución a partir de las mismas. Después se estudia el orden de complejidad –que es comparado con el del algoritmo voraz– y se comprueba que se verifica el principio de optimalidad. Se proponen también otras posibles definiciones recurrentes del problema, viendo que pueden haber diversas maneras de aplicar la técnica.

Para acabar, en el punto 5 se aplica programación dinámica sobre el problema de la multiplicación encadenada de matrices. Se analiza un ejemplo de multiplicación que implica varias matrices de distintos tamaños, donde el orden en el que se realicen las operaciones influye de manera importante en el número de productos escalares necesarios. Así pues, el objetivo es encontrar el orden que requiera el menor número de productos escalares.

Aplicando la técnica, se establece como tamaño del problema el par de valores (i, j), que indican: resolver la multiplicación entre la matriz i-ésima y la j-ésima. La decisión a tomar para este problema será "por dónde colocar los paréntesis en primer lugar, entre i y j". De esta manera aparecen varios subproblemas, de los cuales se toma el mínimo. La ecuación resultante tiene una forma particular, que requiere un orden de rellenado de la tabla por diagonales. Además, al analizar el tiempo de ejecución hay que tener en cuenta que el tiempo no es el mismo para todos los elementos de la tabla. Para finalizar, se da un algoritmo para realizar la multiplicación de las matrices en el orden óptimo.

Aunque este último ejemplo es más complejo que los anteriores, tiene algunas características que hacen interesante su estudio, como el orden de rellenado de las tablas o la forma de la ecuación recurrente. Como ya se ha comentado, adquirir habilidad en buscar descomposiciones recurrentes de un problema requiere cierta experiencia en un variado tipo de casos. Por lo tanto, resulta adecuado ver otros ejemplos en clases de problemas o facilitarlos a los alumnos en los boletines de ejercicios.

Temporización

Tiempo estimado de clases teóricas: **4 horas**. La planificación es mostrada en la tabla 6.5.

Igual que en el tema anterior, la mayor parte de la carga lectiva se encuentra en la aplicación de la programación dinámica sobre los problemas de ejemplo, en los puntos 3, 4 y 5. En este caso no se incluyen horas de clases de problemas, ya que los ejercicios son intercalados con la explicación de los ejemplos.

Tema 10 - Backtracking

Objetivos

- Comprender los fundamentos de la técnica de diseño de algoritmos basada en backtracking, razonando sobre la idea subyacente de recorrido implícito en árbol de soluciones.
- Ser consciente de la gran flexibilidad que ofrece backtracking en cuanto a distintos tipos de árboles, problemas a resolver y esquemas (recursivos o no recursivos) para su implementación.
- Identificar distintos tipos de problemas que se pueden resolver por backtracking, adquiriendo la capacidad de aplicarlo en problemas nuevos, determinando si su aplicación es posible y adecuada.

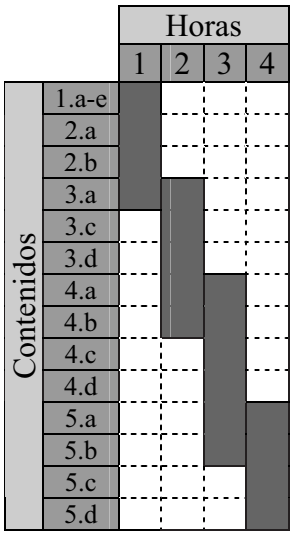


Tabla 6.5: Planificación temporal del Tema 9, Parte II.

- Saber analizar el orden de complejidad de los algoritmos de backtracking, siendo capaz, al menos, de dar cotas superiores del tiempo de ejecución en el peor caso.
- Tomar conciencia de la necesidad de usar diferentes estrategias para reducir la complejidad computacional implícita en los algoritmos de backtracking.

Contenidos

1. Método general de backtracking.
   - 1.a) La idea de la técnica de backtracking.
   - 1.b) Representación de la solución y árboles implícitos.
   - 1.c) Esquema general no recursivo. Datos y funciones usados.
   - 1.d) Distintos casos del esquema general.
2. Análisis de tiempos de ejecución.
   - 2.a) Factores a considerar: número de nodos y tiempo en cada nodo.
   - 2.b) Cálculo del número de nodos en algunos tipos de árboles.
   - 2.c) Reducción del número de nodos.
3. Problema de la mochila 0/1.
   - 3.a) Diseño de la solución con backtracking, representación binaria.
   - 3.b) Algoritmo y funciones del esquema.
   - 3.c) Orden de complejidad en el peor caso.
   - 3.d) Mejora del algoritmo con estimación de beneficio máximo.
   - 3.e) Otras mejoras: orden de generación y árbol combinatorio.
4. Problema de la asignación.
   - 4.a) Definición y objetivos del problema de la asignación.
   - 4.b) Representación de la solución con árboles permutacionales.
   - 4.c) Algoritmo y funciones del esquema.
   - 4.d) Orden de complejidad del algoritmo.

4.e) Mejoras del algoritmo básico.

5. Problema de las N reinas.
   5.a) Soluciones de fuerza bruta y solución con backtracking.
   5.b) Representación de la solución y funciones del esquema.
   5.c) Algoritmo para obtener una o todas las soluciones.
   5.d) Orden de complejidad y evaluación probabilística del tiempo.

Bibliografía básica

Capítulo 12 de [Giménez'03] y apartado 9.6 de [Brassard'97].

Bibliografía complementaria

Existen diversas formas en la bibliografía de plantear la técnica de backtracking. En muchas referencias se considera el backtracking como una técnica recursiva, por lo que se estudia en capítulos dedicados al diseño recursivo, como el capítulo 3 de [Wirth'80], el capítulo 9 de [Collado'87], el capítulo 3 de [Wirth'87], y el capítulo 7 de [Kruse'89]. En varios libros se estudia junto con otras técnicas de recorrido de árboles y grafos, como en el apartado 6.6 de [Brassard'90], la lección 21 de [Campos'95], y en el apartado 9.6 de [Brassard'97]. Por último, algunos libros incluyen el backtracking dentro de un capítulo dedicado a técnicas de diseño de algoritmos, como ocurre en el apartado 10.4 de [Aho'88], el apartado 10.5 de [Weiss'95], y el apartado 8.2 de [Rabhi'99]. Los ejemplos de aplicación que aparecen con más frecuencia son el problema de la mochila y el de las 8 reinas. El enfoque que se utiliza en este tema está más cercano a [Troya'84], donde se estudia el problema de las reinas, se diseñan esquemas y se presenta la idea del estudio estadístico.

Descripción de los contenidos

La técnica de diseño de algoritmos de backtracking[4] es conocida por los alumnos, que la estudiaron en primer curso aunque superficialmente, como un simple ejemplo de aplicación de la recursividad. Este tema supone, por lo tanto, una profundización desde el punto de vista del diseño de algoritmos, centrado en extraer las ideas de la técnica general de backtracking y su aplicación en resolución de problemas. El conocimiento de la técnica por parte de los alumnos es una cierta ventaja en la explicación del tema, aunque hay que tener en cuenta que sus conocimientos están ligados al reducido conjunto de ejemplos que estudiaron. Se pretende que los estudiantes conozcan la gran flexibilidad que aporta backtracking, viendo una variedad de tipos de problemas y de representaciones de la solución usados.

En contraste con el esquema recursivo de backtracking estudiado en primer curso, el esquema propuesto en este tema es no recursivo. Los alumnos deben conocer ambas posibilidades y ser capaces de establecer relaciones entre ellas, considerando diferencias, similitudes, ventajas e inconvenientes. En cualquier caso, debe quedar claro que lo importante es saber usar backtracking, independientemente del tipo de esquema que se aplique.

En el punto 1 se presenta la idea de la técnica de backtracking, como un proceso de búsqueda exhaustiva y sistemática en un árbol de soluciones. Se introducen, pues, las ideas de árbol y espacio de soluciones, ligadas estrechamente a la representación de la solución mediante una tupla. Se hace un repaso del funcionamiento genérico de backtracking, desarrollando la explicación sobre un ejemplo sencillo: encontrar un subconjunto dentro de un conjunto dado de enteros, que sume una cierta cantidad. Para este problema se estudian dos posibles tipos de árboles –un árbol binario y un árbol combinatorio– y se va discutiendo el funcionamiento del algoritmo.

[4] También conocida, aunque menos usualmente, como retroceso o vuelta atrás.

Una vez familiarizados con el funcionamiento de backtracking, se propone un esquema genérico no recursivo; se hace hincapié en que este esquema está basado en una serie de datos y funciones, cuyo significado debe ser establecido para cada problema particular. En concreto, se discute en clase cómo deben ser implementadas las funciones básicas para el ejemplo anterior, que fue tratado de manera más o menos informal. En el desarrollo de los ejemplos se intenta seguir un proceso sistemático de diseño, compuesto por los pasos de: elegir una estructura de representación de la solución (y por lo tanto la forma del árbol implícito), razonar sobre la forma de generar el árbol y finalmente implementar las funciones que llevan a cabo el recorrido.

El esquema genérico presentado considera el caso en que se busca una sola solución al problema y se conoce que al menos existe una. Se discuten en clase las adaptaciones que deberían hacerse para los casos de buscar todas las soluciones, de que pueda no haber solución y de tener un problema de optimización. Los alumnos deberían ser capaces de resolver por sí mismos estas cuestiones, sin excesiva dificultad.

En el punto 2 se hacen algunas consideraciones sobre los tiempos de ejecución de los algoritmos de backtracking, en términos generales. Se asume que el tiempo total es la acumulación del tiempo requerido para cada nodo del árbol implícito; así pues, aparecen los factores: número de nodos y tiempo en cada nodo. Se estudia el número de nodos en algunos ejemplos típicos de árboles, como los árboles binarios, los n-arios, los árboles combinatorios y los permutacionales. La conclusión es que los algoritmos obtenidos son de órdenes de complejidad exponencial o factorial, lo cual motiva la necesidad de buscar formas de reducir el número de nodos estudiados.

A continuación, se estudian algunos ejemplos de aplicación de backtracking. En el punto 3 se vuelve sobre el problema de la mochila 0/1, que ya debe resultar familiar para los alumnos. Para este ejemplo –igual que para el siguiente– se desarrolla primero un algoritmo más o menos sencillo, sobre el que van proponiéndose después modificaciones y mejoras. La aplicación de backtracking sigue el proceso antes comentado. Primero se discute la representación de la solución, lo cual determina la forma del árbol de soluciones, que en este caso será un árbol binario. Después se establece el esquema del algoritmo, haciendo las modificaciones necesarias sobre el esquema genérico. Por último se estudian e implementan las funciones básicas, obteniendo así una especificación completa del algoritmo.

Se muestran algunos ejemplos de ejecución y se analiza el orden de complejidad, dando lugar a un $O(2^n)$, lo que motiva la necesidad de reducir el número de nodos. Se propone añadir una poda basada en el criterio de optimización: si a partir de una solución parcial no se puede mejorar la mejor solución actual, entonces se puede podar ese nodo. Esto implica hacer estimaciones de beneficio máximo a partir de un nodo; se muestra cómo hacerlas usando el algoritmo voraz para el problema de la mochila no 0/1. Se discute el resultado obtenido: la modificación reduce el número de nodos pero aumenta el tiempo necesario en cada nodo. También se proponen otras modificaciones, aunque de manera más superficial, sin llegar a la implementación. Por ejemplo, se plantea cambiar el orden de generación de los nodos (generando primero el valor 1 y luego el 0), ordenar los objetos según el criterio de beneficio por unidad de peso o usar una representación combinatoria de la solución. Se supone que los alumnos son capaces de llevar estas ideas, vistas muy por encima, a nivel de implementación.

En el punto 4 se estudia el problema de la asignación. Se introduce y se da una definición formal del problema, motivando el interés de resolverlo debido a la gran variedad de aplicaciones en las que aparece. El desarrollo de la explicación es similar al expuesto para el punto 3. Se estudia primero la representación de la solución, que en este caso da lugar a un árbol permutacional. Se muestra después el esquema del algoritmo y las funciones básicas, llegando a una solución sencilla donde el criterio de poda sólo incluye la restricción de no asignar más de una vez un objeto. Se ve la ejecución de un ejemplo y se analiza el orden de complejidad en el peor caso, resultando un término factorial. Para reducir este tiempo se estudian algunas mejoras, como incorporar un criterio de poda más restrictivo,

basado en la función objetivo a maximizar, o cambiar el orden en que son generados los descendientes a partir de un nodo.

El último ejemplo de aplicación es el problema de las N reinas, estudiado en el punto 5. A diferencia de los anteriores, no se trata de un problema de optimización sino que se buscan todas las soluciones que satisfagan un conjunto de restricciones. Se plantea la definición del problema, mostrando el elevado número de posibilidades que se requeriría para resolverlo con un simple algoritmo de fuerza bruta. A continuación, se desarrolla la aplicación de backtracking al problema, tomando decisiones sobre la representación de la solución, las modificaciones a realizar sobre el esquema genérico y la forma de implementar las funciones del esquema. Se muestra un ejemplo de ejecución y se hace el análisis de complejidad. En este caso, se ve que la cota superior obtenida del análisis en el peor caso está muy alejada de la complejidad real del problema. El cálculo del orden exacto es difícil, por lo que se propone como alternativa un estudio probabilístico del tiempo de ejecución, del cual se dan algunas ideas. Básicamente, este estudio consiste en generar varias permutaciones de forma aleatoria, obtener el nivel del árbol al que se llegaría con cada una y promediar el número de nodos máximo para cada nivel con la probabilidad estimada de llegar a ese nivel.

Temporización

Tiempo estimado de clases teóricas: **4 horas**. La planificación es mostrada en la tabla 6.6.

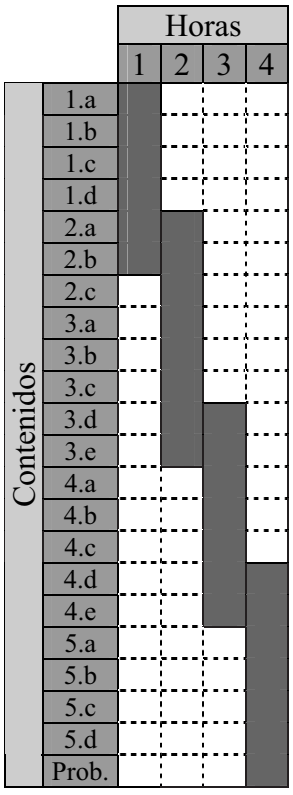


Tabla 6.6: Planificación temporal del Tema 10, Parte II.

En la temporización de este tema hay que tener en cuenta que los alumnos ya están familiarizados con la técnica de backtracking, aunque no se debe suponer un dominio completo de la misma. Por lo

tanto, cuatro horas deben ser suficientes para repasar y profundizar en la técnica, y aplicarla a los ejemplos propuestos. Es posible que quede algo de tiempo para plantear ejercicios, en los cuales dar indicaciones sobre la aplicación de backtracking sobre algunos problemas nuevos.

Tema 11 - Ramificación y poda

Objetivos

- Comprender los fundamentos de la técnica de diseño de algoritmos de ramificación y poda, entendiéndola como una generalización de la idea de recorrido implícito en árbol de soluciones.
- Distinguir y relacionar las técnicas de backtracking y ramificación y poda, analizando cuando puede ser preferible una u otra para un problema dado.
- Ser capaz de aplicar ramificación y poda en problemas nuevos, comprendiendo la importancia de la heurística en la resolución eficiente de problemas de alto coste computacional.
- Entender las implicaciones del compromiso que se plantea en el cálculo de cotas para una solución parcial: cálculo preciso pero lento o rápido pero inexacto.
- Saber diseñar y utilizar de métodos aproximados (como, por ejemplo, heurísticas voraces) como ayuda en la resolución de problemas por ramificación y poda.

◦ Mostrar las ideas de ramificación y poda que se pueden aplicar en la resolución de determinados problemas donde su aplicación completa no es posible o adecuada.

Contenidos

1. Método general de ramificación y poda.
   - 1.a) La ramificación y poda como extensión y mejora de backtracking.
   - 1.b) Cotas de una solución parcial: inferior, superior y valor estimado.
   - 1.c) Estrategias de poda y de ramificación. La lista de nodos vivos.
   - 1.d) Esquema general de ramificación y poda.
2. Análisis de tiempos de ejecución.
   - 2.a) Factores a considerar: número de nodos y tiempo en nodo.
   - 2.b) El compromiso en la estimación de cotas, rápida o precisa.
3. Problema de la mochila 0/1.
   - 3.a) Representación binaria y combinatoria de la solución.
   - 3.b) Estimación rápida de las cotas y usando algoritmos voraces.
   - 3.c) Estrategias de poda y de ramificación adecuadas.
   - 3.d) Algoritmo en pseudocódigo y funciones del esquema.
   - 3.e) Orden de complejidad en el peor caso.
4. Problema de asignación.
   - 4.a) Representación de la solución mediante una permutación.
   - 4.b) Estimación rápida de las cotas y usando algoritmos voraces.
   - 4.c) Estrategias de poda y de ramificación adecuadas.
   - 4.d) Algoritmo en pseudocódigo y funciones del esquema.

4.e) Orden de complejidad en el peor caso.
5. Problema de las N reinas.
5.a) Representación de la solución mediante una permutación.
5.b) Estimación de beneficio como medida heurística de tablero prometedor.
5.c) Modificaciones del esquema genérico.

Bibliografía básica

Capítulo 13 de [Giménez'03] y apartado 9.7 de [Brassard'97].

Bibliografía complementaria

En general, son muchos menos los libros donde se trata la ramificación y poda que el resto de las técnicas estudiadas en los temas anteriores. Por ejemplo, en [Brassard'90] se presenta el método de forma muy breve en el apartado 6.8; en la siguiente edición del libro, [Brassard'97], se incluyen dos de los ejemplos que se proponen para este tema, el problema de la asignación y el de la mochila. En el capítulo II.4 de [Troya'84] se estudia esta técnica y se ilustra con el problema del secuenciamiento de trabajos con plazos. También se puede usar el libro recientemente publicado [Martí'04], centrado en la explicación de ejercicios resueltos, aunque incluyendo unas breves consideraciones teóricas previas; la ramificación y poda es tratada en el capítulo 15.

Descripción de los contenidos

Este es el último tema del bloque de diseño de algoritmos que trata de una técnica general de diseño, la ramificación y poda. La técnica es presentada como una generalización de la idea de recorrido implícito en un árbol de soluciones, introducida en el tema de backtracking. A nivel conceptual, el funcionamiento del método suele ser fácil de asimilar por parte de los alumnos; pero la aplicación a problemas concretos presenta más dificultades, ya que requiere tomar buenas decisiones en un elevado número de aspectos (representación de la solución, cálculo de las cotas, criterios de ramificación y de poda, etc.).

Los alumnos deben ser capaces de saber cómo llegar desde la especificación de un problema, pasando por la definición de un espacio de soluciones posibles, y acabando en la implementación del algoritmo dentro un programa. Por otro lado, deben concienciarse de la necesidad de utilizar buenas heurísticas para el cálculo de las cotas y el valor esperado para una solución parcial.

Para conseguir estos objetivos –igual que en los temas precedentes– se empieza dando una visión de la técnica desde una perspectiva general, para luego aplicarla a una serie de problemas variados y ejemplificantes. Los tres problemas incluidos son los mismos que los tratados en el tema de backtracking. De este modo se facilita la explicación, ya que los alumnos están familiarizados con los problemas y el interés se centra en la aplicación de la técnica. Además, es posible comparar y establecer relaciones entre las distintas soluciones para un mismo ejemplo usando distintas técnicas.

En el punto 1 se presenta la idea general de ramificación y poda. Los alumnos acaban de ver la técnica de backtracking en el tema anterior, por lo que resulta natural establecer una relación entre ambas técnicas: las dos están basadas en el recorrido implícito de un árbol de soluciones. Entonces se destacan las características particulares de la ramificación y poda: la estrategia de ramificación y la estrategia de poda. Se explican los conceptos de cota superior, cota inferior y beneficio (o coste) estimado para una solución parcial. La idea de cota superior en un problema de maximización apareció ya en el tema de backtracking, aunque no como una parte de la técnica general.

A través de algunos ejemplos sencillos de árboles de problemas, se explican las estrategias de ramificación y de poda más frecuentes, viendo su efecto sobre el recorrido de los árboles. En particular,

se tratan las estrategias de ramificación FIFO, LIFO, LC-FIFO y LC-LIFO, y los criterios de poda usados cuando a partir de una solución parcial siempre hay alguna solución final válida y cuando puede no haberla. Se introduce la lista de nodos vivos y se estudia la forma en la que se usa esta estructura para realizar los recorridos.

Una vez visto el funcionamiento de la técnica, se propone un esquema general de algoritmo de ramificación y poda. Igual que en todos los anteriores esquemas algorítmicos, se hace uso de estructuras de datos y funciones básicas, cuyo significado debe ser estudiado y definido para cada problema a resolver. El esquema considera un problema de minimización; la adaptación a un problema de maximización no debe presentar dificultad para los alumnos.

A continuación, en el punto 2, se hacen algunos comentarios relativos a los tiempos de ejecución de los algoritmos de ramificación y poda, de forma genérica. Se ve cómo aparecen los mismos dos factores que en backtracking: número de nodos del árbol y tiempo en cada nodo. Se plantea una comparación con backtracking, en el sentido en que puede necesitar recorrer menos nodos, pero a costa de aumentar el tiempo en cada uno de ellos. Es más, esto da pie a plantear el compromiso que aparece en el diseño de las cotas para la poda: estimar las cotas de forma precisa pero lenta, o rápida pero inexacta. Se hacen también algunos comentarios respecto al uso de memoria, fundamentalmente el requerido para almacenar la lista de nodos vivos.

Igual que con las otras técnicas de diseño de algoritmos, se intenta que los alumnos aprendan y sigan un orden metódico en la aplicación de la técnica. En este caso, el proceso empieza por establecer una forma de representar la solución, y por lo tanto el tipo de árbol implícito. A partir de ahí, se determina cómo obtener los descendientes a partir de una solución parcial, y cómo estimar las cotas y el beneficio (o coste) estimado. Después se diseñan las estrategias de ramificación y de poda adecuadas; y finalmente se concreta el esquema genérico, definiendo las funciones y los datos usados.

En los tres problemas planteados en los puntos 3, 4 y 5 se intenta ejemplificar la aplicación del proceso, hasta llegar a la obtención de los algoritmos correspondientes. Luego se hace un estudio del tiempo de ejecución –sin entrar en mucho detalle–, y se muestra la ejecución de algún ejemplo.

En el punto 3 se desarrolla la aplicación de ramificación y poda sobre el problema de la mochila 0/1. Se proponen dos formas de representar la solución, las mismas que las planteadas con backtracking: mediante árboles binarios y combinatorios. En cuanto al cálculo de las cotas, se analizan también varias posibilidades. La forma más sencilla, pero imprecisa, es suponer que la solución parcial se completa añadiendo todos los objetos restantes o ninguno, lo cual se puede calcular en un tiempo constante. Esto se contrasta con una estimación más precisa, usando los algoritmos voraces adecuados, cuyo resultado sea aplicable como cota inferior o superior del problema. Para el beneficio estimado se usa simplemente la media de las dos cotas. La estrategia de ramificación es una LC, es decir, explorar primero por los nodos con mayor beneficio estimado. Se presenta una posible implementación en pseudocódigo usando la representación binaria, y se muestra un ejemplo de ejecución. Se expone también la ejecución del ejemplo con la representación combinatoria, de la cual se ve el modo de operar, aunque sin llegar a la implementación.

Seguidamente, se trata el problema de la asignación en el punto 4. Nuevamente, se aplican los pasos del proceso definido. La representación de la solución para este problema da lugar a una estructura implícita de árbol permutacional. En relación al cálculo de las cotas, vuelve a aparecer una forma sencilla –asignar los máximos o los mínimos a los elementos pendientes en la solución parcial– y una forma más precisa –definir un algoritmo voraz adecuado– aunque más lenta. En este caso, el problema no fue tratado en el tema de algoritmos voraces, por lo que la solución de avance rápido debe ser discutida en clase con el suficiente detenimiento.

Por último, en el punto 5 se aplica ramificación y poda sobre el problema de las N reinas, suponiendo que buscamos una solución cualquiera. La aplicación de la técnica en este ejemplo resulta

un tanto particular ya que, al no tratarse de un problema de optimización, no tiene sentido definir cotas superiores o inferiores. La estimación del beneficio se define como una medida heurística de lo prometedor que resulta un tablero; en consecuencia, la ramificación y poda permite guiar el recorrido, explorando primero las ramas más prometedoras.

Se analizan varias posibles heurísticas, viendo las dificultades que plantea encontrar una buena medida de lo prometedora que es una solución parcial. Algunas de estas heurísticas permiten, además, definir un criterio de poda específico del problema. Por ejemplo, el número de casillas no amenazadas en el tablero es usado como una medida heurística; si este valor es menor que el número de reinas por colocar, entonces no se puede encontrar ninguna solución a partir de esa situación.

Se plantea a los alumnos la cuestión de qué ocurriría si queremos encontrar todas las soluciones al problema de las N reinas. Se ve que en ese caso la estrategia de ramificación –y, por lo tanto, el cálculo de la heurística– carece de importancia ya que el árbol se debe recorrer de forma completa.

Temporización

Tiempo estimado de clases teóricas: **5 horas**. La planificación es mostrada en la tabla 6.7.

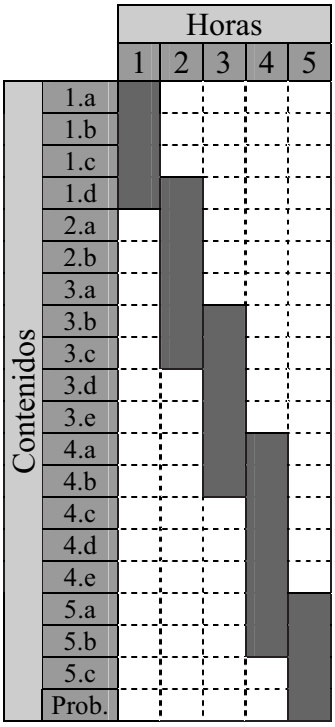


Tabla 6.7: Planificación temporal del Tema 11, Parte II.

Para este tema se ha estimado un total de cinco horas de clase, una más que para los restantes temas de diseño de algoritmos. Esta diferencia es debida a la mayor complejidad implícita de la técnica de ramificación y poda. En su mayor parte, las horas se utilizarán en la explicación de los ejemplos. Se espera que quede tiempo para realizar algunos ejercicios, al final del tema.

Tema 12 - Resolución de juegos

Objetivos

- Comprender los fundamentos de las técnicas de resolución de juegos, entendiendo la representación de los estados de un juego mediante árboles de juego.
- Conocer el proceso de recorrido del árbol de juego y propagación de valores definidos por la estrategia minimax, sabiendo justificar su modo de operar y comprendiendo el significado del resultado obtenido.
- Entender la poda alfa-beta como una forma de reducir el número de nodos explorados, eliminando movimientos no óptimos, y que permite explorar una mayor parte del árbol.
- Adquirir la capacidad de usar las técnicas de resolución de juegos en problemas de juegos nuevos, ya sean finitos o no finitos.
- Ser consciente de la importancia de diseñar buenas funciones de utilidad, especialmente para juegos no finitos o de tamaño grande.

Contenidos

1. Árboles de juegos.
   - 1.a) Características del tipo de juegos considerados.
   - 1.b) Representación mediante árboles de juegos, finitos y no finitos.
   - 1.c) La función de utilidad. Asignación heurística de valores.
2. Estrategia minimax.
   - 2.a) Propagación de valores usando la estrategia minimax.
   - 2.b) Ejemplo: el problema de las tres en raya.
   - 2.c) Implementación de minimax usando backtracking recursivo.
   - 2.d) Aplicación en árboles no finitos.
3. Poda alfa-beta.
   - 3.a) Ramificación por los nodos más prometedores.
   - 3.b) Estudio y justificación de la poda alfa y la poda beta.
   - 3.c) Modificación del algoritmo minimax para incluir poda alfa-beta.
4. Juego del Nim.
   - 4.a) Definición del juego del Nim.
   - 4.b) Función de utilidad y generación de movimientos.
   - 4.c) Algoritmo usando minimax y poda alfa-beta.

Bibliografía básica

Capítulo 14 de [Giménez'03] y apartado 10.4 de [Aho'88].

Bibliografía complementaria

Aunque muchos libros de AED tratan las técnicas de resolución de juegos, pocos de ellos incluyen un capítulo dedicado exclusivamente a los juegos. En algunos casos la técnica es considerada como un ejemplo de backtracking, como en el apartado 10.4 de [Aho'88], la lección 21 de [Campos'95], y el

apartado 10.5 de [Weiss'95]. Otras veces aparece dentro de técnicas de recorrido de árboles, como en [Horowitz'82], [Tenenbaum'88], [Kruse'89], [Langsam'97], o bien de grafos, como en el capítulo 6 de [Brassard'90] y el 9 de [Brassard'97]. En su mayor parte, los esquemas algorítmicos utilizados coinciden con los de [Aho'88].

Descripción de los contenidos

En este tema se hace una breve introducción a los problemas de juegos y las técnicas usadas para diseñar algoritmos que los resuelvan. Se consideran exclusivamente juegos "de tablero", donde participan dos contrincantes y en los cuales no interviene el azar. Se desarrolla la idea de representar el espacio de posibles situaciones del juego mediante los llamados árboles de juego. En este sentido, se establece una clara relación con las técnicas de recorrido implícito en árbol, estudiadas en los dos temas anteriores.

La explicación es realizada fundamentalmente en base a ejemplos de juegos sencillos, como las tres en raya o el juego del Nim. Sobre estos ejemplos se van aplicando los conceptos introducidos, de manera que el significado del árbol de juego, la estrategia minimax, la función de utilidad y la poda alfa-beta son asimilados por los alumnos de forma más fácil. Aparte de comprender y ser capaz de aplicar los anteriores conceptos, es importante dejar claro cuál es el objetivo final del problema de juegos: decidir el siguiente movimiento para aproximarse a la victoria. Paradójicamente, y especialmente en este tema, muchas veces los problemas de aprendizaje de los alumnos se deben a no tener claro qué es lo que se pretende conseguir.

La idea de los árboles de juego es introducida en el punto 1. En primer lugar se motiva el interés del tema, mostrando algunos ejemplos de juegos conocidos como las tres en raya, el ajedrez o las damas. Analizándolos, se ve que todos ellos presentan una serie de características comunes: participan dos jugadores de manera alternativa, existe un número finito de posibilidades, no interviene el azar, etc.

Entonces, se ve la necesidad de representar las posibles situaciones del juego y su evolución en la partida, surgiendo el concepto de árbol de juego. El árbol es descrito en base a sus componentes: los nodos son situaciones del juego y los descendientes de un nodo son los posibles movimientos de cada jugador. Se muestran algunos ejemplos sencillos de árboles en juegos de tamaño finito y no finito. Sobre estos árboles se explica intuitivamente el concepto de función de utilidad, y la necesidad de usar medidas heurísticas de la utilidad en el caso de árboles de tamaño no finito.

Después de ver la representación del problema, se trata su resolución algorítmica en el punto 2. Planteando el objetivo final del juego –encontrar un movimiento que conduzca a una situación ganadora–, se analiza cómo plasmar ese objetivo en el árbol de juego. Se discute cómo un jugador intentará moverse dentro del árbol hacia hojas con valor alto de la función de utilidad, mientras que el otro moverá hacia hojas con valor bajo; así, surge de manera natural la estrategia minimax, para realizar la propagación de valores de utilidad desde las hojas del árbol hasta la raíz, pasando por los nodos internos. Se vuelve sobre los ejemplos del punto anterior, para mostrar sobre ellos la aplicación de la estrategia minimax.

A continuación, se propone una implementación recursiva de la estrategia minimax. Se muestra un esquema genérico de algoritmo, donde la función de utilidad y las funciones para generar los descendientes a partir de un nodo están sin especificar: son los aspectos a definir para cada juego concreto. Se hacen algunas consideraciones sobre la aplicación de minimax en árboles no finitos y que, por lo tanto, no pueden ser expandidos de forma completa.

Tomando como base el algoritmo de minimax, se proponen algunas mejoras en el punto 3. La primera consiste en aplicar la idea –vista en el tema de ramificación y poda– de ramificar primero por las situaciones más prometedoras. Para ello, puede resultar interesante usar la función de utilidad,

aplicada sobre situaciones no terminales, como criterio de nodo prometedor. La segunda extensión es la conocida como poda alfa-beta. A través de dos ejemplos, se analizan ciertas situaciones donde se puede garantizar que a partir de un nodo no se encontrará la solución óptima. En consecuencia, esos nodos del árbol pueden ser podados sin perder la solución del problema. Se muestran las modificaciones a realizar sobre el algoritmo minimax, presentado en el punto 2, para incluir la poda alfa-beta.

La comprensión de la estrategia minimax y la poda alfa-beta no suele ser problemática para los alumnos, ya que al verlas mediante ejemplos asimilan intuitivamente su significado. Sin embargo, la aplicación de estas técnicas sobre un problema nuevo, y la implementación en un algoritmo, suelen presentar mayores dificultades. Por este motivo, se incluye en el último punto el desarrollo completo de un ejemplo de juego, desde la definición hasta la implementación. Este ejemplo es el juego del Nim, o juego de los palillos. Se dispone de varios montones, en cada uno de los cuales hay un número prefijado de palillos. En cada movimiento un jugador debe retirar uno o más palillos, pero siempre dentro del mismo montón. Pierde el jugador que quita el último palillo.

El juego encaja perfectamente en el tipo de juegos estudiados y resulta de resolución más o menos simple. La primera cuestión a tratar es cómo representar cada situación del juego; la opción más sencilla es usar una tabla de enteros, con tantos elementos como número de montones. Una vez con esto, se discute cómo definir la función de utilidad y cómo hacer la generación de movimientos a partir de un nodo. Se aplica el esquema genérico, para obtener un algoritmo completamente especificado. Por último, se muestran algunos ejemplos de ejecución y se hace una estimación del orden de complejidad del algoritmo.

Puede ser interesante plantear algunas cuestiones adicionales a los alumnos, sobre la resolución de juegos de forma genérica, de las cuales se podrían dar algunas sugerencias. Por ejemplo, ¿qué pasa si hay más de dos jugadores?, ¿que ocurre con los juegos donde influye el azar? ó ¿y si el número de posibles movimientos es no finito? No obstante, por los motivos comentados, conviene no extender excesivamente la explicación de este tema.

Temporización

Tiempo estimado de clases teóricas: **2 horas**. La planificación es mostrada en la tabla 6.8.

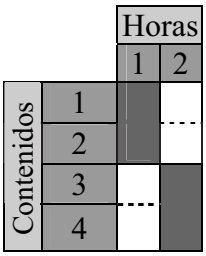


Tabla 6.8: Planificación temporal del Tema 12, Parte II.

Como ya se ha comentado, este tema se considera de carácter no básico, por lo que en caso necesario es posible suprimirlo del temario. Otra posibilidad, en caso de escasez de tiempo, es dejar el ejemplo del punto 3 como un ejercicio a resolver en clase de problemas. No obstante, el tema entra dentro de la planificación, con un tiempo asignado de dos horas de clase, y debería ser explicado normalmente.

Capítulo 7

Programa de prácticas

"Si el código tiene un error que no se produce siempre, ignóralo y sigue escribiendo. [...] *Va a ser difícil de encontrar, puede que en la demostración de la práctica no salga, y además puede que desaparezca solo si no lo miras".*

Agustín Cernuda, Cómo NO hacer unas prácticas de programación

La propuesta de prácticas de la asignatura AED fue descrita en detalle dentro del Capítulo 4 (Selección y organización de contenidos), en el apartado 4.4.3. En este Capítulo profundizaremos en la descripción de los seminarios de prácticas. Recordemos que los tres seminarios propuestos están dedicados a la enseñanza de: C, C++ y especificaciones formales en Maude. Cada seminario está formado por sesiones de dos horas, que son realizadas en los laboratorios con los ordenadores delante. Para cada seminario se describen los objetivos, las sesiones que contiene y la bibliografía recomendada. Por otra parte, para cada sesión se establecen los objetivos educativos específicos, los contenidos a desarrollar y un breve comentario sobre las consideraciones más importantes de la sesión.

7.1. Seminario de C

Objetivos

- Aprender a programar con soltura en el lenguaje de programación C, conociendo los mecanismos básicos y avanzados del lenguaje.
- Conocer y saber manejar las herramientas de Linux para la edición y compilación de código C.
- Conocer los mecanismos de C que permiten la programación modular, sabiendo afrontar la programación de proyectos de tamaño grande a través de su descomposición en librerías.

Sesiones

Sesión 1. Introducción, sintaxis y compilación de código C

Sesión 2. Tipos estructurados y punteros
Sesión 3. Funciones estándar y memoria dinámica
Sesión 4. Programación modular en C

Bibliografía

- Guión de prácticas del seminario
- Apéndice A de [García'03]
- [Kernighan'91]
- [Weiss'93]

Temporización

Tiempo estimado de clases prácticas: **8 horas** (4 sesiones a 2 horas por sesión).

Sesión 1 - Introducción, sintaxis y compilación de código C

Objetivos

- Conocer las características más importantes del lenguaje C y su sintaxis básica.
- Conocer los tipos de datos básicos disponibles en C, sabiendo cuándo usar uno u otro.
- Conocer los operadores y sentencias de control de flujo de C.
- Saber editar, compilar y ejecutar un programa sencillo en C, bajo Linux.

Contenidos

1. Características generales de C
   - 1.a) Sistema de tipos, tipos básicos y declaración de variables
   - 1.b) Funciones y procedimientos, el tipo `void`
   - 1.c) Sentencias de control del flujo
   - 1.d) Entrada/salida básica en C
2. Edición y compilación de un programa C
   - 2.a) Edición del código en Linux
   - 2.b) Compilación usando `gcc`
3. Variables y tipos de datos básicos
   - 3.a) Listado de tipos numéricos básicos de C
   - 3.b) Variantes de los tipos básicos, modificadores `long`, `short`, `signed` y `unsigned`
   - 3.c) Uso de valores booleanos en C
   - 3.d) Definición de tipos enumerados
   - 3.e) El operador `sizeof`
4. Operadores

4.a) Listado de operadores básicos de C
4.b) Precedencia y asociatividad de los operadores
4.c) Conversión de tipos: implícita y explícita
5. Sentencias de control de flujo
5.a) Sentencia `goto` y la programación espagueti
5.b) Sentencias `if ... else ...` y `switch`
5.c) Sentencias `while`, `do ... while` y `for`
5.d) Interrupción de iteraciones con `break` y `continue`
Ejercicios

Comentarios

En esta primera sesión del seminario de C se introducen las características más elementales del lenguaje, que permitirán la escritura y compilación de los primeros programas en C. Se parte del hecho de que los alumnos ya saben programar en algún otro lenguaje como Pascal o Modula-2. Por este motivo la explicación puede realizarse a un ritmo más bien rápido. Es interesante saber exactamente cuál es el lenguaje que conocen los alumnos, para poder contrastar las características de C con las del lenguaje que usaron en primero, no sólo en esta sesión sino también en las restantes. Se destaca la cualidad de C como un lenguaje de relativo bajo nivel, con un sistema de tipos débil. En la sesión se debe dejar tiempo suficiente para que los alumnos empiecen a escribir y probar sus primeros programas, en los que con toda seguridad empezarán a surgir las primeras dudas.

Sesión 2 - Tipos estructurados y punteros

Objetivos

- Conocer los mecanismos de C para la creación de tipos estructurados: registros, arrays, enumerados y uniones.
- Conocer la declaración y uso de punteros en C, incluyendo las operaciones que permiten la aritmética de punteros.
- Valorar las ventajas e inconvenientes de la flexibilidad de C en el manejo de punteros.
- Conocer la sintaxis usada en C para la declaración de funciones.
- Comprender la idea de puntero a función, entendiendo sus posibles aplicaciones.

Contenidos

1. Definición y uso de punteros
1.a) Declaración de punteros en C
1.b) Operadores de dirección (`*`) e indirección (`&`)
1.c) Punteros `void *` y el puntero nulo `NULL`
1.d) Reglas de compatibilidad en la asignación
1.e) Aritmética de punteros, punteros dobles, triples, etc.
2. Arrays

2.a) Definición e inicialización de arrays
2.b) Representación de cadenas con arrays de caracteres
2.c) Arrays n-dimensionales
2.d) Arrays y punteros
3. Estructuras (registros) y uniones
3.a) Declaración y uso de registros (`struct`), la notación punto
3.b) Punteros a registros, la notación flecha (`->`)
3.c) Inicialización de registros
3.d) Declaración y uso de uniones (`union`)
3.e) Definición de tipos nuevos con `typedef`
4. Funciones y procedimientos
4.a) Sintaxis para la definición de funciones, parámetros y valor devuelto
4.b) Variables locales
4.c) La función `main()`, parámetros posibles
4.d) Paso de funciones como parámetros
Ejercicios

Comentarios

En esta sesión se siguen presentando algunos de los conceptos más básicos de la programación en C, como la declaración y uso de funciones, registros y arrays. Se trata un tema muy delicado, como es el uso de punteros, aunque en esta sesión referencian a variables reservadas de forma estática. Es necesario advertir sobre la complejidad y los riesgos potenciales de la aritmética de punteros, que será normalmente una de la grandes dificultades en la programación en C. La sesión también aborda aspectos no triviales, como los punteros a funciones y el paso de funciones como parámetro.

Sesión 3 - Funciones estándar y memoria dinámica

Objetivos

- Conocer y saber usar correctamente las funciones estándar de C para el manejo de la entrada/salida, desde pantalla/teclado y desde ficheros.
- Conocer los mecanismos que ofrece C para la gestión de memoria dinámica.
- Conocer algunas de las principales librerías estándar de C.
- Saber dónde se puede acudir para encontrar información adicional sobre las funciones estándar de librería de C.

Contenidos

1. Funciones de entrada/salida estándar
1.a) La función `printf`
1.b) La función `scanf`
1.c) Variantes de las funciones `printf` y `scanf`
1.d) Otras funciones para la entrada/salida estándar

1.e) Los ficheros `stdin`, `stdout` y `stderr`
2. Manejo de ficheros
   2.a) La librería `stdio.h` y el tipo `FILE`
   2.b) Modos de acceso a un fichero
   2.c) Operaciones con ficheros
3. Gestión dinámica de memoria
   3.a) La librería `stdlib.h`
   3.b) Las funciones `malloc()`, `calloc()` y `realloc()`
   3.c) La función `free()`
4. Otras funciones de interés
   4.a) Funciones con cadenas
   4.b) Funciones con bloques de memoria
   4.c) Otras funciones y lugares de referencia

Ejercicios

Comentarios

Los aspectos tratados en esta sesión hacen referencia a funciones predefinidas que pueden resultar de utilidad, más que a características propias del lenguaje C. Las funciones estudiadas están orientadas al manejo de entrada/salida con ficheros, teclado y pantalla, y a la gestión de memoria dinámica. Por su interés, creemos conveniente que se presenten dentro del mismo seminario del lenguaje. Los alumnos deben tener el tiempo suficiente para practicar el uso de todas las funciones. Obviamente, no todas las funciones de las librerías estándar de C pueden ser incluidas. En consecuencia, es necesario proveer a los alumnos con referencias suficientes sobre dónde buscar más información sobre las librerías de C. Por ejemplo, se puede comentar el manual de Linux, `man`, como un método para obtener ayuda. Esta herramienta es estudiada con más profundidad en las prácticas de la asignatura "Sistemas Operativos", que se desarrollan normalmente de forma simultánea.

Sesión 4 - Programación modular en C

Objetivos

- Conocer y comprender los mecanismos que ofrece C para la programación modular.
- Entender las fases del proceso de compilación en C: preprocesamiento, compilación y enlace.
- Saber usar el programa `make`, para automatizar el proceso de compilación.
- Valorar la importancia de aplicar buenas prácticas de programación, en particular en los programas en C.

Contenidos

1. El preprocesador de C
   1.a) Características y utilidad del preprocesador
   1.b) El comando `#define`, con y sin parámetros
   1.c) El comando `#include`

1.d) Compilación condicional con los comandos `#ifdef`, `#ifndef`, `#else` y `#endif`
2. Programación modular en C
2.a) Ficheros de cabecera y ficheros de implementación
2.b) Variables externas (`extern`)
2.c) Buenas prácticas en la descomposición modular
2.d) Compilación
3. El programa `make`
3.a) El proceso de compilación en C, código fuente, objeto y ejecutable
3.b) Compilación en programas con muchos ficheros
3.c) Objetivo y características del programa `make`
3.d) Uso del programa `make`, definición de ficheros `makefile`
Ejercicios

Comentarios

Esta sesión está centrada en la utilización de C en grandes proyectos, en los que pueden aparecer muchos módulos, el código es muy largo y pueden intervenir distintos programadores. Los mecanismos que ofrece C a este respecto no son características propias del lenguaje, sino más bien del preprocesador. Por ejemplo, la inclusión de librerías externas no se hace con una construcción del lenguaje, sino con la directiva de preprocesador "`#include`". Puesto que los proyectos realizados por los alumnos serán normalmente de cierto tamaño, es necesario que conozcan estos mecanismos, y que sean conscientes de la importancia de crear buenos diseños modulares. Aunque no existe el concepto de módulo en C, es posible usar las directivas que ofrece el preprocesador para separar un programa en distintas partes, y para distinguir, a su vez, entre especificación e implementación de cada parte. También se incluye el uso de la herramienta `make`, para automatizar el proceso de compilación en proyectos grandes. Este programa será también estudiado en las prácticas de "Sistemas Operativos", por lo que no insistiremos excesivamente.

7.2. Seminario de C++

Objetivos

- Aprender a programar en el lenguaje C++ a nivel básico, conociendo los conceptos y mecanismos más importantes del lenguaje.
- Relacionar y saber diferenciar los lenguajes C y C++, conociendo las opciones de C++ que no están disponibles en C.
- Conocer y saber manejar las herramientas de Linux para la edición y compilación de código C++.
- Comprender las bases y los conceptos fundamentales de la programación orientada a objetos, entendiendo el significado de clases y objetos y sabiendo aplicarlos correctamente en el diseño de programas. Se excluyen explícitamente todos los conceptos que surgen de la herencia.
- Conocer los mecanismos que ofrece C++ para la declaración y definición de clases –incluyendo las clases parametrizadas–, y creación, uso y eliminación de objetos.

Sesiones

Sesión 1. El lenguaje C++ como una extensión de C
Sesión 2. Clases y objetos en C++
Sesión 3. Parametrización y excepciones

Bibliografía

- Guión de prácticas del seminario
- Apéndice B de [García'03]
- [Stroustrup'98]
- [Deitel'03]
- [Joyanes'00]

Temporización

Tiempo estimado de clases prácticas: **6 horas** (3 sesiones a 2 horas por sesión).

Sesión 1 - El lenguaje C++ como una extensión de C

Objetivos

- Conocer la relación entre C y C++, entendiendo el segundo como una extensión del primero.
- Entender la posición de C++ en el mundo de los lenguaje de programación de alto nivel, como un híbrido entre los lenguajes imperativos y los orientados a objetos.
- Conocer las modificaciones y novedades, no relacionadas con la programación OO, que añade C++ respecto de C.

Contenidos

1. Características generales de C++
   - 1.a) C++ como superconjunto de C
   - 1.b) C++ como híbrido entre imperativo y OO puro
   - 1.c) Clases y registros
   - 1.d) Modificaciones menores de C++
2. Entrada/salida estándar
   - 2.a) La librería `iostream`
   - 2.b) Flujos de entrada y salida estándar, `cin`, `cout` y `cerr`
3. Variables y tipos de datos
   - 3.a) El tipo `bool`
   - 3.b) Declaración de variables en cualquier punto del código

3.c) Declaración de constantes
3.d) Diferencias en *casting* y definición de tipos
3.e) Espacios de nombres y el operador de resolución de visibilidad "`::`"
4. Funciones
4.a) Paso de parámetros por referencia
4.b) Parámetros por defecto
4.c) Sobrecarga de funciones
5. Memoria dinámica
5.a) Operadores `new` y `new[]`
5.b) Operadores `delete` y `delete[]`
Ejercicios

Comentarios

Justo al acabar el aprendizaje de C, o a las pocas semanas, empieza la explicación de C++ en esta sesión. Es necesario evitar la confusión entre ambos lenguajes y dejar claras las características de C++ no disponibles en C, ya que los alumnos tendrán que utilizar C en otras asignaturas. Esta sesión está centrada en las modificaciones y extensiones de C++ no relacionadas directamente con la POO. En su mayor parte son mejoras interesantes –como la posibilidad de utilizar paso de parámetros por referencia, o los nuevos mecanismos de gestión de memoria dinámica–, por lo que proponemos la presentación de un gran número de ellas. No obstante, teniendo en cuenta la trascendencia de la filosofía OO, al principio de la sesión, cuando se introducen las características básicas de C++, se hace una breve exposición de las clases, comparándolas con los registros. De esta manera, se ve la posibilidad de usar C++ definiendo clases o sin ellas.

Sesión 2 - Clases y objetos en C++

Objetivos

- Aprender los conceptos básicos de la programación orientada a objetos, entendiendo las clases como un mecanismo adecuado de definición de tipos abstractos de datos.
- Valorar las ventajas de utilizar clases en la definición de tipos, frente a la programación modular.
- Conocer la sintaxis de C++ para la declaración y definición de clases, excluyendo los conceptos que surgen de la herencia.
- Conocer la forma de crear objetos dinámicamente en C++, de usarlos y de destruirlos.
- Saber escribir programas sencillos en C++ que hagan uso de clases y objetos.

Contenidos

1. Las bases de la programación OO
1.a) El modelo de ejecución OO
1.b) La naturaleza dual de las clases: un clase como un módulo y como un TAD
1.c) Los objetos como instancias de una clase

1.d) Nomenclatura: miembros públicos y privados, métodos, mensajes y objeto receptor
1.e) Sintaxis para la declaración de clases en C++
2. Implementación de los métodos de una clase
2.a) Implementación fuera de la definición de la clase
2.b) Implementación dentro de la definición, métodos `inline`
3. Constructores y destructores
3.a) El concepto de constructor y destructor de una clase
3.b) Declaración e implementación de constructores en C++
3.c) Sobrecarga de constructores y el constructor por defecto
3.d) Declaración e implementación de destructores en C++
4. Objetos dinámicos
4.a) Creación dinámica de objetos mediante `new` y `new[]`
4.b) Eliminación de objetos mediante `delete` y `delete[]`
4.c) Uso de objetos dinámicos, el operador flecha "`->`"
5. Algunas notas
5.a) Proyectos con varios ficheros en C++
5.b) Importancia de respetar el ocultamiento de la representación
5.c) Clases amigas
Ejercicios

Comentarios

Sin duda alguna, por los conceptos introducidos aquí, esta sesión es una de las más importantes de los seminarios impartidos. En ella se presentan los conceptos fundamentales y básicos de la programación OO, así como la forma de implementarlos en C++. Son muchas las ideas nuevas que son tratadas, así que es esencial preparar una explicación cuidada y bien organizada. Realmente ya se comentó en clases de teoría, al explicar el concepto teórico de TAD, la idea de las clases como un mecanismo para definir TAD. Pero, evidentemente, ahora vamos más allá y tratamos la forma de hacerlo en C++. Los alumnos deben tener una visión general de la filosofía OO, omitiendo lo relativo a la herencia, como ya hemos justificado. Creemos que esta explicación reducida de la OO es posible, puede tener beneficios para el desarrollo de las prácticas de la asignatura, y es una buena base de partida para la asignatura "Programación Orientada a Objetos" de tercer curso. La propuesta es que los alumnos practiquen un enfoque mixto de programación, donde los tipos de datos más relevantes son definidos mediante clases, pero al mismo tiempo pueden haber trozos de código que utilicen una descomposición funcional más tradicional. Es necesario introducir muchos ejemplos de código en esta sesión, algunos de los cuales podrían estar relacionados con las prácticas a realizar.

Sesión 3 - Parametrización y excepciones

Objetivos

- Comprender el significado y utilidad de los conceptos de función y tipo genérico o parametrizado, excepción y aserto.
- Conocer los mecanismos de C++ para la definición de funciones y tipos genéricos, y saber aplicarlos en la definición de nuevas abstracciones genéricas.

- Conocer los mecanismos de C++ para el manejo de excepciones y asertos, entendiendo los segundos como una utilidad para definir pre- y postcondiciones en los procedimientos.
- Conocer otras características avanzadas de C++, como la posibilidad de redefinir los operadores.

Contenidos

1. Funciones y clases genéricas
   - 1.a) La idea de la genericidad
   - 1.b) Las estructuras de plantilla (`template`) en C++
   - 1.c) Funciones genéricas en C++
   - 1.d) Clases genéricas en C++, definición, instanciación y uso
2. Excepciones
   - 2.a) El concepto de excepción: significado, lanzamiento, propagación y manejo
   - 2.b) Lanzamiento de una excepción en C++
   - 2.c) Captura y manejo de una excepción en C++
3. Asertos
   - 3.a) El concepto de aserto: significado, uso como pre- y postcondición
   - 3.b) Definición de asertos en C++ con la función `assert()`
   - 3.c) Otras formas de incluir pre- y postcondiciones
4. El puntero `this`
   - 4.a) Significado del puntero `this`
   - 4.b) Uso del puntero `this`
5. Redefinición de operadores
   - 5.a) Sintaxis para la redefinición de operadores
   - 5.b) Ejemplo de redefinición de operadores

Ejemplo con genericidad, asertos y redefinición de operadores

Comentarios

En esta sesión se explican algunos conceptos avanzados de C++ que, aunque no son imprescindibles, son muy interesantes y se aconseja su utilización en las prácticas de la asignatura. Muchos de estos mecanismos implican la introducción de ideas que son nuevas para los alumnos, como las excepciones y los asertos. En estos casos es necesario explicar en primer lugar el concepto teórico, independientemente de su implementación en C++. En el caso de la genericidad, los alumnos deben conocer el concepto aunque exclusivamente de forma teórica. La presentación de los mecanismos de C++ para usar genericidad es muy importante, ya que será difícil que los alumnos manejen otro lenguaje que la permita, al menos con los planes de estudios y programas actuales. Igual que en la sesión anterior, se deben introducir muchos fragmentos de código que ejemplifiquen los conceptos introducidos.

7.3. Seminario de especificaciones formales

Esta práctica ofrece la posibilidad de poner a prueba los conceptos que se estudiaron de forma teórica en el Tema 1 (Abstracciones y Especificaciones). Tradicionalmente, en nuestra Facultad, las especificaciones formales algebraicas han sido estudiadas en la asignatura AED, pero sin llegar al punto de ponerlas en funcionamiento. De esta manera, las ideas introducidas quedaban algo lejos de cualquier utilidad real. Con este seminario se intenta suplir esta deficiencia, existiendo un desarrollo teórico y una posterior aplicación práctica. Para la realización de la práctica se ha elegido **Maude**,

que es uno de los lenguajes más activos en este área y uno de los más utilizados en prácticas similares de otras asignaturas.

Recordemos que este seminario ha sido propuesto como optativo, debido a la necesidad de racionalizar la carga de trabajo de la asignatura. No obstante, la posibilidad de convalidarlo por un ejercicio del examen (el correspondiente al Tema 1) lo hace muy atractivo para los alumnos, a la vez que se compensaría con la reducción de trabajo en la preparación del examen. La superación de esta práctica requiere la asistencia al seminario y la posterior entrega de algunos ejercicios en Maude, entre dos y tres semanas después del seminario. Se pueden ver ejemplos del tipo de ejercicios planteados en el apéndice C.3.

Aunque lo hemos descrito en tercer lugar, también sería posible su introducción entre los seminarios de C y C++, dejando de este forma algo de tiempo para la asimilación y práctica del lenguaje C. De hecho, en la planificación propuesta en el apartado 4.4.4 aparece de esa manera.

Objetivos

- Aplicar los conocimientos estudiados en clases de teoría en relación a la construcción de especificaciones formales algebraicas o axiomáticas.
- Saber escribir especificaciones formales algebraicas en el lenguaje Maude, y saber utilizarlas para *reducir* expresiones de ejemplo, interpretando los resultados obtenidos.
- Especificar formalmente en Maude algunos TAD básicos, como listas, pilas, colas y árboles binarios.
- Adquirir habilidad en la aplicación del razonamiento inductivo, en el cuál se basa la construcción de especificaciones algebraicas, a través de la especificaciones de nuevos TAD y operaciones.
- Entender la diferencia entre especificación e implementación, comprendiendo que en Maude razonamos a nivel de especificación, a pesar de lo cual es posible obtener prototipos ejecutables de los TAD especificados.

Bibliografía

- Guión de prácticas del seminario
- Capítulo 2 de [García'03]
- M. Clavel, et al. *A Maude Tutorial*. Disponible en: `http://maude.csl.sri.com`

Temporización

Tiempo estimado de clases prácticas: **2 horas** (1 sesión).

Contenidos

1. Descripción general de Maude
   - 1.a) Maude como una herramienta de especificaciones formales
   - 1.b) Nomenclatura de Maude y partes de la especificación
   - 1.c) Descarga, instalación y ejecución

2. Comandos básicos
    2.a) Lectura de un fichero, comando `in`
    2.b) Reducción de expresiones, comando `red`
    2.c) Modo de uso de Maude
3. Formato de especificación
    3.a) Estructura de un módulo
    3.b) Definición de la sintaxis de las operaciones
    3.c) Definición de la semántica de las operaciones
    3.d) Expresiones condicionales
    3.e) Activación y desactivación de la traza
4. Ejemplos
    4.a) TAD natural
    4.b) TAD letra
    4.c) TAD pila
    4.d) Ejemplos de uso

Ejercicios

Comentarios

Este seminario está muy centrado en el manejo de la herramienta Maude para la definición de especificaciones formales algebraicas. Hay que tener en cuenta que todo el desarrollo teórico del método ha sido realizado en las clases de la asignatura, por lo que el objetivo aquí es simplemente traducir el formato visto en clase al utilizado por Maude. Ambos tienen una notación muy parecida y consideramos que con una sesión de dos horas resulta más que suficiente. Además, proponemos hacer un uso *minimalista* de Maude, introduciendo el menor número posible de características necesario para trabajar con ejemplos como los vistos en clase. Muchas facilidades del lenguaje, como la posibilidad de definir operadores en notación infija o prefija, resultan irrelevantes para los objetivos de la práctica. Otras facilidades, como la posibilidad de definir clases genéricas (disponible en la versión 2), están más allá del nivel de profundidad aconsejado para la asignatura. El seminario incluye muchos ejemplos y ejercicios, algunos de los cuales se deberán hacer en la misma sesión. Otros deberán ser resueltos por los alumnos y entregados posteriormente al profesor. Estos ejercicios serán corregidos, dando lugar a la nota correspondiente a esta práctica. Se estima que la dedicación personal de los alumnos en la resolución de los ejercicios será de unas 4 horas.

Capítulo 8

Metodología docente y evaluación

"Guardaos de dárselo todo hecho a los niños, que esto no excita ni desarrolla sus facultades; [...] *un buen maestro debe interrogar frecuentemente a sus discípulos y sondear su juicio, porque si no se hacen activos, ¿cuál es el objetivo de la enseñanza sino ponerlos en estado de pasarse sin maestro?"*

Marco Fabio Quintiliano, Instituciones Oratorias, Siglo I d.C.

La puesta en práctica de un proyecto docente implica necesariamente la aplicación de un método didáctico, que abarca el conjunto variado de actividades que tienen lugar en la interacción entre el profesor y los alumnos. El respeto a los alumnos, como aprendices adultos, debe marcar esta relación desde el punto de vista del profesor. Si hasta este Capítulo nos hemos centrado en los objetivos propios de la materia objeto de concurso –justificados en el contexto en el que se desarrolla– y en la propuesta de contenidos para la asignatura, ahora trataremos los aspectos exclusivamente pedagógicos del proyecto docente, es decir, cómo son organizadas las actividades de clase y de laboratorio, y cómo es realizada la evaluación del proceso de aprendizaje.

Los componentes propios de un método docente universitario incluyen normalmente las siguientes actividades:

- **La lección magistral**, como el método más ampliamente utilizado en el desarrollo de las clases de teoría; aunque no por ello el único ni necesariamente el más adecuado.

- **Las clases de problemas**, utilizadas como un modo de apoyar la consolidación de los conocimientos teóricos.

- **Los seminarios o laboratorios cerrados**, usados para la presentación y desarrollo de las actividades prácticas bajo una guía y supervisión directa del profesor.

- **Las prácticas o laboratorios abiertos**, que permiten el trabajo autónomo de los alumnos en la elaboración de las prácticas, de manera no supervisada.

- **Las tutorías**, como un medio para complementar las restantes actividades, ofreciendo un apoyo directo e individualizado de los alumnos.

El método docente hace referencia tanto al desarrollo individual y específico de cada una de estas actividades, como a la integración y coordinación de todas ellas con el propósito de alcanzar los objetivos propuestos. Ningún método concreto puede atribuirse el derecho de ser "el mejor" [Martínez'01], sino que la mayor o menor adecuación está condicionada por los factores que afectan al contexto. La meta final del diseño didáctico será lograr una **enseñanza de calidad**, adaptada a las circunstancias del contexto educativo y expuesta a una revisión y actualización continua.

Finalmente, el desarrollo de la acción docente culminará con la **evaluación** del aprendizaje de los alumnos, y la evaluación del propio proceso educativo. Pero, obviamente, la evaluación empieza a desempeñar su papel desde el principio, como el principal elemento de motivación para muchos de nuestros alumnos.

8.1. Objetivos y calidad de la enseñanza

El objetivo más directo de la docencia universitaria es conseguir el aprendizaje de los alumnos en las mejores circunstancias. Encontrar las formas en las que se puede incrementar y hacer más duradero y consistente este aprendizaje es uno de los propósitos del diseño de un método docente. Las actividades de clase deberán estar orientadas hacia la consecución de una enseñanza de calidad, entendiendo la calidad en función de los resultados del aprendizaje.

Tradicionalmente, en el mundo universitario, el concepto de calidad de la enseñanza ha estado sometido a la subjetividad de quien lo esgrime. Los profesores universitarios entienden la docencia como una actividad intuitiva, no regida por los procesos de análisis, formalización y revisión propios de una disciplina científica en sí misma, [Zabalza'99]. Esto es debido a la falta de una formación didáctica de los profesores, que desde su entrada a la universidad empiezan –hemos empezado– a dar clases sin una preparación específica en el arte y ciencia de enseñar. Algunas universidades, conscientes de este problema, han adoptado varias iniciativas para tratar de paliar este problema[1], aunque no existe una verdadera conciencia de esta carencia a gran escala.

Si el estudio de las técnicas de enseñanza es una ciencia, la **didáctica** es su disciplina. En concreto, la docencia universitaria es uno de sus ámbitos, y son numerosas las investigaciones realizadas al respecto. Destacamos aquí el artículo [Zabalza'99], donde el autor distingue los 10 rasgos que, según su criterio, caracterizan una enseñanza de calidad. Adoptamos estos rasgos como un punto de partida para el posterior diseño de un método docente y de evaluación. Los rasgos distinguidos por Zabalza son los siguientes.

1 - Proyecto docente y formativo

Indudablemente, uno de los primeros factores imprescindibles para lograr la calidad es el análisis, diseño y planificación previos al desarrollo de la propia acción docente. Esta organización por adelantado de las actividades a realizar se plasma en el proyecto docente y, de forma más sintética y específica, en el programa de cada asignatura. La elaboración del proyecto formativo obliga a reflexionar, plantearse, discutir y consensuar los objetivos, contenidos y métodos que resultarán más adecuados para cada materia; frente a la improvisación que produciría la inexistencia de un proyecto. El documento debe ser público y estar sometido a críticas o propuestas de mejora. Su existencia constituye una especie de compromiso formal sobre el desarrollo de la actividad propuesta.

[1]Por ejemplo, en la Universidad de Valencia, el Servicio de Formación Permanente ofrece a los nuevos profesores unas guías sobre la elaboración de programas, la evaluación y la lección magistral, con numerosos consejos prácticos y recomendaciones. Puede encontrarse más información en: `http://www.uv.es/~sfp/pdi`

La elaboración del proyecto docente es una ocasión única que nos permite replantearnos la enseñanza de la asignatura, considerando: la coherencia entre el programa de la asignatura y el proyecto formativo de la titulación, la coordinación entre nuestro programa y los de las otras materias relacionadas, la relación teoría-práctica en el programa, la estructura y los componentes curriculares del programa, la riqueza informativa y el valor de orientación del programa y su originalidad.

2 - Condiciones y ambiente de trabajo

Las condiciones externas en las que se desarrolla la actividad docente son otro de los rasgos que permiten o impiden la consecución de la calidad: los espacios físicos donde se imparten las clases y las prácticas; los recursos y materiales disponibles, como ordenadores, biblioteca; el ambiente de trabajo en el aula; los servicios auxiliares con los que cuentan los profesores y los alumnos, etc. Por muy obvio que parezca, difícilmente se podrá lograr la calidad en un aula mal iluminada, con una mala disposición de los pupitres, con medios técnicos defectuosos, o con aire acondicionado ruidoso.

La calidad de los espacios acaba teniendo influencia en las alternativas metodológicas que el profesor pueda utilizar, en el nivel de implicación de los estudiantes y en la satisfacción de profesores y alumnos. Incluso en las metodologías tradicionales, basadas en la lección magistral y caracterizadas por un aprendizaje monótono, es necesario que las circunstancias de clase garanticen un buen clima para la concentración de los alumnos.

En este sentido, posiblemente el mayor impedimento de nuestra Universidad es la masificación en las aulas. En una clase con 170 alumnos matriculados, a la que asisten 80 por término medio, es realmente difícil lograr una enseñanza de calidad centrada en el alumno. En estas grandes clases, muchos sitios presentan una mala perspectiva de la pizarra, además de la dificultad de la comunicación de los alumnos con el profesor.

3 - Selección y presentación de contenidos

Obviamente, de poco serviría lograr altos niveles de los restantes factores si los contenidos impartidos no son de calidad, no son significativos o relevantes en la disciplina, o son presentados de una manera inadecuada. Los contenidos son, por lo tanto, un aspecto fundamental en la calidad de la docencia. La cantidad de contenidos enseñados debería alcanzar un justo equilibrio, que evite los extremos de un programa excesivamente cargado, hipertrofiado e imposible de enseñar o aprender en el tiempo disponible, ni llegar al extremo de omitir conocimientos esenciales de la materia. En definitiva, unos buenos contenidos serán aquellos que resulten más importantes dentro de ese ámbito disciplinar, acomodados al perfil profesional, adecuados a las condiciones de tiempo y de recursos con que contamos, y organizados de tal manera que sean accesibles a nuestros estudiantes y que les abran las puertas a aprendizajes posteriores.

Pero no sólo los contenidos en sí son relevantes, sino también la forma en la que son presentados. Una enseñanza de calidad requiere evitar la uniformidad y monotonía. Hay que ser capaz de transmitir un mapa de los conocimientos de la asignatura en relieve, resaltando los elementos más sustanciales y sabiendo colocar cada parte en su justo lugar. Además, el papel del profesor como comunicador, guía y animador es insustituible, frente otros materiales o recursos alternativos.

Algunos consejos en este sentido que se ofrecen en [Zabalza'99] son: incorporar organizadores previos que faciliten la comprensión de los contenidos; tratar con suficiencia los puntos claves de la disciplina; diferenciar entre los conceptos y estructuras básicas y los complementarios; vincular los contenidos de la materia con otros contenidos; incorporar contenidos opcionales; combinar elementos narrativos y elementos conceptuales en los contenidos; combinar entre teoría y práctica de forma adecuada; mejorar la riqueza comunicativa con que son explicados; introducir elementos destinados

a potenciar una interacción fluida y constante (preguntas, ejercicios de comprobación, diálogo, etc.); incorporar actividades de repaso y sistemas de reorganización de los contenidos.

4 - Materiales de apoyo a los estudiantes

Además de la exposición presencial de los contenidos, los alumnos deben recibir un material de apoyo suficiente y de calidad: guías, dossieres, ejercicios, etc. Cualquier enseñanza, y más propiamente en aprendices adultos, debe tener un cierto componente autodidacta. Estos medios son más trascendentes en una situación de masificación como la de nuestras universidades, que dificultan un contacto personal frecuente con nuestros alumnos. Los materiales de apoyo juegan, en parte, esa función mediadora, mostrando las orientaciones para seguir un proceso de aprendizaje adecuado a la naturaleza de los contenidos. Además pueden servir para introducir contenidos o ejercicios adicionales. Cuestiones que se incluyen en este tipo de materiales para guiar el aprendizaje del alumno son las siguientes:

- Qué tipo de conocimientos suelen tener bien asentados y cuáles son las lagunas con que llegan a nuestra materia.
- Qué tipo de consideraciones podrían hacerles sentir la importancia de cada tema y su utilidad.
- Qué aspectos de cada tema merecen la pena ser resaltados para que centren su atención en ellos.
- Qué informaciones podemos incluir en el material (tablas de datos, fórmulas, gráficos, etc.) para que no pierdan tiempo copiando cuando estemos explicándolos.
- Qué tipo de prácticas podrían hacer dentro de cada tema (y cómo se podrían guiar esas prácticas desde este material escrito).
- Qué ejercicios se les puede proponer como forma de reforzamiento de los aprendizajes, como forma de que ejerciten opciones personales eligiendo entre alguna de las alternativas que se les ofrecen (y que podrían incluso ir graduadas por niveles de dificultad), como forma de repaso, etc.
- Qué actividades de autoevaluación les podemos incluir.
- Qué bibliografía (o fuentes de información, incluidas las electrónicas) les aconsejamos para seguir profundizando en ese tema.

En la asignatura AED contamos con suficiente material adicional de apoyo, entre el que podemos destacar: texto guía de la asignatura, transparencias de clase, boletines de ejercicios para cada tema, y página web de la asignatura. Lo describiremos en detalle más adelante.

5 - Metodología didáctica

Evidentemente, como ya hemos comentado, los métodos didácticos usados desempeñan un papel muy importante en la calidad de la docencia, fundamentalmente en aquellas actividades que suponen interacción entre profesor y alumnos. Son muy diversos los métodos didácticos que se pueden aplicar en la Universidad. Atendiendo al grado de participación y control de los profesores, podemos encontrar desde la lección magistral, la enseñanza a pequeños grupos, la supervisión de trabajos, el trabajo de laboratorio, los sistemas de autoinstrucción hasta el estudio independiente. Nuevamente, el grado de masificación de las clases será un elemento limitador en las posibilidades de actuación. A medida que

aumenta el número de alumnos en el aula, disminuye la capacidad para mejorar la participación de los alumnos.

Podemos citar las siguientes peculiaridades que subyacen en un método docente:

- El **estilo** al plantear los contenidos de la materia, desde un estilo retador y problemático, hasta un estilo más basado en nociones y principios.
- El **grado de dependencia o independencia** con que se plantea la actividad didáctica, es decir, la combinación entre control y trabajo autónomo.
- Las modalidades de **interacción entre los alumnos** pudiendo organizarse el trabajo en: grandes grupos, pequeños grupos, y trabajo individual.
- Equilibro entre **exigencias** en calidad, cantidad, tiempo, rigor y forma de presentación.
- El estilo de **interacción entre profesor y el alumno**: accesibilidad frente a distancia; cordialidad en el trato frente a frialdad y trato formal. Este factor determinará el clima de clase.
- **Implicación** real de los estudiantes en las actividades desarrolladas: escuchar, copiar, participar, colaborar, etc.
- **Graduación** del método: desde momentos iniciales más proclives a la ampliación de informaciones y experiencias hasta momentos en los que se antepone la precisión y aplicación de conocimientos.
- Propuesta de **experiencias** *fuertes* que rompan la dinámica anodina y lineal de las clases, como por ejemplo: una visita a un escenario profesional, un intercambio con estudiantes de otros países, unas prácticas en una empresa, etc.

6 - Nuevas tecnologías y recursos diversos

Las nuevas tecnologías permiten un mayor espectro de experiencias educativas, que pueden mejorar la comprensión de los alumnos. Su uso puede tener lugar en distintos ámbitos: en las explicaciones de clase, como material para el trabajo autónomo de los alumnos, para las prácticas, y como mecanismo para la interacción a distancia entre profesor y alumnos. Pero la incorporación de las nuevas tecnologías debe venir supeditada a los objetivos formativos y no emplearse indiscriminadamente. Zabalza identifica algunas condiciones que deberían darse para que su aplicación sea realmente beneficiosa:

- Que los medios tecnológicos estén **integrados** efectivamente **en el currículum formativo** de los estudiantes de manera que puedan sacarles el máximo partido. No se debe suponer que los estudiantes ya conocen los diversos recursos disponibles.
- Que supongan un **avance real** en el enriquecimiento y actualización de los procedimientos de aprendizaje.
- Que permitan ampliar el espectro de **experiencias de aprendizajes** de los alumnos.
- Que faciliten los **intercambios y la transferencia** (modelos de aprendizaje interactivo y en redes) entre escuelas y entre sujetos.
- Que permitan un **aprendizaje más autónomo** por parte de los sujetos.

- Que **no signifique un factor discriminador** entre los estudiantes con recursos y los que no disponen de ellos.
- Que los nuevos recursos **no distraigan la atención** de los alumnos, prestando mayor interés por el medio que por los contenidos sustantivos de la materia.

Creemos que todos estos factores se cumplen en una titulación de ingeniería en informática, haciendo muy aconsejable el uso de estos medios técnicos. Refiriéndonos específicamente a la asignatura AED, la utilización en clase de las nuevas tecnologías es muy recomendable. El uso de animaciones permite mostrar de forma intuitiva el funcionamiento de muchos algoritmos y estructuras de datos, facilitando, acelerando y mejorando la comprensión de los alumnos.

7 - Atención y sistemas de apoyo a los estudiantes

El papel del profesor como orientador y guía del aprendizaje de los alumnos es insustituible. Lograr una atención individualizada de calidad es necesario para mejorar la calidad de la enseñanza. Pero en la relación no sólo influye la orientación y las indicaciones del profesor, sino también en su propio ejemplo como profesional. Los alumnos aprenden de nuestro interés por el ámbito científico, de la manera en que concebimos la profesión, del estilo de nuestro trabajo, de nuestro compromiso por estar al día, de nuestra sensibilidad por los demás, de nuestra visión del mundo, etc.

Pero la masificación de nuestras universidades dificulta de manera determinante la posibilidad de ofrecer una atención con la suficiente dedicación. Esto da lugar a diferentes problemas: se incrementa el fracaso escolar y el abandono, el contacto entre profesor y alumno individual acaba siendo mínimo, produce sentimiento de anonimato en los estudiantes, lo que propicia una actitud pasiva y receptiva por parte del alumnado.

Zabalza ofrece algunos consejos sobre las cualidades que debe mostrar un profesor en su relación con los alumnos: sensibilidad hacia los estudiantes; capacidad de transmitir interés y crear retos; atender los intereses e inquietudes personales de los alumnos; sistemas de apoyo que se hayan creado para resolver las dificultades que vayan apareciendo en el aprendizaje; accesibilidad a las demandas de los estudiantes; mantener altas expectativas respecto a su trabajo; etc.

8 - Coordinación con los colegas

Los anteriores puntos se centran en la labor individual de los profesores. Pero la calidad de la docencia también depende de la adecuada coordinación y trabajo en equipo entre los docentes. Esta relación se debe dar tanto en el sentido horizontal, es decir, entre todos los profesores de la asignatura o del grupo docente, como en el sentido vertical, esto es, con los demás profesores del mismo curso o de otros cursos. Para lograrlo puede ser interesante la creación de comisiones de coordinación, que fomenten la comunicación e interacción entre los profesores que imparten asignaturas de un mismo perfil disciplinar. Además, los profesores deberían conocer el proyecto formativo completo en el que participan para adquirir una perspectiva global que les permita emitir juicios sobre las posibles interacciones entre asignaturas.

La actitud individualista de muchos profesores es el mayor obstáculo para la consecución de este factor. La raíz de los conflictos radica en la tendencia a convertir las asignaturas que impartimos en algo propio, que nos pertenece y que deseamos salvaguardar de la curiosidad y de la intervención de personas contrarias a nuestras posturas. Frente a esto, la discusión y el consenso en los grupos docentes siempre resulta enriquecedora para la mejora de la calidad docente.

En este sentido, pensamos que en la asignatura AED existe una buena coordinación y ambiente de trabajo entre los profesores involucrados en su docencia en las diferentes titulaciones. Fruto de esta

colaboración es la utilización de un programa consensuado en las tres titulaciones, y la elaboración de forma conjunta del texto guía de la asignatura durante el año pasado.

9 - Sistemas de evaluación

La evaluación es uno de los elementos básicos de la actividad docente universitaria, por lo que su mejora resulta esencial para incrementar la calidad de la enseñanza. La evaluación, inicialmente concebida como un instrumento para valorar la progresión en el aprendizaje de un alumno, se puede convertir incluso en un arma amenazadora que actúa como única motivación para el alumnado.

La evaluación es la parte de la actividad docente que más repercusiones produce sobre los alumnos: autoestima, motivación por el aprendizaje, económicas, familiares, etc. Por esta razón, debemos prestarle especial atención. La evaluación es una actividad delicada, en cuanto que desempeña un papel discriminador, determinando si un alumno ha alcanzado o no los objetivos de la asignatura. Por un lado, es necesario que se garantice un suficiente nivel de exigencia, acorde con los requisitos de una titulación universitaria. Por otro lado, debería ayudar a motivar a los alumnos menos aventajados a superarse en la asignatura.

De nuevo indicamos algunas de las recomendaciones realizadas en [Zabalza'99], respecto a las características de una evaluación de calidad:

- Distinguir entre la evaluación de seguimiento del aprendizaje y la evaluación de control que será la base para la calificación empleada en la acreditación del rendimiento.
- Coherencia con el estilo de trabajo desarrollado: evitar exámenes memorísticos, preguntas muy rebuscadas, un reducido número de preguntas cuando los programas son amplios, configurar el examen en base a un documento, etc.
- Variedad y progresión en las demandas cognitivas: incluir preguntas sobre conocimientos de diferente índole y nivel de dificultad.
- Ofrecer información antes y después de la evaluación. Los criterios de evaluación deben quedar claros en el programa. Después de realizar y corregir el examen, se debe ofrecer al alumno información sobre cómo le fue y en qué puntos debería trabajar para conseguir mejores resultados.
- Asegurar un sistema de revisión de exámenes y de atención de reclamaciones.

10 - Mecanismos de revisión del proceso

Una docencia de calidad no puede estar basada en unas pautas invariables fijadas de antemano; debe estar más bien sujeta a la revisión y actualización continua. Sólo de esta manera se podrá mejorar la calidad de la enseñanza, que de otra forma quedaría obsoleta con el tiempo. La enseñanza docente de calidad es un proceso en espiral, generando un nuevo producto partiendo de las revisiones de producto anterior.

Para este propósito es necesario contar con mecanismos de observación y recogida de información sobre la marcha del proceso de enseñanza-aprendizaje. Todos los puntos deben estar sujetos a la revisión.

Algunos datos que pueden resultar interesantes para su recopilación y posterior evaluación podrían ser:

- El diagrama de los resultados en cada una de las materias y actividades desarrolladas durante el curso.

- Nivel de uso de los diversos recursos existentes y satisfacción con los mismos.
- Satisfacción general del profesorado y del alumnado sobre la marcha del curso: horarios, carga lectiva, recursos, coordinación, etc.
- Indicadores externos relativos a la eficacia institucional: nivel de empleo efectivo de los graduados; valoración por parte de los patronos del nivel logrado por los recién licenciados; valoración de la formación recibida por parte de los titulados con varios años de experiencia, etc.

A estos 10 factores identificados por Zabalza podemos añadir como un elemento imprescindible la adecuada preparación de los profesores. Como ya hemos comentado al principio, muchos problemas en la enseñanza universitaria se derivan de una falta de preparación pedagógica de los profesores. Saber cómo aprenden los alumnos, cómo motivarlos y cómo hacer uso del tiempo de clase de la mejor forma es necesario para alcanzar una docencia de calidad. En este sentido, recordamos las consideraciones realizadas en el apartado 2.3.3, respecto a la teoría del aprendizaje del **constructivismo**, que es actualmente la predominante en el ámbito de la pedagogía. De forma resumida, las tres principales conclusiones que extraíamos de esta teoría eran: (1) el aprendizaje es siempre un proceso activo, ya que el conocimiento no se transmite de forma directa, sino que cada individuo construye sus propias estructuras de conocimiento; (2) la analogía, como medio de relación entre los conocimientos previos y los nuevos, es uno de los principales mecanismos de construcción del conocimiento; y (3) para guiar a los alumnos correctamente es necesario "ponerse en su cabeza", entendiendo las estructuras mentales de conocimiento que se están formando en su interior.

Por otro lado, más allá de los objetivos educativos, en los que nos hemos centrado hasta ahora, la docencia universitaria debe conseguir otros propósitos más amplios, relacionados con la formación de los alumnos como ciudadanos y como profesionales, como ya comentamos en el Capítulo 3 (Finalidades y objetivos). Entre estos propósitos podemos señalar: alcanzar una formación comprensiva e integrada; fomentar una actitud crítica constructiva; adquirir hábitos de indagación, observación, reflexión y auto-evaluación; potenciar la capacidad de planificar, desarrollar y evaluar; fomentar el trabajo en equipo y la toma de iniciativas; e impulsar el aprendizaje no dirigido. Todas estas metas no pueden formar parte de los contenidos de una asignatura, pero deben estar presentes en el desarrollo de todas las actividades docentes.

8.2. Métodos didácticos

Ya hemos visto que un método docente universitario debe basarse en la combinación adecuada de un conjunto variado de tipos de actividades, que van desde la lección magistral –totalmente guiada y controlada por el profesor–, hasta las prácticas no supervisadas –donde predomina el trabajo autónomo de los alumnos–. Una asignatura no puede estar centrada exclusivamente en un solo tipo de actividad, sino que debe aprovechar las ventajas de todas ellas. La necesidad de diversificar los métodos didácticos surge de la propia diversidad de objetivos que persigue la formación universitaria: adquisición de conocimientos, desarrollo de habilidades, actitudes y valores, y desarrollo personal. Vamos a analizar los distintos métodos didácticos que consideramos relevantes en la enseñanza de la materia AED, haciendo énfasis en las peculiaridades concretas de esta materia.

8.2.1. La lección magistral

De acuerdo con [SFPlecc'02], *"entenderemos por lección magistral el modelo de desarrollo de clase en el que el docente lleva el mayor peso del discurso, donde la información y la organización de*

ideas, contenidos, etc., descansan fundamentalmente en la exposición verbal por parte del profesor". Aunque en su aplicación es posible obtener información de los alumnos, a través de la realización de preguntas o ejercicios, se trata de una forma de comunicación predominantemente unidireccional, que ha evolucionado muy poco desde las primeras universidades medievales en las que los profesores leían sus propias notas y manuscritos. Pero como se señala en [Valcárcel'01], *"la lección magistral no es mejor ni peor que otros métodos de enseñanza, depende de los objetivos que se pretenden alcanzar y cómo se desarrolla en la práctica"*.

El arraigo de este método en nuestro sistema universitario surge de la tradición –podríamos decir casi milenaria– tanto por parte de los profesores –que se encuentran más cómodos controlando el desarrollo de las clases–, como de los alumnos –que están acostumbrados a tomar apuntes y recibir la información bien estructurada–. Por su parte, los profesionales de la pedagogía insisten en que es necesario innovar y aplicar otras técnicas en las clases de teoría, que potencien la implicación de los alumnos en su propio aprendizaje. Sea como fuere, está claro que la lección magistral es el método más ampliamente utilizado en nuestras clases para el desarrollo teórico de las asignaturas. Por este motivo, vamos a estudiar los objetivos y características de este método didáctico.

Objetivos, ventajas e inconvenientes

La lección magistral, también denominada en ocasiones técnica expositiva, debe perseguir principalmente tres grandes objetivos:

- **Facilitar información**, mediante explicaciones estructuradas y claras.
- **Organizar los contenidos**, estructurando la materia de forma comprensible.
- **Motivar al alumno**, fomentando la reflexión y el desarrollo de capacidades como: resumir, interpretar, analizar, aplicar, etc.

Pero si nuestro objetivo docente es mejorar el aprendizaje de los alumnos y optimizar su formación, debemos tener claro que no necesariamente existe una relación directa entre lo que exponemos en clase y lo que aprenden los alumnos. Desde los planteamientos constructivistas, que presentamos en el apartado 2.3.3, se sugiere que el conocimiento no es una *copia* de la realidad existente o de una información exterior. El aprendizaje es más bien *"un proceso interactivo mediante el que la información externa es interpretada y asimilada desde la estructura cognitiva del aprendiz, que va construyendo así progresivamente modelos explicativos cada vez más complejos"*, [Valcárcel'01]. En consecuencia, el estudiante no es un receptor pasivo de información; lo importante para el aprendizaje es la actividad mental que le permite relacionar la nueva información con la estructura de conocimiento que ya tiene.

Muchas de las principales críticas realizadas a la lección magistral se entienden desde los principios de esta teoría del aprendizaje. En concreto, podemos señalar los siguientes inconvenientes:

- Parte de la suposición falsa de que existe una relación directa entre lo que enseña el profesor en la exposición y lo que el estudiante aprende. Esto puede ocurrir, por ejemplo, cuando se dan por supuestos conocimientos que no tienen los alumnos.
- **Reduce las fuentes de información** a la palabra del profesor.
- Reduce las funciones del profesor en el aula a la transmisión de información.
- Favorece la **pasividad de los alumnos**, ya que incluso si el alumno presta atención, toma apuntes, y sigue detenidamente lo que expone el profesor, su actividad mental y física es solamente receptora y no creativa.

- Incita a la **memorización** a partir de los apuntes tomados en clase.
- Existe una **falta de control** de lo que el alumno va comprendiendo o aprendiendo.
- Se adecúa a un alumno tipo que imagina el profesor sin posibilidad de contemplar las diferencias individuales, suponiendo una uniformidad en el ritmo de aprendizaje.
- Predomina la **enseñanza** (actividad del profesor) **frente al aprendizaje** (actividad del alumno).
- Sólo permite que el alumno conozca y, en el mejor de los casos, comprenda lo que se transmite, no siendo posible el aprendizaje de procedimientos y actitudes.
- La **relación profesor-alumno es escasa** y unidireccional.
- La eficacia está muy condicionada por el nivel de atención de los alumnos.

La mayoría de las críticas y limitaciones se derivan del hecho de que el método expositivo no propicia la actividad mental del alumno, sino la memorización de los contenidos presentados por el profesor. Pero pensamos que estos problemas realmente no son inherentes a la lección magistral, sino que se deben a una mala forma de entenderla. Por ejemplo, la exposición no reduce las fuentes de información si el profesor solicita en clase la lectura de algún documento. Es más, muchos de los inconvenientes aparecerían si el método expositivo fuera el único utilizado en una asignatura, lo cual no ocurre en AED, como tampoco en la mayoría de las asignaturas de nuestras titulaciones. Por su parte, existen argumentos a favor del uso de esta técnica, siempre que se haga de forma adecuada. Entre las ventajas reconocidas por algunos autores podemos señalar:

- Permite establecer una **primera relación personal** con los alumnos, promoviendo el interés, para una acción tutorial posterior.
- Presenta la **información de forma estructurada**, lo que facilita su comprensión.
- Permite **promover el entusiasmo** en el trabajo de la materia relacionando ese campo de conocimiento con los intereses de los discentes.
- Reduce la materia a los **puntos esenciales y básicos**.
- Ayuda a enfrentarse a contenidos desconocidos y de difícil comprensión.
- Centra los temas y **evita divagaciones**.
- Posibilita la presentación de contenidos en **poco tiempo**, lo que facilita el desarrollo del programa.
- Se avanza en el programa aunque el alumno tenga actitud reacia a trabajar.
- Ayuda en situaciones en las que hay una **bibliografía escasa o poco adecuada**, sintetizando fuentes de información dispersas y, a veces, de difícil acceso.
- Permite **integrar un conjunto de actividades** que faciliten la construcción del conocimiento y el aprendizaje de los alumnos.
- La información llega a un **gran número de alumnos** al mismo tiempo.

Posiblemente, los argumentos más esgrimidos y los que finalmente determinan el uso habitual de esta metodología son su capacidad de servir a un número elevado de alumnos, desarrollando muchos contenidos en un tiempo reducido. Estas ventajas permiten paliar dos problemas recurrentes de nuestro sistema educativo: la masificación en las aulas, y la escasez y pérdida de horas de clase. Todos hemos experimentado alguna vez con el uso en clase de una técnica más participativa, obteniendo resultados más o menos buenos pero con un consumo de tiempo varias veces mayor el requerido con una técnica meramente expositiva. Por ejemplo, en [Kopec'99] se presenta una experiencia de aplicación de una metodología basada en trabajos de investigación en equipo, en una asignatura similar a AED. Aunque reconociendo buenos resultados, los autores destacan que los alumnos acaban conociendo únicamente la técnica de diseño de algoritmos sobre la que han investigado.

Hacia una lección expositiva y activa

Si el carácter expositivo es la principal propiedad que define a la lección magistral, como ya hemos visto, es necesario convertirla también en una tarea activa para alcanzar plenamente sus objetivos formativos. Estos deberían ser los dos rasgos diferenciadores de una lección magistral de calidad. Valcárcel sugiere algunas recomendaciones para hacer más activas las clases magistrales:

1. **Diagnosticar qué es lo que el alumno sabe.** Para ello se pueden introducir diferentes actividades al inicio de la clase, como: hacer preguntas a toda la clase, pedir que recuerden lo que saben del tema, plantear problemas de dificultad creciente, pedir que expresen acuerdo o desacuerdo con determinadas proposiciones, etc.

2. **Favorecer el protagonismo de los estudiantes**, motivándoles y propiciando su participación activa.

   - La motivación en la exposición debe ser un objetivo prioritario, tratando de crear en el alumno intereses e inquietudes por el objeto de estudio; por ejemplo, a través del planteamiento de problemas o aplicaciones que justifiquen la utilidad de los nuevos conocimientos. Si el alumno no encuentra sentido a los contenidos que se enseñan, difícilmente realizará el esfuerzo de aprenderlos.
   - La introducción de otras tareas y actividades intercaladas con la exposición ayuda a renovar la atención, a generar interés, y dar oportunidades para que los estudiantes piensen, obteniendo una realimentación de su grado de seguimiento y comprensión. Entre estas actividades podemos incluir: la utilización de preguntas, el planteamiento y resolución de problemas, y las actividades en grupos.

3. **Promover un clima adecuado**, a través de un esquema de relaciones comunicativas, afectivas y de poder, donde nadie tenga miedo a exponer sus ideas o plantear cuestiones, pero estableciendo unas "reglas del juego" aceptadas por el alumno.

4. **Organizar los contenidos de la lección.** La organización de la lección es propia de cada disciplina, pero en cualquier caso debe tener también en cuenta los conocimientos previos necesarios de los alumnos. Se debe favorecer la *reactivación* de estos conocimientos, indicando las relaciones con los nuevos conocimientos. La utilización de esquemas organizativos explícitos del contenido en forma de estructuras conceptuales, constituye una herramienta didáctica de gran utilidad para evitar la disgregación de los conocimientos y su acumulación como compartimentos estanco.

5. **Utilizar organizadores previos.** Un organizador previo es un esquema o mapa conceptual cuyo objetivo es establecer un puente cognitivo entre lo que el alumno sabe y lo que necesita saber para aprender significativamente los nuevos conceptos. Estos constituyen un pequeño segmento de enseñanza, que se plantea al inicio de una unidad de información más amplia, o de la lección, y que constituye la idea más general del contenido de la unidad.

6. Contemplar tanto la **construcción de conocimiento como la competencia de utilizarlo y expresarlo**. Ambos son aspectos fundamentales del aprendizaje. Normalmente incidimos en el primero, pero apenas dedicamos tiempo a desarrollar las habilidades que requiere la construcción de respuestas. Es necesario dedicar tiempo suficiente para que los alumnos utilicen y consoliden los aprendizajes con una práctica reiterada del conocimiento.

Además de estos grandes ejes, se señalan otras recomendaciones para el desarrollo de las clases:

- Apoyarse en imágenes visuales, como la pizarra, transparencias, animaciones, etc.
- Utilizar guiones y el texto de la lección.
- Disponer del texto escrito, lo que evita el agobio que supone tomar apuntes de todo.
- Usar la voz de forma adecuada, evitando la monotonía, hacer llamadas de atención, pausas, subir y bajar el tono, hacer inflexiones, interrogaciones, etc.
- El manejo del cuerpo también es importante. Es más fácil captar la atención estando de pie y moviéndose que estando sentado en el asiento.
- Otros consejos: espontaneidad y naturalidad en la expresión, dicción clara, convicción y entusiasmo en lo que se trasmite, respeto a los alumnos, y ritmo adecuado de la exposición.

Todas estas cuestiones requieren una adecuada preparación previa de las clases por parte del profesor. En la asignatura AED, hacemos uso de un cañón de vídeo, con el que se proyectan transparencias y animaciones previamente diseñadas en el ordenador. Estas transparencias están disponibles para los alumnos por lo menos una semana antes de cada clase. Además, existe la posibilidad de consultar el texto guía de la asignatura para obtener información ampliada, ya que las transparencias son necesariamente esquemáticas e incompletas.

Secuenciación y desarrollo de la exposición

El desarrollo de la lección magistral debe contemplar diferentes fases. Se pueden encontrar distintas propuestas a este respecto, pero básicamente todas ellas distinguen entre las fases de introducción, desarrollo y conclusiones. Algunos autores añaden una fase de aplicación de los conocimientos aunque, desde nuestro punto de vista, podría ser simultánea con el desarrollo de la lección. La descomposición en fases persigue dos propósitos: motivar y consolidar. Si la introducción permite despertar el interés por los contenidos que van a ser estudiados, las conclusiones buscan consolidar los conocimientos adquiridos. Por lo tanto, cada fase persigue unos objetivos diferenciados.

- **Fase de introducción.** Los objetivos de esta fase son: presentar la lección, contextualizarla en el programa de la asignatura, y motivar el interés por el tema. Una buena introducción debe empezar resumiendo los aspectos fundamentales estudiados en la lección anterior, sobre todo si esa lección pertenecía al mismo tema. Esta tarea puede suponer un reto, ya que significa sintetizar en pocas palabras lo más relevante de los contenidos explicados en la hora, o dos horas,

anteriores. De esta manera se recuerda lo más importante, estableciendo un vínculo de conexión entre lo explicado un día y el siguiente. A continuación, se adelanta brevemente lo que se va a estudiar en la lección, destacando fundamentalmente los problemas que motivan su estudio. Este adelanto será mayor cuando se trata de una lección que dé comienzo a un tema nuevo.

Analizar aplicaciones llamativas puede resultar adecuado para despertar el interés de los alumnos por el tema. Por ejemplo, el estudio de las extraordinarias capacidades ofrecidas por el buscador Google, o los combates de ajedrez hombre/máquina y el Deep Blue, son algunos ejemplos concretos que han servido para motivar el estudio de las estructuras de datos y los algoritmos en la asignatura AAED. Consideramos que una introducción razonable debería durar en torno a 5 o 7 minutos.

- **Fase de desarrollo.** Durante el desarrollo tienen lugar una serie de tareas de diversa índole: explicación de nuevos conceptos, ejemplificación y aplicación de los conocimientos, resolución de ejercicios, planteamiento de preguntas, emisión de juicios críticos sobre el contenido tratado, etc. Su organización depende completamente del tema desarrollado, pero es adecuado que se combinen distintos tipos de tareas, con el fin de evitar la monotonía. Siguiendo a [Montoya'01], los principios básicos que deben guiar el desarrollo de la explicación son:

  - **Motivación.** Los contenidos que se expongan en cada tema deben ser siempre motivados previamente. La motivación empieza desde la introducción, en la que debe exponerse la importancia de los conceptos que se van a estudiar, en el por qué deben ser estudiados y, a ser posible, ilustrar qué ventajas o novedades aporta la nueva materia frente a lo ya conocido. Pero el interés por el tema debe mantenerse a lo largo del desarrollo de la lección. De esta manera, si se expuso algún ejemplo de aplicación en la introducción, se deberá retomar en algún punto de la explicación posterior.
  - **Continuidad.** Debe evitarse, en la medida de lo posible, las *fallas* o discontinuidades abruptas en la materia que se expone. Cada tópico que se presente debe ser previamente relacionado con la materia ya conocida por el alumno, haciendo énfasis en las nuevas posibilidades que la siguiente materia va a aportar a lo ya conocido.
  - **Estructuración.** Al comienzo de cada tema se debe proporcionar al alumno un buen esquema de todos los contenidos que se van a desarrollar. Este esquema será referenciado cada vez que se avance en el tema, con objeto que el alumno tenga claro en todo momento el punto de progresión en que se encuentra el tema tratado.
  - **Comunicación.** La clase magistral no debe de ninguna manera consistir en un monólogo por parte del profesor, sino que debe establecerse una comunicación bidireccional en la que tanto profesor como alumnos se sientan libres de interrumpir la clase en cualquier momento para realizar preguntas. Para esto es necesario que exista un clima de clase adecuado, que fomente la participación de los alumnos.
  - **Ejemplificación.** Además de los problemas y prácticas que posteriormente se desarrollarán tanto en las clases de problemas como en las prácticas de laboratorio, es fundamental la frecuente intercalación de ejemplos entre la exposición de los contenidos teóricos. Estos ejemplos deben ser lo suficientemente simples como para no suponer una fuerte interrupción del desarrollo de la materia teórica, y lo suficientemente complejos como para motivar de manera adecuada la materia presentada.
  - **Seguimiento.** A pesar de ser una tarea difícil debido al problema de la masificación en las aulas, el profesor debe desarrollar cierta psicología que le permita inferir en qué momentos

los alumnos (o la mayoría de ellos) han dejado de seguir la clase, o se han perdido en algún punto por alguna razón. En otras palabras, la habilidad del profesor para *leer en las caras* de los alumnos es, en nuestra opinión, importante a la hora de desarrollar de manera adecuada una clase magistral.

- **Fase de conclusión.** Realizar una rápida recapitulación al final de la lección facilitará la integración de los puntos más relevantes. De esta forma, la conclusión desempeña un papel de consolidación de los conocimientos. Adicionalmente, esta síntesis de lo estudiado puede servir para dar una nueva visión de la materia. Por ejemplo, en el caso de la asignatura AAED, las conclusiones después de estudiar diversas técnicas de estructuración de datos pueden servir para establecer de forma fundada las condiciones en las que puede resultar más adecuada la aplicación de cada técnica. Al mismo tiempo, se pueden dejar cuestiones abiertas, dando pie a los siguientes puntos del temario.

  Obviamente, las conclusiones aparecen al final de la lección. Dejar tiempo para las conclusiones puede ser difícil si intentamos aprovechar el tiempo de clase al máximo, la hora de clase ha terminado y los alumnos empiezan a mostrar impaciencia por salir. Un método que hemos usado en AAED para evitar esta precipitación en la fase de conclusiones es incluirlas explícitamente en las transparencias utilizadas en la lección. De esta forma, las conclusiones forman parte de los contenidos, incidiendo así en su relevancia dentro de la explicación. El tiempo dedicado a las conclusiones puede depender de muchos factores siendo, en principio, similar al dedicado en la introducción.

Un hábito común en nuestra Facultad es la disposición de las clases de una misma asignatura en grupos de dos horas, dentro de los horarios. Desde el punto de vista organizativo, esta estrategia simplifica muchos inconvenientes de un horario donde las asignaturas no tengan más de una hora seguida (en cuanto a desplazamientos al aulario, preparación de las clases, uso de recursos). Pero pensamos que desde el punto de vista pedagógico esta disposición presenta importantes desventajas: una sesión de dos horas de una misma asignatura siempre resultará más agotadora que si se tratará de dos asignaturas diferentes; aumenta el lapso entre una clase y la siguiente, existiendo una menor continuidad temporal. Durante el presente curso 2003/04, la asignatura AED en la II –que cuenta con dos horas semanales de teoría–, tiene concentradas sus dos horas los lunes. Esto permite una mayor flexibilidad en el uso de las horas, pudiendo extender más o menos la primera hora según resulte conveniente. La realización de un descanso de unos 10 minutos entre clase y clase se hace obligatoria. Y, por otro lado, las fases de introducción y conclusiones se ven acortadas entre las clases sucesivas.

La técnica de preguntas

La formulación de preguntas es una técnica muy importante en el proceso de enseñanza/aprendizaje. Intercaladas con la exposición, sirven para hacer las lecciones magistrales más participativas, rompiendo la unidireccionalidad predominante en las clases. De esta manera se consiguen dos efectos positivos: para el alumno, mantener despierto su interés y su atención ante la posibilidad de recibir una pregunta; para el profesor, obtener realimentación del nivel de comprensión y seguimiento de los alumnos.

Preparar buenas preguntas requiere tiempo. Una buena pregunta debe generar en los alumnos reflexiones e interés por responderlas. Algunas de ellas pueden estar planteadas explícitamente en las transparencias de clase, mientras que otras pueden surgir durante el desarrollo de la clase. Por parte del profesor, la realización de preguntas requiere paciencia para esperar la contestación y saber

escuchar para saber responder. En las respuestas, se debe *"adoptar una actitud de investigación y descubrimiento junto a los alumnos"*, [Valcárcel'01].

Las preguntas planteadas pueden ser de distintos tipos:

- Cuestiones de recordar, extraer conocimiento.
- Cuestiones que nivel el grado de comprensión.
- Cuestiones que requieren aplicación, solucionar.
- Cuestiones para fomentar el análisis y el razonamiento.
- Cuestiones que invitan a la síntesis y la creación de conocimiento.
- Cuestiones que promueven la evaluación y el juicio.

En principio, las preguntas deberían ser abiertas, pudiendo ser contestadas por cualquier alumno. En la práctica, esto hace que sean muy pocos los alumnos que participan en clase. Por ello, proponemos que las preguntas, o parte de ellas, se dirijan a estudiantes concretos. De esta manera se intenta mantener el nivel de atención sobre los alumnos.

El futuro de la lección magistral

Hoy por hoy, es indudable que la lección magistral es el método más usado en las universidades españolas. Pero esta situación tiene visos de cambiar a medio plazo. Las corrientes que impulsan el desarrollo del Espacio Europeo de Educación Superior promueven un modelo educativo basado en el aprendizaje de los alumnos, más que en la enseñanza de los profesores, como vimos en el apartado 2.1.2. Es inmediato pensar que esto es un ataque velado a las metodologías expositivas, en las que la unidad de tiempo –y por tanto el núcleo del sistema– es el número de horas de clase presencial. Al proponer un modelo centrado en el alumno, se pretenden potenciar las metodologías basadas en los sistemas de tutorización de los alumnos, al estilo de los modelos típicos en las universidades anglosajonas y americanas.

Las clases magistrales, tal y como las conocemos, deberán adaptarse al nuevo entorno educativo. Respecto a la situación actual, el nuevo método de clase se caracterizará por los siguientes aspectos. Antes de cada clase, los alumnos deberán dedicar por lo menos una hora en la preparación de la misma. Esto requerirá, por parte de los profesores, la difusión de materiales adecuados y la tutorización y orientación de cara a la preparación. Las actividades en el aula darán por hecho que los alumnos han leído y estudiado sobre el tema, por lo que se podrán centrar en los aspectos más problemáticos o novedosos. Los alumnos tendrán que olvidar el viejo principio de que "sólo entra para el examen lo que se ha visto en clase". Además, después de la clase los estudiantes tendrán que seguir trabajando sobre los contenidos, posiblemente resolviendo ejercicios y haciendo tareas de investigación. Nuevamente, esto supondrá una potenciación de la actividad del profesor como tutor de los alumnos.

Es difícil valorar el nivel de preparación de nuestro sistema educativo para la adaptación a esta filosofía propuestas desde las altas instancias. Para los profesores, será necesario desarrollar programas de formación pedagógica, que les permita adquirir habilidad en los nuevos métodos didácticos. Pretender aplicar los cambios sin esta formación podrá ser absolutamente contradictorio. Además, los docentes deberán elaborar nuevos materiales, adecuados para el trabajo autónomo de los alumnos. Para los estudiantes, el cambio de mentalidad implicará un mayor esfuerzo y dedicación individual. Actualmente, es difícil encontrar alumnos que dediquen tiempo al estudio, o a la realización de prácticas, hasta que la fecha del examen, o de la entrega de prácticas, es inminente[2]. Finalmente, desde el

[2]Como podemos deducir del número de tutorías atendidas a lo largo del curso.

punto de vista material, la masificación en nuestras aulas es un grave impedimento para la adaptación a este nuevo modelo de enseñanza. Con el número actual de estudiantes, los recursos humanos de las universidades se deberían multiplicar para permitir una tutorización adecuada y de calidad de los alumnos.

8.2.2. Las clases de problemas

Las clases de problemas son un complemento indispensable a las de teoría, y como tales deben ser convenientemente intercaladas con estas. Ayudan a fijar los conceptos y métodos aprendidos en la teoría, al tiempo que fomentan las capacidades analíticas y sintéticas de los alumnos. En las clases de problemas se resuelven cuestiones concretas sobre la materia, y se proponen problemas nuevos que estimulen la capacidad de generalización del alumno. Si bien es conveniente salpicar constantemente de pequeños ejemplos los contenidos teóricos, es casi más importante dedicar clases completas a la resolución de problemas adicionales, proporcionando tiempo suficiente para que los alumnos las resuelvan, incentivando la participación en clase, y exponiendo la resolución a los compañeros. Esto último es, en nuestra opinión, bastante interesante, ya que en muchos casos los alumnos pueden entender mejor un concepto explicado en palabras de un compañero que en palabras del profesor. Es más, a veces es incluso más productivo para el estudiante el comentario de una solución equivocada que la simple observación una correcta.

El empleo de transparencias por parte del profesor permite la rápida exposición de contenidos en las clases de teoría, pero para las clases de problemas conviene más el uso de la pizarra, con el fin de mejorar la comprensión del razonamiento seguido en la resolución del ejercicio. Las características que debe reunir una buena clase de problemas son las siguientes:

- **Nivel de dificultad progresivo.** Los problemas propuestos deben empezar con un nivel bajo de dificultad, aumentándolo de manera progresiva conforme se avance en el temario teórico. El propósito es que los alumnos pierdan el miedo a salir a la pizarra y enfrentarse con un problema, y adquieran cierto grado de autoconfianza. Idealmente, los problemas tratados deberían tener un nivel similar al exigido en el examen de la asignatura, siguiendo el principio de que no se puede evaluar a los alumnos algo para lo que no se les ha preparado. No obstante, también puede ser adecuado introducir problemas de menor dificultad, que consoliden los conceptos estudiados en teoría, pero resolución rápida y simple.

- **Representatividad.** Es conveniente que los problemas propuestos traten todos los puntos vistos en teoría, y sean representativos sobre el tipo de conocimientos y habilidades necesarios para resolver otros problemas dentro del mismo ámbito. Una buena opción es resolver problemas planteados en exámenes de años anteriores, y especialmente aquellos en los que los alumnos hubieran tenido más dificultad. No obstante, no conviene centrarse sólo en resolver exámenes, lo que equivaldría a decir que el objetivo de la asignatura es simplemente pasar el examen.

- **Participación.** Los problemas *no deben* ser resueltos en la clase de problemas por el profesor, sino por los propios alumnos. Es fundamental recordar este principio, ya que en otro caso la clase de problemas se puede convertir en una clase teórica más. Deben ser los alumnos los que expondrán sus soluciones, al tiempo que van explicando al resto de compañeros los razonamientos que les han conducido hasta la solución propuesta. Con demasiada frecuencia sucede que los alumnos tienen reparos a la hora de salir a la pizarra a plantear su resolución del problema, por miedo a cometer errores delante de todo el grupo. El anonimato de una clase masificada dispersa la "responsabilidad" de salir a la pizarra, de forma que ningún alumno se da por aludido.

Sin una participación suficiente y generalizada las clases de problemas pierden toda su efectividad y difícilmente pueden alcanzar sus objetivos. La importancia de este factor es tal que ha sido uno de los grandes *quebraderos* de cabeza de los profesores universitarios. Por ejemplo, [Montoya'01] propone dos formas no excluyentes de fomentar la participación:

- La consulta al profesor en el horario de tutorías de las soluciones obtenidas. De este modo, estos alumnos irán con más seguridad a las clases de problemas, y posiblemente venzan su miedo a salir a resolverlos.
- Que en lugar de salir a la pizarra, el alumno dicte al profesor las directrices generales y el esquema de resolución del problema, y que sea el profesor quien realice la solución que se pueda obtener en base a todo ello.

Pero, desde nuestro punto de vista, es necesario aplicar medidas más expeditivas para conseguir que una gran mayoría de los alumnos –y no sólo unos cuantos– lleven hechos los ejercicios. En la asignatura AAED hemos aplicado hasta la fecha dos medidas:

- Dar puntos adicionales en la nota por la realización de ejercicios. La puntuación es siempre positiva, o nula, y puede incrementar hasta en 1 punto la nota del examen, siempre que supere un mínimo. En concreto, se da medio punto por cada problema resuelto correctamente.
- Indicar previamente a los alumnos los ejercicios que serán resueltos en la clase de problemas. Puesto que los boletines de ejercicios cuentan con unos 20 ó 30 ejercicios, es prácticamente imposible que se resuelvan todos en clase.

Ambas medidas se justifican únicamente por la necesidad de incrementar la participación de los alumnos, pero ni aun así se consigue una participación masiva de los alumnos.

- **Crítica.** Aunque en un cierto momento se llegue a una solución correcta para un problema dado, es muy deseable que otros alumnos planteen otras soluciones alternativas. Ante varias soluciones correctas, se plantearán cuestiones acerca del modo en que éstas han sido obtenidas y sobre la mejor o peor adecuación de la solución elegida para el problema propuesto. Es más, tras la resolución de un ejercicio en la pizarra por parte de un alumno, sería conveniente que fueran los mismos alumnos los que juzgaran la corrección o no de la misma.

Las clases de problemas aportan una información esencial tanto al profesor como a los alumnos. Para el profesor, esta actividad le ofrece una realimentación sobre el grado de comprensión de la materia, mayor que la que se puede obtener en las clases expositivas. Se pueden conocer los errores y lagunas más frecuentes de los estudiantes, que pueden ser distintos en diferentes cursos. Para los alumnos, el proceso de resolución le permite desarrollar la habilidad de poner en práctica los conceptos teóricos, verificando si su solución es correcta o no.

No podemos dejar de insistir en la necesidad de la participación de los alumnos en esta actividad, y más específicamente en una materia como los algoritmos y las estructuras de datos. A medida que el temario va avanzando, los problemas crecen de tamaño y requieren de una mayor dedicación. Muchas técnicas de diseño de algoritmos resultan de aplicación no trivial, y el tiempo de clase es insuficiente para que los alumnos piensen y propongan una solución. Si los alumnos no llevan preparados los ejercicios, será de poca utilidad que salgan a la pizarra a intentar escribir un algoritmo. En estos casos, nos vemos obligados a resolver nosotros los ejercicios, resultando que los alumnos se dedican exclusivamente a copiar lo que el profesor escribe en la pizarra.

En cuanto a la dedicación temporal a esta actividad, consideramos que el tiempo adecuado debería estar en torno a 1 hora de clases de problemas por cada 2 horas de clases teóricas. No está claro si estas horas deben entrar en la carga teórica o práctica de la asignatura, encontrándose propuestas en ambos sentidos. En este proyecto docente proponemos una solución intermedia, reservando algunas horas de teoría y otras de prácticas –sobre todo a la finalización de los seminarios– para la realización de clases de problemas. En cualquier caso deben contabilizar en la carga de la asignatura.

8.2.3. Los seminarios de prácticas

Los seminarios, también denominados "laboratorios cerrados", constituyen una actividad práctica que se desarrolla bajo la guía y supervisión directa del profesor, con el objetivo de introducir contenidos o herramientas concretos. En nuestro caso, el propósito de los seminarios es enseñar los lenguajes de programación que se usarán posteriormente en la resolución de los problemas propuestos. En particular, recordamos que para las prácticas de la asignatura AED nuestra propuesta incluye la realización de tres seminarios:

- Seminario de programación en C.
- Seminario de programación en C++.
- Seminario de especificaciones algebraicas en Maude.

La asistencia a los seminarios no es obligatoria, teniendo en cuenta que algunos pueden conocer ya el lenguaje, o bien no tienen posibilidad de asistir en las horas establecidas. No obstante, sí es muy recomendable para aquellos alumnos que no sepan manejar el lenguaje.

Los seminarios se organizan en sesiones de dos horas, realizadas en los laboratorios de prácticas. El seminario combina una parte expositiva con otra de aplicación de lo estudiado. En la primera –cuya duración no debería ser superior a una hora– se explican los conceptos, mecanismos del lenguaje o herramientas que serán de utilidad en la realización de la práctica. La explicación sigue los principios expuestos en la lección magistral, pero incidiendo aun más en la ejemplificación y aplicación práctica de los puntos tratados. Para su desarrollo es adecuado la utilización de un guión de prácticas, de manera que los alumnos tengan de antemano un documento donde se establecen los objetivos y estructura de los contenidos de esa sesión.

La explicación debe dejar lugar suficiente para la puesta en práctica de los conocimientos adquiridos en el ordenador. Para ello, en el guión de prácticas se deben incluir numerosos ejemplos y ejercicios de pequeño tamaño, que puedan ser resueltos en las horas del seminario. Otros ejercicios de mayor tamaño pueden incluirse también en el guión, para ser resueltos por los alumnos fuera del seminario. En ese caso, sería muy conveniente que en la siguiente sesión se empezara haciendo un repaso de las dificultades encontradas para resolver los problemas planteados. Es más, podría resultar conveniente proponer la entrega obligatoria o voluntaria de algunos de estos ejercicios. No obstante, no lo consideramos imprescindible en las prácticas de AED.

Por otra parte, los seminarios de prácticas no se pueden centrar exclusivamente en enseñar la sintaxis de un lenguaje de programación, sino que deben hacer un énfasis muy especial en las buenas prácticas de programación y en la calidad del software. Los guiones de los seminarios deben contener recomendaciones y sugerencias muy concretas sobre estos buenos hábitos de programación. Entre otros, podemos comentar los siguientes:

- **Separar interface e implementación**. En C y C++, las interfaces se colocan en los ficheros de cabecera (`.h` y `.hpp`) y las implementaciones en ficheros de código (`.c` y `.cpp`). Se dan consejos e indicaciones sobre cómo realizar esta separación.

- Usar un **buen estilo de programación**, en cuanto a sangrado del código, uso de identificadores significativos, introducción de comentarios, uso de constantes, etc.
- **Evitar** al máximo las **características de bajo nivel**. Aunque se introducen y los alumnos deben conocerlos, no resulta recomendable la utilización de características como la operación `goto`, la mezcla de tipos, y la aritmética de punteros.
- **Utilizar asertos** para comprobar condiciones de error en la precondición, en la postcondición, o en casos de error no recuperable. De manera superficial se introduce la idea de la programación por contrato.
- **Factorizar código**. En particular, usar genericidad siempre que sea posible.
- Conocer y hacer uso de las **librerías estándar** del lenguaje. Si una funcionalidad está ya disponible en alguna librería, no tiene sentido que los alumnos la vuelvan a implementar, escribiendo sus programas siempre desde cero.

Posiblemente, los problemas más frecuentes y graves en las prácticas de la asignatura son los relacionados con el descontrol en la aritmética de punteros. Los errores ocasionados por la lectura o escritura en una zona incorrecta de memoria pueden aparecer de forma aleatoria y esporádica; en una máquina parece que la ejecución es correcta, en otra sale un "`core dumped`", y en una tercera da el mensaje pero en otro punto distinto. Es difícil dar muchas recomendaciones al respecto, más allá de decir que se debe controlar que cada acceso a un puntero es a una posición correcto. Por su parte, en la asignatura "Sistemas Operativos" de segundo los alumnos estudiarán las herramientas de depuración, además del manejo de comandos y otras herramientas de Linux.

8.2.4. Prácticas de laboratorio abierto

Las prácticas de laboratorio abierto tienen importancia capital en la asignatura, puesto que conceden al alumno la oportunidad de aplicar los conocimientos adquiridos en la teoría. La principal ventaja didáctica de las mismas es que colocan al estudiante en una situación donde puede empezar a medir su rendimiento en condiciones más realistas, complejas y a más largo plazo que los ejercicios de clase. Al no estar supervisadas de manera directa, requieren una mayor implicación, lo cual suele estimular más a los alumnos, al verse más involucrados en la resolución de problemas concretos y tener contacto directo con el ordenador. Este efecto se puede aumentar si la práctica propuesta se enmarca dentro de un contexto de aparente utilidad. Este tipo de prácticas más o menos realistas aumentan el interés inicial de los alumnos por la asignatura, y su resolución les proporciona un mayor grado de satisfacción.

Ya justificamos en el apartado 4.4.3 las ventajas de que unas prácticas como las de AED estén centradas en el trabajo autónomo de los alumnos, frente a la organización de pequeñas prácticas guiadas o seminarios para todos los temas. No obstante, esto no evita la labor del profesor en la guía y seguimiento de los alumnos, que debe tener lugar en varios momentos:

- **Inicialmente**, al distribuir el enunciado de prácticas a los alumnos, se debe debería dar una sesión presencial en la que se explicaran los objetivos perseguidos por las prácticas, indicando qué es lo importante y qué no, aclarando dudas y posiblemente dando algunas sugerencias generales sobre la manera de afrontar el problema.
- **Durante el desarrollo**, el profesor estará disponible para resolver las dudas de los alumnos. Esta actividad no es presencial, sino que ocurre a iniciativa de los alumnos. Se debe conseguir un

equilibrio adecuado en cuanto a la libertad de elección del diseño (el profesor no puede imponer un esquema sin más) y la orientación para evitar soluciones inapropiadas (programas que funcionan "por fuera" pero por dentro rompen con los principios de una buena programación).

- **Al finalizar** la práctica, a través de una entrevista con los alumnos. La evaluación de práctica no puede reducirse a una simple nota. Para alcanzar plenamente los objetivos formativos, los alumnos deben saber exponer y justificar su solución, y conocer los posibles errores cometidos, teniendo la posibilidad de defender sus decisiones frente al profesor. En consecuencia, la realización de la entrevista es obligatoria para la evaluación de la práctica.

En cualquier caso, no creemos conveniente la propuesta de una única práctica anual, sino de tres prácticas de tamaño medio, con una duración en torno a los dos meses. Estas prácticas están centradas en cada uno de los tres grandes bloques de la asignatura:

- Estructuras de datos.
- Análisis de algoritmos.
- Diseño de algoritmos.

En cuanto a la organización de los alumnos, ya propusimos y justificamos, también en el apartado 4.4.3, la conveniencia de resolver las prácticas en grupos de dos o tres alumnos. La disposición en grupos, aunque sean de pequeño tamaño, permite la práctica del trabajo en equipo, y las habilidades de coordinación, comunicación y reparto de responsabilidades. Además, el apoyo mutuo entre los miembros del grupo tiene sin duda un efecto positivo en la comprensión de la materia.

Es necesario insistir en que el objetivo de las prácticas no es simplemente escribir un programa, sino aplicar adecuadamente los conceptos vistos en la teoría de la asignatura, siguiendo un proceso metódico en la resolución. Por lo tanto, debemos advertir a los alumnos los siguientes aspectos:

- **Necesidad de análisis y diseño.** Con mucha frecuencia, los alumnos cometen el error de empezar la resolución de una práctica directamente escribiendo código en la máquina, sin plantearse antes la solución y a veces, incluso, sin tener claros los mecanismos y sintaxis del lenguaje. La escritura de un programa en un lenguaje concreto debe entenderse como una fase de un proceso que empieza con la obtención de los requisitos –dados en el enunciado–, el estudio y análisis del problema de interés, y la propuesta de una solución "en papel". Creemos que la mejor forma de remarcar la importancia de este punto es requerir que en la memoria de prácticas se incluyan apartados dedicados al análisis, al diseño y a la documentación del desarrollo del proyecto.
- **Usar modularidad y abstracciones.** Relacionado con lo anterior, un buen diseño debería producir un esquema global de la solución, basado en la descomposición modular y en el uso de abtracciones funcionales y de datos. La funcionalidad debe estar bien repartida entre los módulos, de manera más o menos uniforme. Debe estar claro el sentido y la responsabilidad de cada módulo. Por su parte, la implementación de los tipos abstractos debe respetar los principios del concepto de TAD: encapsulación y ocultamiento de la información. Se deben conocer y aplicar los mecanismos del lenguaje más adecuados para cada caso.
- **No cuidar los interfaces.** Es necesario que los alumnos se centren en los conceptos importantes de la asignatura, dejando a un lado aspectos secundarios como la interface de usuario. Es habitual, entre los profesores de primer curso, la queja de que los alumnos entregan prácticas con interfaces muy cuidadas pero usando algoritmos erróneos, mientras que los profesores tienen que

"escarbar entre cientos de líneas de código buscando una de cada seis o siete (o más) que son las que realmente hacen el trabajo pedido", [Montoya'01]. La mejor forma de evitar la distracción de los alumnos en el diseño de interfaces es hacer que el enunciado de la práctica sea muy explícito en este sentido, por ejemplo, indicando de antemano un formato de entrada y salida.

- **Cuidar la eficiencia.** La cuestión de la eficiencia es un tema transversal en toda la asignatura y también en las prácticas. Cuando hablamos de eficiencia no nos referimos a optimizaciones de bajo nivel, sino a las implicaciones que tiene sobre los tipos y estructuras de datos seleccionados, tanto en tiempo de ejecución como en uso de memoria.

8.2.5. Las tutorías

Las tutorías son un complemento indispensable para las restantes actividades del proceso docente. Ofrecen al alumno la posibilidad de recibir una ayuda personalizada en la resolución de dudas o problemas relacionados con la teoría o las prácticas de la asignatura. El profesor puede adaptar sus explicaciones al nivel y ritmo adecuado para cada alumno. Pero, en cuanto complemento que son, no pueden considerarse como un sustituto para la asistencia a clase o a los seminarios. Para un adecuado desarrollo de las tutorías es conveniente que los alumnos aclaren cuáles son sus dudas, revisen los materiales de que disponen y faciliten de alguna manera la resolución de la duda[3].

Los objetivos de las sesiones de tutorías, tal y como se entienden en nuestra Universidad, son los siguientes:

- Resolver dudas puntuales de la materia teórica o práctica.
- Facilitar ejemplos que ilustren y ayuden a comprender los conceptos teóricos.
- Adecuar el método de explicación a cada alumno y a la forma en que éstos puedan comprender mejor los conceptos.
- Recomendar bibliografía complementaria, en el caso en que el alumno desee ampliar sus conocimientos sobre algún aspecto particular.

El Real Decreto 898/1985 sobre el régimen del profesorado universitario fija en su artículo 9.4 que el número de horas de tutoría a la semana de un profesor a tiempo completo debe ser de 6. En todos los casos se procurará que éstos horarios garanticen:

- Su distribución homogénea a lo largo de la semana.
- Su no coincidencia con las horas lectivas de los alumnos a los que están dirigidas.
- El respeto a la igualdad de condiciones de los turnos de mañana y tarde.

Una característica fundamental de las tutorías es que tienen lugar a iniciativa de los alumnos. Esto hace que, en la práctica, las tutorías sean un recurso muy infrautilizado: son pocas las tutorías que se atienden al cabo del año, la mayoría tienen lugar cuando se aproxima el examen o la fecha de entrega de las prácticas, y casi siempre por parte de los mismos alumnos. Básicamente, el profesor no tiene otro medio de promover un mayor uso de las tutorías sino ofrecer un buen servicio: respetar el horario, crear un clima apropiado y atender adecuadamente a los alumnos cuando se deciden a realizar consultas.

[3] No es infrecuente el caso del alumno que pide al profesor que encuentre un fallo en su programa sin ni siquiera llevar el código del mismo, o incluso llevando una versión anterior en la que el programa no fallaba.

En otras universidades españolas el concepto de tutoría adquiere otra dimensión: la orientación al alumno en la elección de su currículum. El profesor se convierte en un asesor de los alumnos, recomendándole las especialidades y asignaturas se adecúan más a su perfil o intereses. En todo caso, el alumno siempre retiene plena la capacidad de decisión pudiendo considerar o ignorar los consejos del profesor. Desde el Decanato de la FIUM se está impulsando la implantación de un sistema de tutorización de este tipo en nuestra Facultad. Los alumnos serían asignados desde primer curso a un profesor, encargado de asesorarle a lo largo de toda su formación. Obviamente, la viabilidad de este método requerirá la implicación y la formación de los profesores.

8.3. Material docente

Los distintos tipos de actividades docentes se apoyan en el uso de medios materiales, que van desde la clásica pizarra hasta los modernos medios electrónicos. La selección de los medios más adecuados y la cuidada preparación de los materiales que van a ser entregados a los alumnos son dos aspectos a tener en cuenta en la planificación de un proyecto docente. En esta sección vamos a hacer una rápida revisión de los tipos de materiales docentes de los que haremos uso en la asignatura AED, para después centrarnos en la bibliografía y material electrónico recomendados para la asignatura.

8.3.1. Tipos de materiales docentes

Son muchos los tipos de materiales docentes que pueden ser utilizados en una asignatura. La adecuación o no de los mismos se debe establecer en función de los objetivos formativos y de las infraestructuras disponibles. En general, es recomendable el uso de una amplia variedad de materiales, ya que ninguno puede cubrir por sí solo todo el espectro de necesidades formativas de una docencia universitaria.

Básicamente, podemos clasificar los materiales docentes en dos tipos: los que son usados en clase para el desarrollo de la explicación, y los que son ofrecidos o entregados a los alumnos. Además, podemos encontrar otros medios de comunicación entre profesor y alumnos basados en Internet, que sin duda merecen un repaso.

Materiales docentes de uso en clase

En cuanto a los medios y materiales usados en clase, ya sea de teoría, de problemas o de prácticas, destacamos los siguientes:

- **Proyector de transparencias.** Las transparencias facilitan la presentación de información en clase, ilustrando gráficamente los conceptos, y ofreciendo la posibilidad de exponer una síntesis integradora de ideas. Para el profesor ofrecen la ventaja de ser una referencia a partir de la cual recordar y desarrollar los contenidos de la materia sobre los que versará la exposición. Pero el uso de este medio requiere el diseño de unas buenas transparencias. Consideramos que unas buenas transparencias de clase deben tener las siguientes características:

  - Esquematizar los contenidos que van a ser presentados. Imprimir una página de un libro en una transparencia para proyectarla en clase[4], es una estrategia nefasta; y más aun si el profesor se dedica simplemente a leerla, y el alumno empieza a pensar que podría haberse ahorrado la asistencia a clase.

[4] No sería el primer caso...

- La transparencia se debe poder leer y entender. Es bastante evidente, aunque muchas veces olvidado. ¿De qué vale una transparencia con una letra tan pequeña que no se puede leer, o con tantos contenidos que el expositor no da tiempo a que sea leída? En nuestra opinión, el tamaño de letra adecuado de una transparencia es aquel que permita su lectura a un lector un poco miope situado en el sitio más lejano del aula.
- Deben ser lo suficientemente significativas. Tampoco es adecuado llegar al extremo opuesto, en el que resulte totalmente inútil el contenido las transparencias. En nuestra opinión, las transparencias deben ser autocontenidas –en la medida de lo posible–, su lectura debe ayudar a un alumno que ha perdido el hilo de la lección, y al mismo tiempo deben dejar un amplio margen para los comentarios y explicaciones adicionales.
- No desviar la atención del objeto de interés. El uso de colores llamativos o dibujos de fondo es muy habitual, pero totalmente contraproducente. Mi consejo personal es: letra negra sobre fondo blanco. El uso de distractores sólo puede estar sometido a los objetivos del tema.
- Es aconsejable que las transparencias sean, en cierto grado, un reflejo de clase. Si no hay nada que lo desaconseje, las transparencias pueden contener las preguntas y los ejercicios propuestos en clase. Las respuestas quedarían fuera.

Las transparencias permiten un ahorro sustancial de tiempo en la presentación de esquemas, algoritmos y representaciones gráficas, frente a tener que pintarlas en la pizarra. Pero es necesario cuidar que esto no suponga un incremento inaceptable del ritmo de clase. Es necesario autocontrolarse.

- **Cañón de vídeo.** Todo lo dicho para las transparencias es aplicable para las presentaciones realizadas usando ordenador y cañón de vídeo. Pero además este formato añade una posibilidad muy interesante: las animaciones. En una materia como AED es posible encontrar muchas aplicaciones posibles para las animaciones; fundamentalmente, la mayor parte de ellas serían mostrar la ejecución de algoritmos o el funcionamiento de algunas estructuras de datos. Las animaciones pueden hacer más rápida la asimilación de los conceptos, ya que "se ve cómo funcionan" los algoritmos o las estructuras. Pero hay que evitar que la animación dé una visión parcial o incorrecta del problema. También hay que eliminar el uso de animaciones que sólo sean distractores.

  Otra ventaja del uso del ordenador y cañón es la posibilidad de mostrar la ejecución de programas. Esto puede ser especialmente interesante en los seminarios de prácticas, aunque también podría usarse en clases de teoría.

- **Pizarra.** El uso de transparencias o cañón de vídeo no impide la utilización de un medio tan arcaico como es la pizarra. Casi nunca falla[5] y su consumo es muy reducido. Además, el uso de la pizarra otorga a la clase de mayor espontaneidad, y nos permite salirnos momentáneamente del camino trazado por las transparencias. Este cambio puede ser útil para evitar la monotonía de usar un solo medio. El uso de la pizarra puede tener varias aplicaciones:

  - Ampliar algún contenido que no está incluido en las transparencias.
  - Mostrar los pasos de ejecución de un algoritmo, pudiendo avanzar poco a poco.
  - Resolver ejercicios, que pueden estar propuestos pero no resueltos en las transparencias.

[5] Está comprobado que la probabilidad de que hayan problemas con un medio es directamente proporcional con el nivel de tecnificación del mismo.

Saber manejar los medios más adecuados en cada momento es una habilidad que da la experiencia en la enseñanza en general, y en la misma asignatura en particular.

Materiales docentes para los alumnos

Los materiales entregados a los alumnos son normalmente documentación escrita, que se puede clasificar en los siguientes tipos:

- **Transparencias de clase.** Las mismas transparencias usadas en clase deben estar disponibles para los alumnos, con la suficiente antelación al desarrollo de la clase. De esta manera los alumnos podrán evitar tener que copiar todo el contenido de la clase y podrán hacer sus propias anotaciones. En caso de usar presentaciones que incluyan animaciones, también deberán darse los documentos electrónicos a los alumnos, para que puedan recordar en casa las animaciones.

- **Texto guía de la asignatura.** Las transparencias de clase son necesariamente incompletas y esquemáticas. Esto se viene a suplir con el texto guía de la asignatura, donde se desarrolla el contenido de los puntos vistos en clase. Además, el texto recoge otra información de interés como los objetivos, ejercicios y las actividades de autoaprendizaje adecuadas para un buen seguimiento del curso. El uso del texto guía y su disponibilidad tanto en horas lectivas como no lectivas libera al alumno de la ardua tarea de tomar apuntes, permitiéndole atender a las explicaciones sin estar preocupado por cuestiones propias de copista. Su concentración y su rendimiento se ven considerablemente favorecidos por este medio didáctico. El alumno tan solo debe apuntar aquellos aspectos que le resulten menos claros a lo largo de la exposición. En la asignatura AED disponemos del texto guía [García'03], [Giménez'03], elaborado por los profesores que imparten la materia en las tres titulaciones de la FIUM.

- **Boletines de ejercicios.** Para cada tema del programa de teoría se entrega a los alumnos un boletín de unos 20 ó 30 ejercicios. En el apéndice C se puede ver un fragmento de uno de estos boletines. Ya comentamos en el apartado 4.4.3 los aspectos más relevantes de estos boletines de ejercicios, pensados para ser resueltos en papel. Algunos de estos ejercicios están resueltos en el texto guía o en exámenes anteriores. Otros serán resueltos en clases de problemas.

- **Exámenes de años anteriores.** Dejar disponibles exámenes de la asignatura de años anteriores es conveniente para evitar el "miedo a lo desconocido", en referencia a cómo será el tipo de examen. En AED tenemos algunos exámenes de años anteriores, algunos resueltos y otros no. Hay que evitar, no obstante, que el examen se convierta en la única obsesión de los alumnos.

- **Guiones de prácticas.** Los guiones de prácticas son un material de uso parecido a las transparencias de clase. La principal diferencia es que el guión de prácticas debe ser más explícito sobre las actividades a realizar por los alumnos. El guión debe contener numerosos ejemplos, ejercicios y referencias, y debe estar diseñado para ser seguido y utilizado de manera autónoma por los alumnos.

- **Bibliografía adicional.** En una asignatura como AED, situada en primer curso, es necesario empezar a concienciar a los alumnos de la importancia de consultar bibliografía y buscar información adicional. Es obvio que la disponibilidad de todos los medios antes mencionados es algo que disuade de la búsqueda de material extra. Aun así, hay que ofrecer referencias de interés y animar a los alumnos a que la consulten.

Medios docentes basados en Internet

Además de los métodos más tradicionales, el desarrollo de la red de redes, Internet, ha provocado la aparición de un nuevo medio de comunicación, rápido, barato, multidireccional, y actualmente disponible para la mayoría de los alumnos. Mientras que en otras titulaciones el uso de estas herramientas puede ser algo más o menos aconsejable, en una carrera de informática se hacen imprescindibles para el desarrollo habitual de la acción docente. La disponibilidad de medios adecuados repercutirá positivamente en la calidad de la enseñanza. Pero es importante que los alumnos sepan usarlos de forma adecuada.

Entre los medios más relevantes para la materia AAED destacamos los siguientes:

- **Correo electrónico.** Este servicio ofrece un medio de comunicación entre el profesor y los alumnos cuya principal ventaja es que permite la atención de aquellos estudiantes que no pueden a asistir a las tutorías durante el horario establecido para las mismas, por distancia geográfica o por incompatibilidad de horarios, por ejemplo. Los alumnos pueden utilizar el correo electrónico para plantear al profesor sus preguntas, dudas o comentarios sobre la materia teórica o práctica de la asignatura, que el profesor contestará posteriormente, normalmente utilizando este mismo medio. Además, el correo electrónico permite el intercambio de ficheros (material bibliográfico, memorias, etc.) entre el profesor y los estudiantes.

  Pero, dicho esto, es conveniente señalar que el correo electrónico no puede ser un sustituto a la atención personal en horas de tutoría. El correo no permite –al menos por ahora– la interactividad de una comunicación cara a cara, que posibilita establecer una conversación y guiar las explicaciones en función de las necesidades y respuestas de cada alumno. El correo electrónico debe usarse para hacer preguntas muy puntuales y claramente definidas, ya que en otro caso es imposible interpretar lo que el alumno quiere.

- **Página web de la asignatura.** La utilización de la web como apoyo a la docencia de una determinada asignatura supone un importante y valioso complemento que conlleva notables beneficios en el desarrollo de la asignatura, especialmente en lo referente al acceso por parte de los estudiantes al material docente preparado por el profesor. De hecho, a través de la página web de la asignatura es posible poner a disposición de los alumnos diversa información y material de carácter docente, como el que indicamos a continuación:

  - Información general acerca de la asignatura: carácter, número de créditos, aulas de teoría y prácticas en las que se imparte, horario, etc.
  - Información acerca del profesor: nombre, ubicación del despacho, dirección electrónica, horario de tutoría.
  - Objetivos de la asignatura.
  - Programas de teoría y de prácticas.
  - Indicaciones acerca de la metodología docente y la evaluación.
  - Bibliografía básica y complementaria recomendada.
  - Anuncios acerca de las fechas de entrega de las prácticas, las convocatorias de exámenes, cambios de horario eventuales, etc.
  - Apuntes de la asignatura, transparencias de clase, boletines de ejercicios, guiones de prácticas, exámenes de años anteriores, material adicional acerca de temas específicos, etc.
  - Enunciados de las prácticas.

- Manuales, tutoriales y documentación diversa relacionada con los productos software empleados en las prácticas.
- Enlaces o información de interés para los alumnos interesados en ampliar conocimientos en la asignatura.

La página web de la asignatura AED es mostrada en el apéndice C.6.

Aunque la mayoría de los estudiantes tienen la posibilidad de acceder a Internet[6], el uso de páginas web no debe sustituir al sistema tradicional de proporcionar el material docente a los alumnos –dejar una copia en papel en el servicio de reprografía–, puesto que es imprescindible garantizar que todos los alumnos tengan acceso a la información y material docente mencionados.

- **La aplicación SUMA.** La aplicación SUMA[7] ha sido desarrollada, en el marco del proyecto con el mismo nombre, por el Área de Tecnologías de la Información y las Comunicaciones Aplicadas (ATICA) de la Universidad de Murcia, en colaboración con empresas y organismos públicos de la región. SUMA es en una aplicación de *campus virtual*, que integra las tecnologías de la información y de las comunicaciones con el propósito de facilitar el acceso remoto a la Universidad de Murcia por parte de alumnos, profesores y personal de administración y servicios. Los servicios que ofrece SUMA permiten la realización remota, vía Internet, de tareas tales como las que indicamos a continuación:

  - Tareas administrativas: automatrícula, consultas de índole académica, inscripciones en cursos, tablones informativos, ventanillas de sugerencias, etc.
  - Tareas extracurriculares: reserva de instalaciones deportivas, reserva de ordenadores en las aulas de libre acceso, citas en el Centro de Medicina Deportiva, bolsas de trabajo, etc.
  - Tareas docentes: acceso a temarios y material docente de asignaturas, tutorías remotas, correo electrónico con fines docentes, aulas virtuales, autoevaluación, participación en foros de discusión, acceso a fondos documentales (bibliotecas), etc.

  De todos estos servicios destacamos los relativos a la posibilidad de realizar autoevaluaciones. Por un lado, los alumnos pueden emplear este servicio para medir su nivel de conocimientos respecto de determinados contenidos docentes, a través de la realización de tests preparados por los profesores. Por otro lado, los docentes pueden detectar las deficiencias y necesidades de formación a través de la observación de los resultados obtenidos por sus alumnos. De este modo, SUMA contribuye a la mejora de la calidad de la docencia, así como a la simplificación de muchas tareas burocráticas, como rellenar las actas de notas.

Si ya mencionamos en el apartado 2.1 (El contexto institucional) que una de las misiones de la Universidad es la extensión de la cultura universitaria a la sociedad, Internet es uno de los mejores medios para lograr la apertura de nuestros centros al exterior. Estos medios permiten un mayor contacto entre distintas universidades y una disponibilidad de información para los países con menos posibilidades[8]. Por lo tanto, somos partidarios de que la información de la asignatura sea accesible, en la medida de lo posible, de manera pública y gratuita.

[6] Según una encuesta realizada el curso 2002/03, disponible en el apéndice B.2, el 78 % disponía de acceso a Internet.

[7] Servicios Universidad de Murcia Abierta, disponible en: `http://www.um.es/atica/suma`

[8] A modo de ejemplo, la página web de la asignatura AED es casi tan visitada por internautas de México como de España. El porcentaje de visitas de otros países de Iberoamérica está sobre el 12 %.

8.3.2. Bibliografía recomendada para la asignatura

Ya hemos visto que el material escrito es uno de los principales medios para llevar a cabo la docencia universitaria. En este apartado vamos a concretar más, haciendo un repaso de la bibliografía más interesante para la preparación y estudio de la asignatura objeto de concurso. Empezaremos en primer lugar con la bibliografía recomendada para los alumnos, para ampliar después con algunos otros libros que pueden resultar útiles de forma puntual y que aparecen como "recomendados para el profesor".

Bibliografía básica para el alumno

Existen muchos textos docentes sobre la materia de algoritmos y estructuras de datos, algunos de ellos muy buenos. Pero un libro recomendable para los alumnos debe tener otras características: ser próximo al enfoque y a los contenidos usados en la asignatura, tener un nivel adecuado para los estudiantes, preferiblemente existir una versión en español, estar disponible con cierta facilidad, etc. A continuación seleccionamos y comentamos los libros que consideramos más interesantes para el desarrollo de la asignatura AED, de cara a los alumnos. Hemos intentado adoptar una posición "minimalista", seleccionando los diez libros que creemos más adecuados.

G. García Mateos, J. Cervera López, N. Marín Pérez, D. Giménez Cánovas.
Algoritmos y Estructuras de Datos. Volumen I: Estructuras de datos.
Editorial Diego Marín, Colección textos-guía, 2003.

[García'03]

Este libro está diseñado específicamente para ser utilizado en AED, de hecho es el texto guía de la asignatura. Por motivos de espacio, el texto fue descompuesto en dos volúmenes. Este es el primero de los dos, dedicado a la parte de estructuras de datos. Obviamente, sigue el mismo acercamiento que usamos en la asignatura, describiendo todos los temas de manera bastante detallada. Por lo tanto, resulta un complemento muy interesante a las transparencias de clase. En primer lugar, aparece un capítulo introductorio, con algunos conceptos y generalidades sobre la materia AED. En el capítulo 2 se tratan las abstracciones y las especificaciones, haciendo énfasis en las algebraicas. Los capítulos 3 y 4 tratan las representaciones eficientes de conjuntos y diccionarios con estructuras lineales y arbóreas, respectivamente. En el capítulo 5 aparecen los grafos, definición, representación, problemas y algoritmos. Además de la descripción de los contenidos, que incluyen numerosos ejemplos, cada capítulo empieza con la lista de objetivos del mismo, y acaba con referencias bibliográficas y ejercicios: resueltos, propuestos y cuestiones de autoevaluación.

Todo el desarrollo teórico es realizado utilizando un pseudolenguaje propio, con palabras clave en español, muy sencillo de entender y de traducir a un programa estilo C/C++ o Pascal. De esta manera se consigue independizar el razonamiento algorítmico de las cuestiones relativas a la sintaxis de un lenguaje. No obstante, de cara a las prácticas de la asignatura, este volumen contiene también en los apéndices un tutorial de C, una introducción a C++ (a nivel básico), y un ejemplo de una práctica típica de estructuras de datos.

D. Giménez Cánovas, J. Cervera López, G. García Mateos, N. Marín Pérez
Algoritmos y Estructuras de Datos. Volumen II: Algoritmos.
Editorial Diego Marín, Colección textos-guía, 2003.

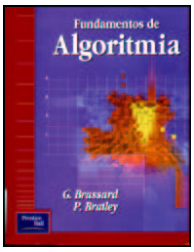
[Giménez'03]

Este es el segundo volumen del texto guía, dedicado a la parte de análisis y diseño de algoritmos. Igual que el anterior, sigue el mismo enfoque que el aplicado en la asignatura, y contiene para cada capítulo: los objetivos, descripción de los temas –usando pseudocódigo–, ejercicios resueltos, propuestos y bibliografía. Los contenidos del bloque de análisis de algoritmos están distribuidos en tres capítulos: el capítulo 6 trata la medición del consumo de recursos, el 7 las notaciones asintóticas, y el 8 las ecuaciones de recurrencia. Los alumnos pueden usar el texto para ampliar las demostraciones o conceptos matemáticos que son vistos superficialmente en clase. Los seis siguientes capítulos describen las técnicas generales de diseño de algoritmos de: divide y vencerás, algoritmos voraces, programación dinámica, backtracking, ramificación y poda, y árboles de juegos. En este punto pueden resultar muy interesantes los ejercicios resueltos, ya que la aplicación de las técnicas es una de las dificultades más frecuentes entre los alumnos. Adicionalmente se incluye un capítulo de complejidad algorítmica, donde se explican algunas ideas básicas sobre complejidad, clases de problemas y reducción de problemas. La explicación es muy breve, aunque en principio no consideramos necesaria su inclusión en la asignatura. Finalmente, el volumen contiene dos apéndices, que muestran dos ejemplos de resolución de las prácticas de análisis y diseño de algoritmos.

A.V. Aho, J.E. Hopcroft, J.D. Ullman.
Estructuras de Datos y Algoritmos.
Addison-Wesley, 1988.

[Aho'88]

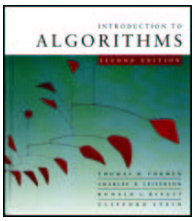
Indudablemente, este libro es uno de los grandes clásicos en la materia y uno de los más referenciados. A pesar de haber pasado ya más de quince años desde su redacción, los contenidos y el modo de abordarlos siguen siendo aún muy válidos. Algunas partes del libro son utilizadas también en la asignatura de programación de primero, por lo que muchos alumnos ya dispondrán del mismo. La descripción de los temas es adecuada, sin llegar a un nivel excesivo de formalización y detalle. De cara a la asignatura AED, el texto resulta especialmente aconsejable para la parte de estructuras de datos. Las tablas de dispersión, estructuras arbóreas y grafos son estudiados a un nivel muy similar al visto en clase. No obstante, hay que señalar algunas lagunas en las que sería conveniente profundizar, como las especificaciones formales, las funciones de dispersión, y los árboles AVL. Por otro lado, en relación a la parte de algoritmos, se pueden utilizar algunos capítulos del libro a modo introductorio. La profundidad con que son tratados resulta algo escasa para lo que es requerido en la asignatura. Por ejemplo, todas las técnicas de diseño de algoritmos están en un sólo capítulo, y no aparece la ramificación y poda.



G. Brassard, P. Bratley.
Fundamentos de algoritmia.
Prentice-Hall, 1997.

[Brassard'97]

Este libro es otro de los clásicos de la materia. Si el anterior resultaba más adecuado para la parte de estructuras de datos, este hace mayor énfasis en la parte de análisis y diseño de algoritmos. Las descripciones son bastante exhaustivas, y los algoritmos son descritos en un pseudocódigo de alto nivel, similar al que utilizamos en clase. En cuanto al análisis de algoritmos, el libro es bastante completo y usa un enfoque riguroso. Aunque no creemos que sea necesario introducir en la asignatura todas las técnicas aquí estudiadas, sí pensamos que los conceptos que sean presentados deben serlo con el rigor matemático adecuado, para lo cual este libro es imprescindible. Las técnicas generales de diseño de algoritmos reciben también una atención especial, existiendo un capítulo específico para cada una. En muchos casos, los ejemplos son los mismos que los tratados en clase. Posiblemente la mayor deficiencia es la técnica de ramificación y poda, que es descrita muy superficialmente. Por otro lado, aunque el libro incluye un capítulo de estructuras de datos, el nivel es claramente insuficiente para lo que necesitamos en clase; tampoco se encuentra información sobre abstracciones y especificaciones.

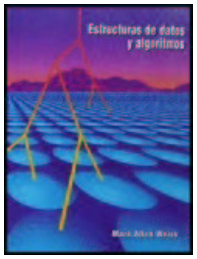


T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein.
Introduction to Algorithms, second edition.
The MIT Press, 2001.

[Cormen'01]

Se trata de un libro bastante exhaustivo, que pretende ser un compendio de todas las técnicas conocidas en el ámbito de los algoritmos y las estructuras de datos. Más que como un libro a seguir como guía, debería usarse para completar aspectos puntuales o ampliar en determinados problemas que se introducen en clase de forma breve. Por ejemplo, puede ser recomendable usar este libro para ampliar en los temas de: tablas de dispersión, árboles B, grafos (a los cuales dedica cinco capítulos), y programación dinámica. Se echan en falta la técnica de ramificación y poda, así como los árboles trie y las especificaciones formales. Por otro lado, creemos que el tratamiento del análisis de algoritmos deja bastante que desear. Aunque se dedican tres capítulos al mismo, el tratamiento que se hace es muy poco riguroso[9], lo que hace que en este punto sea más aconsejable el libro [Brassard'97]. Otra característica del libro, que será un inconveniente para muchos alumnos, es que está escrito en inglés.

[9] Por ejemplo, [Cormen'01] utiliza para las notaciones asintóticas la "igualdad en una sola dirección", que [Brassard'97] desaconseja, por ser algo incoherente que puede llevar a confusiones.



M.A. Weiss.
Estructuras de Datos y Algoritmos.
Addison-Wesley, 1995.

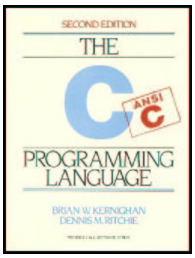
[Weiss'95]

Se trata de una interesante referencia para ser usada como libro de texto, por la sencillez de contenidos y el nivel de profundización, que consideramos adecuado para la asignatura. El libro aborda la mayoría de los conceptos vistos en clase, usando en muchos casos ejemplos similares a los que proponemos. Resulta especialmente adecuado para el estudio de las estrategias de dispersión, los árboles AVL, las relaciones de equivalencia, grafos, y técnicas generales de diseño de algoritmos, excluyendo la ramificación y poda que es omitida. Aunque incluye un capítulo de análisis de algoritmos, creemos que no es nada aconsejable, por la falta de rigor matemático con que se trata; las ecuaciones de recurrencia no reciben la suficiente atención, y se usa la notación *o*-pequeña con un significado completamente distinto al significado estándar. Aún así, el capítulo de diseño de algoritmos puede ser interesante, ya que es detallado aunque no demasiado exhaustivo. Por otro lado –como viene siendo habitual en muchos libros que son traducciones de versiones en inglés–, no se tratan en absoluto los temas relativos a abstracciones y especificaciones, formales o informales.

S. Baase, A.V. Gelder.
Computer Algorithms. Introduction to Design and Analysis, third edition.
Addison-Wesley, 2000.

[Baase'00]

Este libro es la última edición disponible de una serie con una amplia solera en el mundo de los algoritmos y las estructuras de datos. El libro trata la mayor parte de los temas que son estudiados en la asignatura, y en algunos puntos puede resultar interesante para obtener una perspectiva de los problemas algo distinta a la planteada en la mayoría de los restantes libros, por ejemplo en el estudio de algoritmos sobre grafos y la programación dinámica. Quizá lo más interesante del libro, de cara a la asignatura AED, es la introducción a la NP completitud; además de estudiar los conceptos teóricos, se presentan algunos problemas típicos y se proponen métodos para abordarlos de forma heurística. No obstante, pensamos que el libro no resulta aplicable como libro de texto de AED, ya que sigue una aproximación completamente distinta de la materia. El libro se organiza por problemas (ordenación, selección, recorridos en grafos, cierre transitivo, etc.), más que por técnicas generales, como proponemos nosotros. Por otro lado, no existe una traducción del libro al español, lo que supone otro inconveniente para muchos de nuestros estudiantes.



B.W. Kernighan, D.M. Ritchie.
El Lenguaje de Programación C.
Prentice-Hall, 1991.

[Kernighan'91]

Además de los libros de referencia para la teoría de la asignatura, incluimos dos libros de consulta para las prácticas. Este libro es la guía básica de referencia del lenguaje C, escrita por sus propios autores. A pesar de ser un libro algo antiguo, creemos que sigue teniendo validez, ya que es completo pero manejable. El desarrollo de las prácticas se realizará usando el guión de prácticas, y este libro se podrá usar para consultar aquellos aspectos que no queden suficientemente claros. Esto se complementará con el uso de la ayuda que ofrece el propio sistema operativo (en Linux, el manual `man`). El libro contiene numerosos ejemplos típicos y ejercicios, que puede ser interesante plantear en los seminarios correspondientes.

B. Stroustrup.
El Lenguaje de Programación C++.
Addison Wesley, 1998.

[Stroustrup'98]

Este libro es también seleccionado para el apoyo en las prácticas de la asignatura, en concreto en el seminario de C++. Nuevamente elegimos el texto escrito por el propio creador del lenguaje, como el principal manual de consulta. El libro llega a un nivel de descripción y de detalle muy superior a lo que se necesitará en la asignatura. Recordemos que en nuestra propuesta dejábamos de lado todos los conceptos que surgen de la herencia (que son bastantes). De esta forma, el libro no resulta recomendable para usarlo como un guión en las prácticas, sino más bien como una referencia precisa sobre aspectos puntuales que puedan quedar poco claros.

N. Martí, Y. Ortega, A. Verdejo.
Estructuras de Datos y Métodos Algorítmicos: Ejercicios Resueltos; Primera edición.
Prentice Hall, 2004.

[Martí'04]

A pesar de ser un libro de reciente publicación, se trata de una referencia muy interesante, que intenta cubrir uno de los espacios más vacíos en la literatura existente sobre la materia: la resolución de problemas de ejemplo aplicando las técnicas de AED. El libro ha sido escrito por tres profesores de la Universidad Complutense de Madrid, y se divide en dos partes, dedicadas a las estructuras de datos y a las técnicas de diseño de algoritmos. En total el texto contiene más de cien problemas

resueltos, que abordan la mayor parte de los temas tratados en la asignatura AED. Entre los temas contenidos y poco estudiados en otras referencias podemos mencionar las especificaciones formales algebraicas (para las cuales se utiliza una notación similar a la que proponemos) y la técnica de ramificación. Cada capítulo empieza con una breve descripción teórica de algunos conceptos, pero sin duda el núcleo y lo realmente interesante del libro son los problemas resueltos. Para la descripción de los algoritmos se usa un pseudocódigo, siguiendo un acercamiento similar al utilizado en la asignatura AED. Por contra, la principal carencia de este libro es que no se tratan muchas de las estructuras eficientes de representación de conjuntos vistas en clase: tablas de dispersión, árboles trie, y árboles B.

Bibliografía recomendada para el profesor

Obviamente, existen muchos más libros sobre la materia de los algoritmos y las estructuras de datos que los seleccionados en el anterior punto. En general, todos ellos suelen tratar problemas parecidos, por lo que el interés reside en conocer el enfoque que aplican o en los aspectos puntuales en los que se pone mayor énfasis. La siguiente lista no pretende ser una enumeración exhaustiva de todos los libros existentes, sino más bien un repaso de los textos que han sido más influyentes en la preparación de la asignatura AED.

- A.V. Aho, J.E. Hopcroft, J.D. Ullman. *The design and analisys of computer algorithms.* Adison-Wesley, 1974.
- A.V. Aho, J.E. Hopcroft, J.D. Ullman. *Data structures and algorithms.* Adison-Wesley, 1983.
- S. Baase. *Computer algorithms. Introduction to design and analysis.* Addison-Wesley, 1983.
- G. Brassard, P. Bratley. *Algorítmica. Concepción y análisis.* Masson, 1990.
- G. Brassard, P. Bratley. *Fundamentals of algorithmics.* Prentice-Hall, 1996.
- J.C. Laclaustra. *Estructuras de datos y algoritmos.* Colección Textos Docentes, Prensas Universitarias de Zaragoza, 1995.
- M. Collado Machuca, R. Morales Fernández, J.J. Moreno Navarro. *Estructuras de datos, realización en Pascal.* Ediciones Díaz de Santos, 1987.
- T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.
- M.D. Davis, E.J. Weyuker. *Computability, Complexity, and Languages.* Academis Press, 1983.
- H.M. Deitel, P.J. Deitel. *Cómo programar en C++.* Prentice Hall, 2003.
- R.G. Dromey. *How to solve it by computer.* Prentice Hall, 1982.
- A. Drozdek. *Data Structures and Algorithms in Java.* Brooks/Cole, 2001.
- M. Franch Gutiérrez. *Estructuras de datos, Especificación, diseño e implementación.* Ediciones UPC, 1994.
- J. Galve, J.C. González, A. Sánchez, J.A. Velázquez. *Algorítmica, diseño y análisis de algoritmos Funcionales e Imperativos.* Ra-Ma, 1993.

- M.R. Garey, D.S. Johnson. *Computers and intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.
- G.H. Gonnet, R. Baeza-Yates. *Handbook of Algorithms and Data Structures.* Addison Wesley, 1991.
- J. Gonzalo Arroyo, M. Rodríguez Artacho. *Esquemas algorítmicos: enfoque metodológico y problemas resueltos.* Universidad Nacional de Educación a Distancia, 1998.
- D. Harel. *Algorithms. The Spirit of Computing(2nd Edition).* Addison-Wesley, 1992.
- G.L. Heileman. *Estructuras de Datos, Algoritmos y Programación Orientada a Objetos.* McGraw-Hill, 1998.
- R. Hernández, J.C. Lázaro, R. Dormido, S. Ros. *Estructuras de Datos y Algoritmos.* Prentice Hall, 2001.
- J.E. Hopcroft, J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addisson-Wesley, 1979.
- J.E. Hopcroft, R. Motwani, J.D. Ullman. *Introducción a la Teoría de Autómatas, Lenguajes y Computación.* Addisson-Wesley, 2001.
- E. Horowitz, S. Shani. *Fundamentals of Computer Algorithms.* Pitman, 1978.
- E. Horowitz, S. Shani. *Fundamentals of Data Structures.* Computer Science Press, 1982.
- E. Horowitz, S. Shani, D. Mehta. *Fundamentals of Data Structures in C++.* W.H. Freeman & Co., 1995.
- L. Joyanes Aguilar *Programación en C++. Algoritmos, estructuras de datos y objetos.* McGraw-Hill, 2000.
- B.W. Kernighan, D.M. Ritchie. *El lenguage de programación C.* Prentice-Hall, 1991.
- D.E. Knuth. *El arte de programar ordenadores. Vol 1: algoritmos fundamentales.* Reverté, 1985.
- D.E. Knuth. *El arte de programar ordenadores. Vol 3: clasificación y búsqueda.* Reverté, 1987.
- D.E. Knuth. *The Art of Computer Programming. Vol 1: Fundamental Algorithms, Third Edition.* Addison-Wesley, 1997.
- D.C. Kozen. *The Design and Analysis of Algorithms.* Springer, 1991.
- R.L. Kruse. *Estructura de datos y diseño de programas.* Prentice-Hall, 1989.
- Y. Langsam, M.J. Augenstein, A.M. Tenenbaum. *Estructuras de datos con C y C++.* Prentice-Hall, 1997.
- H.R. Lewis, C.H. Papadimitriou. *Elements of the theory of computation.* Prentice-Hall, 1981.
- A. Levitin. *The Design and Analysis of Algorithms.* Addison Wesley, 2003.
- R.C. Linger, H.D. Mills, B.I. Witt. *Structured programming - theory and practice.* Addison Wesley, 1979.

- U. Manber. *Introduction to Algorithms. A Creative Approach.* Addison-Wesley, 1989.
- Mehlhorn. *Data Structures and Algorithms. Vol 1-3.* Springer, 1984.
- J. Nievergelt, K.H. Hinrichs. *Algorithms and Data Structures: with Applications to Graphics and Geometry.* Prentice Hall, 1993.
- R. Peña Marí. *Diseño de Programas. Formalismo y Abstracción.* Prentice-Hall, 1998.
- F. Rabhi, G. Lapalme. *Algorithms, a Functional Programming Approach.* Addison-Wesley, 1999.
- R. Sedgewick. *Algorithms.* Addison-Wesley, 1988.
- R. Sedgewick. *Algorithms in C++.* Addison-Wesley, 1992.
- S.S. Skiena. *The algorithm design manual.* Springer-Verlag, 1998.
- A.M. Tenenbaum, M.J. Augenstein. *Estructuras de datos en Pascal.* Prentice-Hall, 1988.
- J.M. Troya Linero. *Análisis y diseño de algoritmos.* VI Escuela de Verano de Informática. AEIA, 1984.
- M.A. Weiss. *Data structures and algorithm analysis in C.* Benjamin Cummings, 1993.
- M.A. Weiss. *Estructuras de datos en Java.* Addison-Wesley, 2000.
- N. Wirth. *Algoritmos+Estructuras de datos=Programas.* Ediciones del Castillo, 1980.
- N. Wirth. *Algoritmos y estructuras de datos.* Prentice-Hall, 1987.
- D. Wood. *Theory of Computation.* John Wiley & Sons, 1987.

8.3.3. Material en formato electrónico

Hoy en día la gran mayoría de nuestros alumnos tienen acceso a Internet, ya sea en casa o en los ordenadores de la Facultad. El material disponible a través de la red puede resultar un complemento interesante a las referencias bibliográficas; pensamos que nunca un sustituto. Podemos señalar los siguientes enlaces de interés:

Página web de la asignatura AED
`http://dis.um.es/~ginesgm/aed.html`

A través de esta web los alumnos pueden acceder a todo el material repartido en clase, así como al programa de la asignatura, indicaciones sobre la metodología y evaluación de la asignatura, y posibles avisos. El contenido actual de la web se muestra en el apéndice C.

Dictionary of Algorithms and Data Structures
`http://www.nist.gov/dads/`

Una interesante colección de definiciones de problemas, técnicas algorítmicas, estructuras de datos, etc. En total son más de 300 conceptos, entre los más importantes dentro de la materia. Para cada uno de ellos hay una definición, unas notas explicativas y, lo que es más importante, referencias a sitios donde se puede obtener más información o incluso implementaciones.

Free On-Line Dictionary of Computing
`http://foldoc.doc.ic.ac.uk/`

Un proyecto parecido al anterior, aunque no tan centrado en la materia de AED sino más general. Puede ser útil para encontrar algunas definiciones puntuales.

The Stony Brook Algorithm Repository
`http://www.cs.sunysb.edu/~algorith/`

Se trata de una completa colección de implementaciones de algoritmos, entre los más clásicos en la materia. La página es mantenida por el autor de [Skiena'98], como un complemento a su libro. Está centrado sobre todo en problemas de optimización combinatoria, usando diferentes lenguajes (C, C++, Java, Pascal, ADA, etc.). Puede ser útil para probar la ejecución de algoritmos que, por las limitaciones de tiempo, no se pueden incluir en clase.

Sorting and Searching Algorithms
`http://epaperpress.com/sortsearch/`

Parecida a la anterior, aunque más centrada en los problemas de ordenación y búsqueda. No obstante, también contiene técnicas de representación de diccionarios, como árboles B y tablas de dispersión. Se incluye una descripción de las técnicas implementadas.

Enlaces de Visualización de Programas
`http://www-zo.iinf.polsl.gliwice.pl/~jfrancik/aa/links.htm`
Complete Collection of Algorithm Animations
`http://www.cs.hope.edu/~alganim/ccaa/`

Los temas relacionados con visualización de programas y animación de algoritmos son una de las principales áreas emergentes en la docencia de la materia AED. Son la base de los métodos innovadores en la enseñanza de la programación, y deben ser muy tenidos en cuenta de cara al futuro. Estas dos páginas contienen numerosas referencias a proyectos de animación de algoritmos. La mayoría de los trabajos en este sentido se basan en Java y en interfaces web; no obstante, esto no debe requerir el conocimiento de este lenguaje (no siempre). Aunque los dos enlaces que hemos incluido son de los más destacados dentro del ámbito, sin duda la mejor manera de ampliar información y tenerla actualizada es buscar "algorithm animation" en un buscador tipo Google.

ACM International Collegiate Programming Contest
`http://icpc.baylor.edu/icpc/`

Ya hemos comentado la existencia de este concurso, que es considerado como unas olimpiadas de programación, y el interés de fomentar la participación de los alumnos en el mismo. Desde esta página se puede acceder a toda la información disponible, sobre normativas, fechas, fases regionales, etc.

Valladolid Programming Contest Site
`http://acm.uva.es/`

En relación con la referencia anterior, quizá más que la organización del concurso mundial, lo interesante para los alumnos es el mundillo existente alrededor del mismo. Esta es una de las mejores referencias en este sentido, donde se pueden encontrar enunciados de problemas (unos cuantos miles), sugerencias para resolverlos, y un juez on-line. Este juez es una herramienta muy interesante, que recibe programas escritos en C, C++, Pascal o Java, y devuelve respuestas indicando si el programa es correcto o no.

Cómo NO realizar una práctica de programación
`http://www.di.uniovi.es/~cernuda/noprog.html`
Una entretenida página con sugerencias sobre cómo no hacer unas prácticas de programación, en una asignatura como AED. A pesar de ser cosas evidentes[10], el autor acierta en encontrar los fallos más habituales de los alumnos. Se recomiendo su lectura antes de hacer las prácticas.

Se puede encontrar más información a través de los enlaces contenidos en las páginas anteriores, o mediante una simple búsqueda en Internet.

8.4. Evaluación de la asignatura

El diseño del sistema de evaluación desempeña un papel primordial en el proceso de enseñanza-aprendizaje, en cuanto que establece la forma de medir el grado de la consecución de los objetivos propuestos. De acuerdo con [Pérez'97], *"la evaluación del aprendizaje es un proceso sistemático mediante el cual se reconoce información acerca del aprendizaje del estudiante y que permite en primer término mejorar ese aprendizaje y que, en segundo lugar, proporciona al docente elementos para formular un juicio acerca del nivel alcanzado o de la calidad del aprendizaje logrado y de lo que el estudiante es capaz de hacer con ese aprendizaje"*. En consecuencia, existen dos objetos de análisis en la evaluación: el grado de aprendizaje del alumno, y la calidad y adecuación del propio proceso docente. El primero dota al profesor del papel de juez y acreditador del nivel alcanzado por cada uno de sus alumnos de forma individualizada. El segundo constituye el elemento de base para la revisión, actualización y mejora del proceso docente, incluyendo el propio sistema de evaluación.

Existen diversas formas y mecanismos de evaluación del alumnado, desde los exámenes orales hasta las pruebas objetivas, cada uno con sus ventajas e inconvenientes. Pero creemos que la evaluación no debe basarse en un único criterio, sino que el profesor debe recoger y valorar diferentes elementos de juicio sobre los logros de los alumnos. Estos elementos deben referirse al trabajo de los alumnos en las dos vertientes de la asignatura: teoría y prácticas. A continuación vamos a presentar la propuesta de evaluación de la teoría y las prácticas de la asignatura AED, así como la forma de obtener la calificación global. Después trataremos los mecanismos de evaluación de la actividad docente, con el propósito de garantizar la revisión del proyecto docente. Pero antes vamos a hacer algunas consideraciones generales sobre la evaluación en los sistemas de educación superior.

8.4.1. Objetivos y fases de la evaluación

En general, evaluar es *"el procedimiento mediante el cual determinamos el valor o mérito de algo"*, [SFPeval'02]. En el caso de la educación universitaria, implica un proceso mediante el cual recogemos información, aplicamos ciertos criterios de calidad y, por último, elaboramos un juicio sobre el valor o mérito del aprendizaje de cada estudiante. Dicho juicio se explicita oficialmente a través de una calificación: suspenso, aprobado, notable, sobresaliente o matrícula de honor[11].

El principal propósito de la evaluación universitaria es la **función de acreditación**: valorar si los estudiantes poseen o no las habilidades y destrezas precisas para el ejercicio de una profesión o, al menos, en lo referente al ámbito de la materia de que se trate. Pero, además, determinar la calidad de los aprendizajes del estudiante puede tener otros propósitos, como por ejemplo:

[10] Como por ejemplo, "*ignora los enunciados de prácticas*", "*escribe el código directamente sin pensar*", o "*ignora los mensajes de error*". Recordemos que se trata de consejos sobre como *no* hacer las prácticas.

[11] A este respecto, recordemos que la adaptación al EEES implicará la utilización de calificaciones numéricas, de 0 a 10, con un decimal.

- Diagnosticar las dificultades de los estudiantes.
- Valorar métodos de enseñanza.
- Valorar la eficacia de un curso.
- Motivar a los estudiantes hacia el estudio.

No obstante, debemos evitar la obsesión de los alumnos por la evaluación, como única y principal fuente de motivación. En palabras de J. Sarramona, *"no se enseña para evaluar, sino que se evalúa para garantizar la mejor enseñanza"*.

La evaluación del aprendizaje implica, al menos, tres momentos o etapas en el proceso evaluador:

- **Establecer los criterios de calidad.** Los objetivos o criterios de calidad constituyen la referencia básica a la hora de determinar o elaborar un juicio sobre el valor del aprendizaje. De hecho ninguna evaluación tiene sentido si no es en relación con unos criterios establecidos de antemano. En consecuencia, los alumnos deben conocer los objetivos de la asignatura y los criterios con los que serán evaluados.
- **Recoger información.** A la hora de elaborar un juicio sobre el aprendizaje del estudiante necesitamos apoyarnos en evidencias que den razón de ese juicio. Un examen, por ejemplo, es una forma de recoger evidencias, información, sobre determinados aspectos del aprendizaje, aunque no la única. Cuanta más información dispongamos de los alumnos mejor podremos evaluar el grado de conocimientos y destrezas de los alumnos. Pero, obviamente, este factor está limitado por los recursos disponibles por el profesor (fundamentalmente de tiempo).
- **Elaborar un juicio.** Una vez recogida toda la información, el siguiente paso es determinar el valor o mérito de los aprendizajes en relación a los criterios de calidad previamente establecidos; es decir, corregir los exámenes o las prácticas. Esta es una de las tareas menos gratas para los profesores universitarios, primero, por las implicaciones que tiene sobre los alumnos, y, segundo, por el volumen de trabajo que conlleva. Pero dedicar todo el empeño y concentración en lograr una corrección de calidad es una responsabilidad de todo profesor, aunque muchas veces parezca quedar fuera de las iniciativas de calidad de la enseñanza.

En definitiva, hay que plantear las pruebas de evaluación dentro de un marco amplio y acorde con una concepción de la enseñanza centrada en el estudiante. Desde este punto de vista, una buena evaluación debe reunir las siguientes condiciones:

- **Sistemática**, es decir, debe responder a normas conocidas por profesores y alumnos de antemano.
- **Coherente** con los objetivos que se han fijado.
- **Fiable y objetiva**, dentro de las posibilidades reales.
- **Global**, es decir, debe abarcar todos los niveles del conocimiento que se han fijado en los objetivos.
- **Continua**, para facilitar las correcciones necesarias en el aprendizaje, tanto por parte del alumno, como por parte del profesor.
- **Práctica**, de manejo fácil tanto para el profesor como para el alumno.

- **Integridad**, que se pongan de manifiesto una amplia gama de factores que intervienen en el hecho educativo (hechos, conceptos, principios, procedimientos, actitudes, etc.).

Además, los alumnos deben disponer de un sistema de revisión, que les permita conocer cómo han sido evaluados. Es más, el profesor debería animar a los alumnos a revisar el examen como medio de reconocer y corregir sus posibles fallos; o al menos debería hacer públicos tanto los enunciados como las respuestas del examen, y en el caso de las prácticas las correcciones realizadas.

8.4.2. Evaluación de la teoría

En este apartado vamos a desarrollar la propuesta de evaluación para la asignatura AAED, que coincide plenamente con la de AED. Nuestra propuesta se fundamenta básicamente en los exámenes escritos, que en nuestra opinión son el método más fiable y seguro de garantizar la autoría del examen[12], así como el más adecuado para medir el rendimiento individual de cada estudiante. El examen es una prueba escrita del rendimiento del alumno, que puede ser complementada por otro tipo de pruebas, aunque siempre conservando el papel predominante en la evaluación.

Exámenes parciales y final

El amplio contenido teórico de la asignatura AED aconseja la realización de un parcial eliminatorio. Existen básicamente dos posibilidades en cuanto a los exámenes establecidos:

- **Primer parcial + Segundo parcial + Final.** La realización de los dos parciales es independiente. Un alumno que apruebe los dos parciales tendrá superada la asignatura, y no deberá realizar el examen final. Un alumno que suspenda ambos exámenes tendrá la posibilidad de superar la parte teórica de la asignatura si pasa el examen final. En cuanto a los que aprueben un parcial y suspendan otro, existen varias alternativas: (1) considerar o no la media de las notas para aprobar, posiblemente a partir de unos mínimos; (2) hacer el examen completo o sólo la parte que no haya sido superada.

- **Primer parcial + Segundo parcial o Final.** Todos los alumnos tienen la posibilidad de realizar el primer parcial. Aquellos que lo superen podrán realizar el segundo parcial, en el que no entran los contenidos del primero. La calificación de estos alumnos será la media ponderada de ambos parciales, siempre a partir de ciertos mínimos exigibles. Los que no superen el primer parcial deberán enfrentarse directamente al examen final de la asignatura. También existe la posibilidad de dejar que los alumnos que no superan el segundo parcial puedan realizar el examen final, que no tendrá lugar si ambos exámenes coinciden el mismo día.

En el primer caso, los alumnos disponen de más oportunidades para superar la teoría de la asignatura; más aun si los parciales son eliminatorios de cara al examen final. Pero obviamente esto supone una duplicación de las pruebas para los alumnos que no hayan aprobado, y por consiguiente un incremento del trabajo de corrección. En el caso del segundo parcial, las pruebas son realizadas con muy poca diferencia de tiempo. Parece poco lógico que un alumno que ha mostrado un nivel insuficiente en el segundo parcial sea capaz de mejorar mucho una o dos semanas después. El trabajo del estudiante debería desarrollarse a lo largo de todo el curso, y no la semana antes del examen. Por otro lado, de cara a los alumnos que suspenden el primer parcial, las dos posibilidades suponen el mismo esfuerzo de estudio. En ambos casos deben estudiar los contenidos de toda la asignatura. La

[12] Aunque, por desgracia, no al 100 %.

única diferencia es que en el primer caso tendrán dos exámenes seguidos y en el segundo uno sólo con toda la materia.

En conclusión, proponemos que la evaluación teórica de la asignatura AED se organice en torno a dos exámenes parciales y un final, estando dirigido el segundo parcial a los alumnos que hayan superado el primero. Los contenidos y fechas de los parciales serían los siguientes:

- **Primer parcial.** Corresponde a la primera parte de la asignatura, Estructuras de Datos. El examen se realizará durante el mes de febrero, después de los exámenes oficiales de las asignaturas anuales, de común acuerdo entre profesor y alumnos.
- **Segundo parcial.** Corresponde a la parte de Análisis y Diseño de Algoritmos. Será también de carácter eliminatorio, y se deberá realizar antes del examen final o en la misma fecha.

  El examen final incluirá, obviamente, todos los contenidos de la asignatura.

En la evaluación final de la teoría de la asignatura, la primera parte –y por lo tanto el primer parcial– supondrá un 40 % de la nota total, y la segunda parte –evaluada en el segundo parcial– el 60 %. Dentro de esta segunda parte, el bloque de análisis de algoritmos supondrá alrededor de un tercio de la nota, y el diseño de algoritmos los dos tercios restantes. No obstante, aquí la distinción es más imprecisa, ya que el análisis aparece también en el diseño de algoritmos.

La nota total de teoría de la asignatura se obtendría como una media de cada parte, ponderada por los porcentajes establecidos. No obstante, proponemos que se establezcan unos mínimos necesarios para realizar la media. Un umbral razonable, y no muy exigente, es un mínimo de 4 sobre 10. De esta manera se pretende evitar que un alumno que muestre un conocimiento muy deficiente en una parte pueda compensar con otro parcial razonable y unas prácticas de autoría dudosa; o, peor aun, la estrategia de eliminar dos o tres temas de un plumazo, por parte de estudiantes que buscan simplemente el aprobado fácil.

Características del examen

Lógicamente, el tipo de examen planteado para evaluar la materia debe estar sometido al proceso de revisión, consenso y mejora que se debe aplicar a todo el método docente, como veremos más adelante. En principio, las características que proponemos para un examen de la asignatura son las siguientes:

- **Duración.** La duración del examen no debería ser superior a 3 horas y media. Es necesario tener en cuenta el factor humano y la capacidad física y mental de los alumnos. Recomendados que se planifiquen exámenes de entre 2 horas 30 minutos y 3 horas, dejando a los alumnos 30 minutos más que el tiempo que hemos estimado en teoría. En cualquier caso, debemos recordar que la normativa de exámenes de la UM sugiere que la duración no sea mayor que 3 horas y media, y en caso contrario *"habrá que establecer un período de descanso de 15 minutos, salvo que de común acuerdo con los alumnos se decida lo contrario"*.
- **Tipo de preguntas.** Creemos que en una materia como AED no resulta adecuada la evaluación mediante preguntas de tipo test, ni de desarrollo teórico de un tema. Las preguntas deben estar orientadas fundamentalmente al objetivo básico de la asignatura: la resolución de problemas. Su resolución debe requerir la aplicación de técnicas o conceptos vistos en clase. Intentando variar el grado de dificultad de las distintas preguntas, podemos tener problemas que supongan una aplicación más o menos directa de un algoritmo dado, otras que requieran el uso y adaptación de alguna técnica o esquema estándar, y otras que impliquen modelar un problema y diseñar un algoritmo.

- **Enunciado.** El enunciado de las preguntas debe ser conciso, claro y no ambiguo. Pero esto no significa que se deba marcar trivialmente cómo se debe resolver el problema. Los enunciados deberían dejar cierta libertad para que el alumno decida, de todos los conocimientos que posee, cuál o cuáles son de utilidad en el problema dado. Con esto se pretende fomentar la capacidad de decisión y modelado de problemas, más allá de la simple memorización. Puede resultar conveniente, al principio del examen, explicar en voz alta los enunciados o hacer alguna aclaración en algún aspecto en el que podamos prever que hayan dificultades generalizadas. Posteriormente, el profesor debe estar disponible para atender dudas individuales.
- **Dificultad.** Las preguntas deberían tener diferente grado de dificultad, aunque sin llegar a extremos ni en uno ni en otro sentido. En cualquier caso, proponemos que las preguntas más complejas se puedan realizar en no más de 45 minutos. Hay garantizar que las preguntas que sumen el mínimo para aprobar supongan un grado razonable de dominio de la materia.
- **Número de preguntas.** Evidentemente el número de preguntas depende de la complejidad de las mismas. Consideramos que cuatro o cinco preguntas son un número razonable para un parcial. En el caso del examen final, puede ser conveniente añadir más preguntas, aunque necesariamente de menor tamaño.
- **Contenidos.** Las preguntas no deberían centrarse en uno o unos pocos temas, sino que tienen que abarcar el mayor rango posible de aspectos estudiados en la asignatura. Esto no significa necesariamente introducir más preguntas. Por ejemplo, se pueden buscar ejercicios cuya solución requieran aplicar los conocimientos estudiados en diferentes sitios.

Diseñar un buen examen no es una tarea sencilla. Y, sin embargo, un examen bien diseñado sólo puede ser utilizado una vez. Encontrar preguntas que sean relevantes, abordables por los alumnos, y no basadas en la aparición de una "idea feliz", implica un trabajo que se puede dilatar casi una semana –aunque no a dedicación completa, claro–. La evaluación de este tipo de exámenes tampoco es sencilla. Es más o menos fácil discernir las respuestas completamente erróneas o correctas. La dificultad se encuentra principalmente al valorar las respuestas que no están no del todo bien ni mal, que serán la práctica totalidad. En este sentido, algunas fuentes, [SFPeval'02], recomiendan que se corrijan los exámenes pregunta por pregunta, en lugar de hacerlo alumno por alumno, y que se *"agrupen las preguntas corregidas en tres montones: deficientes, suficientes y excelentes"*.

8.4.3. Evaluación de las prácticas

Recordemos que la carga crediticia de las prácticas de la asignatura AAED, del plan de 1996, difiere de la carga de la asignatura AED, del plan de 2002. Vamos a analizar la evaluación de ambas prácticas, centrándonos en la asignatura de los planes actualmente vigentes.

Prácticas en la asignatura AAED

En la asignatura AAED las prácticas tienen una menor importancia relativa. Como ya expusimos, nuestra propuesta de prácticas para este caso consiste en la resolución y entrega de algunos ejercicios "en papel", de entre los entregados en los boletines de cada tema. Pero aunque la puntuación de esta parte tiene poco peso en la nota final, la evaluación de este tipo de prácticas supone importantes beneficios tanto para el profesor como para los alumnos:

- **Para el alumno**, le permite conocer su nivel de conocimientos y detectar sus principales fallos y errores conceptuales antes de enfrentarse al examen. En este aspecto desempeñan un papel

fundamental las correcciones del profesor sobre las prácticas, que deben guiar sobre cómo corregir los posibles errores cometidos. Además, la realización de los ejercicios requiere una preparación previa de los temas, con lo que se evita la estrategia habitual de no estudiar hasta la semana antes o dos semanas antes del examen.

- **Para el profesor**, permite obtener una interesante realimentación del nivel de seguimiento de las clases. Esto puede servir para encontrar las lagunas y errores más frecuentes, que podrán variar entre un curso y el siguiente. La posibilidad de obtener esta información antes del examen permite subsanar estos fallos, haciendo en clase las indicaciones oportunas o haciendo énfasis en los puntos menos claros.

La corrección de unas prácticas de este tipo supone un considerable esfuerzo ya que, como hemos visto, su objetivo principal no es obtener una calificación numérica sin más, sino guiar el aprendizaje de los alumnos. Por ello es fundamental que las prácticas corregidas sean devueltas a los alumnos. Teniendo esto en cuenta, no consideramos que sea necesario realizar una entrevista de prácticas con todos los alumnos.

Prácticas en la asignatura AED

La carga práctica de AED, en los planes actualmente vigentes, supone la mitad de la carga total de la asignatura. Por lo tanto, la realización de las prácticas y su evaluación tienen un peso muy importante en la materia. Recordemos que nuestra propuesta para esta asignatura consistía en realizar diversas actividades prácticas, que podemos clasificar básicamente en tres tipos: seminarios de programación, prácticas de programación y prácticas opcionales.

En cuanto a los **seminarios de programación**, consideramos que no deben ser evaluados de forma directa, sino indirectamente a través del conocimiento de los lenguajes C y C++ demostrado en las restantes actividades prácticas. Sería posible, no obstante, incluir una evaluación de estos seminarios, a través de la entrega de ejercicios o mediante la observación del nivel de seguimiento del seminario por parte de cada alumno. En todo caso, esta calificación tendría más bien el papel de motivar la asistencia y participación en los seminarios; en consecuencia, podría introducirse en caso de detectar una necesidad en este sentido. En principio, teniendo en cuenta la experiencia del presente curso 2003/04, no creemos que sea necesario.

El mayor volumen de trabajo se encuentra en las **prácticas de programación**, para las cuales establecimos tres entregas diferentes, una por cada uno de los grandes bloques de la asignatura: estructuras de datos, análisis de algoritmos y técnicas de diseño de algoritmos. Cada una de estas prácticas debe ser evaluada por separado, obteniendo la nota final de prácticas mediante una media ponderada, siempre que todas las prácticas estén superadas o alcancen un mínimo exigible. Las principales consideraciones en la evaluación de estas prácticas son las siguientes:

- **Elementos a evaluar.** Existen diferentes elementos a tener en cuenta en cada una de las tres prácticas de la asignatura, no limitados exclusivamente a los programas desarrollados por los alumnos. Fundamentalmente podemos señalar:
  - La memoria de prácticas.
  - El análisis del problema y el diseño de la solución, contenidos en la memoria.
  - Los programas realizados.
  - La entrevista con los alumnos.
  - En la práctica de análisis de algoritmos, los métodos usados y resultados obtenidos.

Cada elemento de juicio es evaluado independientemente. Es necesario saber combinar todos ellos para obtener la valoración final de la práctica.

- **Requisitos de las prácticas.** Independientemente de la calificación obtenida, es necesario garantizar que todas las prácticas aprobadas superan unos mínimos exigibles. En algunos casos muy justificados podría relajarse algunos de estos requisitos, pero en general consideramos que como mínimo una práctica debe cumplir:
  - La entrega de la práctica debe ajustarse a los plazos establecidos de antemano.
  - La memoria de prácticas debe contener todos los apartados solicitados.
  - Los programas desarrollados deben compilarse sin errores, por lo menos en las salas de prácticas.
  - Los programas deben obtener los resultados correctos, según el enunciado del problema.
  - Cualquier otro requisito que sea establecido en el enunciado de la práctica.
- **Criterios de valoración.** Ya hemos visto que existen diferentes elementos de juicio, no estando limitados al programa implementado. La calificación de cada uno de ellos debe realizarse en base a sus propios criterios. Los principales criterios de valoración son los siguientes:
  - La adecuación del análisis y el diseño al problema propuesto, incluyendo el modelado del problema y la selección de técnicas y algoritmos.
  - Uso de las abstracciones, como por ejemplo encontrar los TAD más relevantes y programarlos respetando el principio de ocultación de la implementación. Está relacionado en parte con el anterior punto.
  - Definición y uso de módulos, debe estar claro el sentido y la responsabilidad de cada módulo.
  - Uso correcto del lenguaje: código legible, robusto, eficiente, uso de características avanzadas (por ejemplo, genericidad y asertos).
  - En la memoria de prácticas, la adecuada presentación y organización (redacción, ortografía, estructura, comentarios, etc.), aunque lógicamente no debe ser lo más relevante.
  - En la entrevista, la capacidad de expresar las ideas propias con claridad y concisión, la habilidad de responder a las cuestiones planteadas por el profesor y la coordinación del grupo.

  Es conveniente que la calificación final de la práctica no se base en una impresión global sin más, sino que tenga en cuenta de alguna manera –mejor explícita que implícita– la valoración de cada uno de estos criterios. Evidentemente, no todos tienen la misma importancia, ni debe hacerse necesariamente una suma de los valores de puntuación de cada criterio, aunque en principio podría ser un buen método de evaluación.
- **Entrevista de prácticas.** Las entrevistas de prácticas son un elemento más a tener en cuenta en la evaluación. Las entrevistas permiten establecer un diálogo entre profesor y alumnos, con los siguientes beneficios:
  - El alumno tiene la oportunidad de defender su trabajo, practicando habilidades como la expresión oral, capacidad de convicción en una discusión, etc.
  - En ocasiones, es necesario una aclaración sobre algún aspecto de la práctica que, bien el profesor no ha logrado entender, bien el alumno no ha sabido expresar adecuadamente en la memoria.

- La entrevista produce un efecto disuasorio sobre alumnos que pretendan copiar la práctica de otros compañeros, de acuerdo con [Fernández'02].

Además de este papel "consultivo", la entrevista en sí puede usarse en un sentido más "inquisitivo", introduciendo preguntas del tipo: ¿por qué habéis usado determinada técnica o estructura de datos?, ¿qué ventajas o inconvenientes tendría cierta modificación?, ¿cómo solucionar cierta limitación o añadir alguna funcionalidad? No obstante, creemos que en la entrevista no debe primar este papel, más próximo a lo que sería un examen oral.

Sería conveniente que todas las entregas de prácticas tuvieran sus correspondientes entrevistas con los alumnos. Si esto no resulta posible por un exceso de alumnos, debería hacerse al menos una entrevista a cada grupo a lo largo del curso.

- **Calificación en grupos.** Por motivos pedagógicos, propusimos que las prácticas se organizaran en grupos de dos o tres alumnos[13]. Uno de los inconvenientes reconocidos de los trabajos en equipo es determinar y valorar acertadamente la aportación de cada miembro. Podemos adoptar diferentes estrategias: dar una nota distinta a cada miembro o dar la misma nota a todo el grupo. La primera opción requeriría encontrar algún mecanismo adecuado para discernir el trabajo de cada componente, lo cual puede no ser viable. Pero si el trabajo es entendido de modo compartido y colaborativo, todos los alumnos de un grupo son responsables de los resultados del mismo. Además, en grupos de tamaño tan reducido es difícil encontrar casos en los que un miembro se desinhiba del trabajo de prácticas. En consecuencia, proponemos que todos los miembros del grupo reciban la misma nota de prácticas. Introducimos dos elementos para fomentar el trabajo equitativo entre los miembros:

  - En la memoria de prácticas se debe documentar la coordinación y el reparto de tareas, indicando explícitamente las responsabilidades asignadas a cada persona.
  - En la entrevista de prácticas se debe garantizar la participación de todos los componentes, intentando verificar que todos conocen los desarrollos realizados.

Por otro lado, recordemos que la propuesta de prácticas incluía la posibilidad de realizar **prácticas opcionales**. En concreto, sugerimos la realización de dos prácticas: un seminario de especificaciones formales, correspondiente a la primera parte de la asignatura; y una práctica de técnicas avanzadas de diseño, coincidente con la parte de algorítmica. La evaluación de estas prácticas sigue básicamente los mismos parámetros que los definidos en las prácticas de programación, aunque teniendo en cuenta que estas actividades deben suponer una menor carga de trabajo.

En cuanto a la calificación de estas prácticas voluntarias, proponemos que sean convalidables por algunos ejercicios de examen referidos a los mismos temas sobre los que traten estas prácticas. Por ejemplo, el seminario de especificaciones formales convalidaría el ejercicio de examen del tema 1, y la práctica de técnicas avanzadas de diseño daría nota al ejercicio de los temas 11 o 12. En principio, estos ejercicios ponderan aproximadamente alrededor de 2 puntos sobre 10, en los exámenes correspondientes. Las prácticas optativas se puntuarán de 0 a 10; para los alumnos que hayan aprobado (nota mayor o igual a 5) la nota en el ejercicio correspondiente del examen será la proporción de la nota de la práctica optativa, en relación al máximo de ese ejercicio.

Este método de evaluación permite que el alumno configure, siquiera mínimamente, su modelo de calificación en la asignatura. En la práctica, son una gran mayoría los alumnos que encuentran atractiva esta posibilidad, y se animan a trabajar en estas actividades. Por otro lado, desde el punto

[13] El enunciado de las prácticas deberá añadir algún requisito extra para los grupos de tres.

de vista del aprendizaje, la realización de las prácticas optativas garantiza un nivel de habilidades igual o superior que lo que supondría su preparación para el examen. Es más, los temas están elegidos estratégicamente entre los que más dificultades suponen habitualmente para los alumnos[14].

8.4.4. Calificación global de la asignatura

La evaluación global de una asignatura establece el mecanismo mediante el cual se conjuga el espectro de calificaciones, anotaciones y otras consideraciones sobre el aprendizaje de cada alumno para obtener una nota final. Es, por lo tanto, un proceso complejo que supone sintetizar en un simple número el trabajo de un estudiante en la asignatura a lo largo del curso. Y al mismo tiempo será uno de los factores más influyentes para los alumnos, en los ámbitos familiar, económico, sentimental, etc.

La evaluación global de cada estudiante supone la realización de, al menos, dos juicios:

- **Aprobado o suspenso.** El aprobado es una certificación de que un alumno tiene los conocimientos y habilidades mínimos exigidos en la materia y, en lo que le corresponde, es parte de la acreditación del alumno para el ejercicio de la actividad profesional. Determinar si un alumno ha superado la asignatura o no es el primer juicio que debe realizar el profesor. En una asignatura como AED, con una importante carga anual, no basta con utilizar un criterio sumativo, sino que es necesario garantizar unos mínimos en todas las partes fundamentales de la asignatura. No tiene sentido que un alumno compense una ignorancia absoluta en estructuras de datos con un buen conocimiento de algoritmos, o un buen examen teórico con la no realización de las prácticas.

- **En caso de aprobado, con qué nota.** Una vez conseguido el aprobado, se debe establecer la nota de acuerdo con el nivel de conocimientos y habilidades adquiridos por el alumno. En este caso, y ante la necesidad de reducir una entrada n-dimensional a un valor unidimensional, se utilizará una evaluación aditiva mediante una media ponderada o similar. El problema en sí está mal condicionado e implica una inevitable pérdida de información. ¿Hasta qué punto merece la misma nota un alumno que demuestra regularidad en la asignatura que otro mediocre que acierta a dar la respuesta precisa en una pregunta clave del examen?

El primer juicio conduce al establecimiento de mínimos necesarios para aprobar la asignatura. El segundo requiere decidir la ponderación de cada parte en la nota final.

En cuanto a los mínimos exigidos, es necesario encontrar el nivel de exigencia adecuado, evitando los extremos en uno u otro sentido. En concreto, establecemos los siguientes mínimos:

- Para calcular nota media de teoría, cada parcial debe tener un mínimo de 4 sobre 10.

- Para calcular la nota media de prácticas, cada práctica debe tener como mínimo un 4,5 sobre 10. Este mínimo sólo es aplicable en AED. En la asignatura de plan antiguo AAED, donde las prácticas tienen una importancia secundaria, proponemos que no exista ningún mínimo.

- Para calcular la nota total de la asignatura, la nota de teoría debe ser como mínimo de 4,5 sobre 10.

- Para aprobar la asignatura, la nota total debe ser como mínimo de 5 sobre 10.

En cuanto a la ponderación de las diferentes partes, proponemos los siguientes porcentajes siempre que se superen los mínimos establecidos:

[14] Y los que son para muchos estudiantes los principales candidatos a ser descartados del temario, en una nada aconsejable táctica de estudiar lo mínimo para aprobar.

- Nota total de teoría: 40 % primer parcial, y 60 % segundo parcial.
- Nota total de prácticas: 35 % práctica 1, 30 % práctica 2, y 35 % práctica 3.
- Nota total de la asignatura (en AED): 65 % nota total de teoría, y 35 % nota total de prácticas.
- Nota total de la asignatura (en AAED): 80 % nota total de teoría, y 20 % nota total de prácticas.

En definitiva, tenemos una "mapa de notas" en el que cada elemento desempeña su papel en la calificación final. En la figura 8.1 se muestra gráficamente este mapa para la asignatura AED.

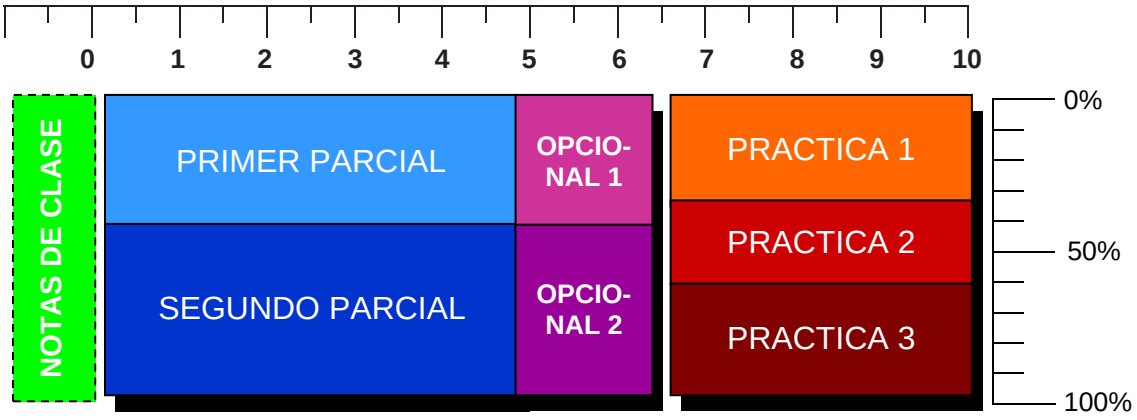


Figura 8.1: Mapa de notas de la asignatura Algoritmos y Estructuras de Datos. "OPCIONAL 1 y 2" hacen referencia a las prácticas optativas, convalidables por ejercicios de examen.

Como ya hemos comentado, las prácticas optativas ofrecen al alumno la posibilidad de configurar su propio formato de evaluación. Para aquellos que hayan realizado la práctica, la nota de la parte "OPCIONAL" corresponderá a la nota de esa práctica. Para los que no la hayan superado se evaluará dentro del parcial correspondiente. Su ponderación dentro del parcial será aproximadamente de un 20 % de cada parcial. De esta manera la parte de prácticas puede suponer entre el 50 % y el 35 % de la asignatura, a elección del alumno.

Por otro lado, añadimos a las nota final de la asignatura la posibilidad de incluir notas de clase. Como ya vimos, el objetivo de esta nota es fomentar la participación de los alumnos en clase, y la realización voluntaria de otro tipo de actividades, como las Olimpiadas Murcianas de Programación. En total, esta nota puede suponer hasta un punto a sumar en la nota final de la asignatura. La valoración es siempre positiva, lo cual en teoría podría suponer una nota máxima de 11 puntos. Su función es matizar la nota final, pudiendo transformar un suspenso dudoso en un aprobado, o un notable alto en un sobresaliente. En cualquier caso, esto no supone la disminución de los mínimos exigidos en cada parte.

8.4.5. Evaluación de la actividad docente

La evaluación en el contexto universitario no sólo se refiere a la calificación del alumnado, sino que incluye también el análisis y reflexión sobre el propio proceso docente. Evaluar la eficacia de la metodología propuesta constituye un aspecto clave para lograr una progresiva mejora en la calidad de la docencia. De esta manera, no podemos concebir el presente proyecto docente como un documento

cerrado e inflexible, sino que las consideraciones realizadas deben estar sujetas al proceso de evaluación, revisión y actualización.

La necesidad de ocuparse de este problema es un hecho que ha preocupado siempre a un importante sector de la comunidad universitaria [Zabalza'99]. Con la promulgación de la LOU, se decidió por fin concretar legalmente esta preocupación. En nuestra opinión, una de las aportaciones más positivas de la nueva ley es la apuesta por la profundización en la cultura de la evaluación institucional, tendente a baremar externamente la calidad de los centros de educación superior. Pero también en un ámbito más reducido resulta vital medir la eficacia docente de cada binomio profesor-asignatura individualmente. Ya el artículo 45.3 de la antigua LRU recogía que las universidades, en el ejercicio de su autonomía, eran las primeras responsables de evaluar la calidad de la docencia impartida por su profesorado.

En este último apartado del proyecto docente nos ocuparemos de las dos caras de la evaluación del proceso docente: la **evaluación externa**, practicada desde agentes ajenos a la propia universidad, y dirigida a medir imparcialmente el rendimiento de la misma; y la **evaluación interna**, más concreta y cercana al ámbito del profesor y la asignatura, utilizada más bien como fuente de información de realimentación dirigida a detectar problemas y proponer mejoras en el día a día de la docencia universitaria.

Evaluación externa

La Agencia Nacional de Evaluación de la Calidad y Acreditación (ANECA) es una de las principales novedades introducidas por la LOU (artículo 31.3), como ya vimos en la sección 2.1. La agencia tiene como función principal la evaluación de las actividades docentes, investigadoras y de gestión de las universidades. Según sus estatutos, sus objetivos se pueden resumir en estos dos:

1. Medir y hacer público el rendimiento de la Educación Superior, mediante acciones de evaluación y otras conducentes a la certificación y acreditación, de acuerdo con procedimientos objetivos y procesos transparentes.

2. Reforzar la transparencia y comparabilidad de nuestro sistema universitario, como medio para promover y garantizar la calidad de las Universidades, así como para el establecimiento de criterios para la rendición de cuentas a la sociedad.

Se pretende, pues, implantar un sistema externo de evaluación objetivo y transparente, con el fin de aumentar los niveles de calidad, competitividad y cooperación entre universidades. Asimismo, se espera que la medida tenga repercusiones en la mejora de la toma de decisiones, tanto de los alumnos a la hora de elegir el centro donde estudiar, como de los profesores y las administraciones públicas a la hora de elaborar sus políticas educativas.

En la actualidad están aún por concretar algunas cuestiones sobre su funcionamiento, pero la mayor parte de los planes de evaluación se encuentran ya en marcha, aunque sea en forma de programas piloto. La acreditación del profesorado en función de su currículum docente e investigador (Programa de Evaluación del Profesorado), que le habilita para poder acceder a determinados puestos docentes, por ejemplo, es una de sus funciones más conocidas que se encuentra ya plenamente operativa.

Pero quizá en este punto lo que más nos importe sea su Programa de Evaluación Institucional. Se trata de un programa dirigido a la realización de estudios y análisis sobre los programas y servicios universitarios –en particular, y sobre todo, las titulaciones impartidas por las distintas universidades españolas– que han pasado previamente el proceso de evaluación interna, o autoevaluación, por iniciativa de la propia Universidad. El objetivo inicial es la definición de una serie de criterios e indicadores

que, con posterioridad, serán empleados en cada una de las actividades de acreditación y certificación de la Agencia, algunas de las cuales ya se encuentran en fase experimental.

En la actualidad la Facultad de Informática de la Universidad de Murcia está inmersa en uno de estos procesos que, como ya hemos comentado, se realizan todavía de forma voluntaria y con intención meramente informativa. La guía metodológica elaborada por la ANECA para llevar a cabo este programa de evaluación establece seis criterios que serán utilizados en la calificación final. Los capítulos que se evaluarán serán, más concretamente, el *programa formativo*, la *organización docente*, los *recursos humanos*, los *recursos materiales*, el *proceso formativo* y los *resultados obtenidos*.

En una primera fase la Facultad creó un comité interno de autoevaluación formado por el equipo decanal (decano y vicedecanos) y una representación del Personal de Administración y Servicios (PAS), del Personal Docente Investigador (PDI) y de los alumnos. Las funciones de este comité consistían en la elaboración de un informe previo sobre el estado actual de los capítulos mencionados, y en la revisión de los criterios e indicadores de calidad definidos por la ANECA. En una segunda fase, que se espera que comience este mismo año 2004, un comité externo visitará la Facultad para emitir un informe proponiendo una serie de recomendaciones y mejoras. La ANECA, finalmente, emitirá el informe final de evaluación teniendo en cuenta todos estos informes previos.

Con la participación voluntaria en este programa, el profesorado de la FIUM muestra claramente su interés por estos temas. Se espera que los datos obtenidos de esta evaluación ayuden a la detección de posibles fuentes de problemas y estimulen el afianzamiento de los puntos fuertes. En definitiva, se trata de colocar a nuestra Facultad en un buen punto de partida para los subsiguientes planes de evaluación que, con carácter más vinculante, serán empleados en la expedición de certificaciones que reconozcan la calidad de los programas y servicios ofrecidos por las distintas instituciones universitarias.

Evaluación interna

Además de la evaluación del alumnado y las posibles auditorías externas, otro de los factores que permiten incrementar la calidad y eficacia de los procesos de enseñanza y aprendizaje es la revisión de la actuación del profesor. Una posible forma de llevar a cabo esta tarea es a través de la realización de encuestas anónimas al alumnado, practicadas en los últimos días del curso lectivo. Si se hacen las preguntas adecuadas a los alumnos, y se garantiza la representatividad de la muestra, se trata de una información valiosa que puede utilizarse para corregir ciertos defectos de la metodología docente empleada.

La Universidad de Murcia también se ha mostrado concienciada en este aspecto. Desde el curso 1988/89 viene funcionando una unidad específica, la Unidad de Calidad, encargada de pasar una encuesta de modo periódico a los alumnos –habitualmente cada dos años por profesor y asignatura–, con el fin de recabar cierta información genérica sobre su opinión acerca de varios aspectos relativos al desarrollo de la docencia. En general, las encuestas se suelen centrar en torno a los siguientes puntos[15]:

- En primer lugar, los **contenidos de la asignatura**: el seguimiento del programa, el nivel de conocimientos adquirido, la coordinación entre teoría y prácticas, etc.
- En segundo, la propia **actuación del profesor**: el interés mostrado por este en la docencia, el dominio de la materia, el trato con los alumnos, etc.
- Finalmente, el **grado de satisfacción global** durante la marcha del curso: carga lectiva, adecuación de los recursos empleados, métodos de evaluación, etc.

[15]Un modelo completo del cuestionario empleado se puede encontrar en la dirección web de la Unidad de Calidad de la Universidad de Murcia: `http://www.um.es/unica/evalprof.htm`

El resultado de estas encuestas se recoge en una serie de informes, que son enviados, respectivamente, a cada profesor, a cada departamento y a cada centro, de forma que cada docente no sólo tiene acceso a sus datos, sino que también puede compararlos con unas estadísticas globales de valoración de sus compañeros de departamento y titulación.

Como complemento a estas encuestas, de carácter quizá demasiado general, el profesor puede pasar a sus alumnos un cuestionario más personalizado, en el que le sea posible concretar más en los contenidos teóricos y prácticos propios de la asignatura, recabar opiniones personales, etc.

Muchas veces este tipo de cuestionarios son criticados por un cierto sector del profesorado, alegando razonamientos tales como la subjetividad del alumno a la hora de contestarlos, la influencia que las notas y la dificultad de la asignatura tienen en el resultado final de las encuestas, la participación en los mismos de alumnos que no acuden a clase regularmente, o, en definitiva, a que muchas veces se emplean más para castigar al profesor exigente que para medir verdaderamente la calidad de su docencia. A pesar de sus problemas, sin embargo, debemos insistir en que los cuestionarios constituyen un elemento muy interesante de reflexión para el profesor, que debe hacer una lectura constructiva de los mismos y emplearlos para extraer conclusiones positivas.

Pero las encuestas a los alumnos no son la única vía de la que dispone el profesor para mejorar la calidad de su docencia. El intercambio de opiniones con los colegas suele revertir también de modo muy positivo en su labor como enseñante. En este sentido, debemos destacar que en nuestro departamento existe un Grupo Docente de Programación, en el que se integran los ocho profesores que imparten clases en la asignatura MTP de primero y en AED de segundo. Este grupo se reúne fundamentalmente cuando se tratan cuestiones de ordenación docente. Creemos que sería muy aconsejable planificar también reuniones, por ejemplo a mitad y al final del curso, para debatir cuestiones relativas a la coordinación de la enseñanza de temas comunes o consecutivos en ambas asignaturas, y no sólo a la distribución de los profesores.

Por otro lado, y más específicamente, creemos que la coordinación entre los cinco profesores involucrados actualmente en la docencia de AED en las tres titulaciones es bastante buena: todas las decisiones sobre la asignatura se toman de modo consensuado, con las consiguientes mejoras en la homogeneidad, calidad y equilibrio de los resultados obtenidos. Fruto de esta estrecha cooperación es la edición del texto guía de la asignatura durante el pasado curso, [García'03] y [Giménez'03]. Estamos convencidos de que la comunicación y el constante intercambio de opiniones resultan extraordinariamente fructíferos para cada miembro del grupo a la hora de llevar a cabo su docencia individual.

Finalmente, hay que mencionar también la importancia del intercambio de opiniones con colegas de otras universidades, aprovechando el contacto con ellos en congresos o reuniones de investigación nacionales o internacionales. En este sentido, naturalmente, es si cabe más recomendable la asistencia a congresos dedicados exclusivamente a la docencia. A nivel nacional, las Jornadas de Enseñanza Universitaria en Informática (JENUI), se han ganado a pulso el puesto de reunión anual más importante para los profesores de la disciplina. Muchas de las ideas expuestas en este proyecto docente han sido extraídas de forma más o menos directa de ponencias que fueron presentadas en este foro. Estar atento a las tendencias que nos muestran estas jornadas respecto a la materia e intentar participar en las mismas, son dos propósitos a tener en cuenta para buscar la mejora de nuestra calidad docente.

Parte I

Apéndices

Apéndice A

Programas de las asignaturas relacionadas

A.1. Metodología y Tecnología de la Programación

CURSO ACADEMICO: 2003-2004
TITULACION: Ingeniería en Informática
CICLO: 1º
CURSO: 1º
PERIODICIDAD: Anual
CARÁCTER: Troncal
CREDITOS: 15 Totales (9 Teóricos + 6 Prácticos)
DEPARTAMENTO: Ingeniería de la Información y las Comunicaciones

PROGRAMA DE TEORÍA

1. Introducción a la Programación
- 1.1. Resolución de Problemas y Algoritmos
- 1.2. Concepto de Programa
- 1.3. Ciclo de Vida del Software
- 1.4. Paradigmas de Programación
  - 1.4.1. Paradigma Imperativo
  - 1.4.2. Paradigma Funcional
  - 1.4.3. Paradigma Orientado a Objetos
- 1.5. Evolución de la Programación
- 1.6. Lenguaje de Especificación en Pseudocódigo (EAP)
- 1.7. Clases de Especificaciones: Condicional y Operacional

2. Principios de Diseño de Algoritmos
- 2.1. Concepto de Abstracción
- 2.2. División del Problema
  - 2.2.1. Programación Modular
  - 2.2.2. Refinamiento Progresivo
  - 2.2.3. Diseño Top-Down
- 2.3. Modelización de los Datos
  - 2.3.1. Objetos, Atributos y Relaciones
- 2.4. Tratamiento Secuencial

3. Elementos Básicos de la Programación Imperativa
- 3.1. Identificadores, Variables, Constantes, Expresiones

- 3.2. Tipos de Datos Simples: Natural, Entero, Real, Carácter, Booleano
- 3.3. Tipos de Datos Estructurados: Enumerado, Subrango, Registro, Vector, String, Secuencia, Fichero
- 3.4. Estructura Secuencial
- 3.5. Estructuras de Control de Flujo: Programación Estructurada
  - 3.5.1. Iterativa
  - 3.5.2. Condicional
  - 3.5.3. Noción de Invariante
- 3.6. Procedimientos y Funciones
- 3.7. Paso de Parámetros
  - 3.7.1. Paso por Valor
  - 3.7.2. Paso por Referencia
- 3.8. Ámbito y Extensión
- 3.9. Gestión de Memoria
  - 3.9.1. Memoria Estática vs. Memoria Dinámica
  - 3.9.2. Concepto de Apuntador
- 3.10. Entrada/Salida

4. Introducción a la Complejidad Algorítmica

- 4.1. Complejidad de Algoritmos
  - 4.1.1. Factores del Tiempo de Ejecución
  - 4.1.2. Casos Peor y Promedio
- 4.2. Notaciones Asintóticas
  - 4.2.1. Notación O y Ω
  - 4.2.2. Funciones de Complejidad más usuales
- 4.3. Calculo del Tiempo de Ejecución de un Algoritmo
  - 4.3.1. Operaciones en Notación Asintótica
  - 4.3.2. Reglas Generales para el Análisis de la Complejidad de Algoritmos

5. Técnicas básicas de Ordenación y Búsqueda

- 5.1. Búsqueda
  - 5.1.1. Búsqueda Secuencial
  - 5.1.2. Búsqueda Binaria no Recursiva
- 5.2. Ordenación
  - 5.2.1. Ordenación por Inserción Directa
  - 5.2.2. Ordenación por Intercambio Directo (Método de la Burbuja)
  - 5.2.3. Ordenación por Selección Directa
- 5.3. Complejidad de los Algoritmos de Ordenación y Búsqueda

6. Recursividad

- 6.1. Definiciones Recursivas
- 6.2. Recursividad Directa e Indirecta
- 6.3. Demostración por Recurrencia
  - 6.3.1. Definición Recurrente de una Función
- 6.4. Árbol de Recursión
- 6.5. Recursión y Compiladores
- 6.6. Recursión vs. Iteración
- 6.7. Esquemas Recursivos:
  - 6.7.1. Backtracking
    - 6.7.1.1. Backtracking por Anchura y por Profundidad
    - 6.7.1.2. Resolución de Problemas Tipo (Monedas, N-reinas...)
  - 6.7.2. Divide y Vencerás
    - 6.7.2.1. Búsqueda Binaria
- 6.8. Análisis de Complejidad de Algoritmos Recursivos

7. Tipos de Datos Abstractos

- 7.1. Concepto y terminología
- 7.2. Clasificación de Tipos de Datos Abstractos

7.3. Especificación de Tipos de Datos Abstractos
7.3.1. Especificaciones informales
7.3.2. Especificaciones formales
7.4. Técnicas de implementación de Tipos de Datos Abstractos
7.4.1. Implementaciones estáticas y dinámicas
7.4.1.1. Asignación dinámica de memoria
7.4.2. Representaciones contiguas y enlazadas
7.5. Ejemplos
8. Listas
8.1. Descripción del TDA Lista
8.2. Especificación del TDA Lista
8.3. Ejemplos de uso
8.4. Implementaciones del TDA Lista
8.4.1. Representaciones contiguas
8.4.2. Representaciones enlazadas
8.4.2.1. Representación con simple enlace
8.4.2.2. Representación con doble enlace
8.4.3. Comparación de las implementaciones
8.5. Otras alternativas en la definición del TDA Lista
8.6. Modalidades de listas
9. Colas
9.1. Descripción del TDA Cola
9.2. Especificación del TDA Cola
9.3. Ejemplos de uso
9.4. Implementaciones del TDA Cola
9.4.1. Implementación basada en el TDA Lista
9.4.2. Implementación con vectores circulares
9.4.3. Implementación con apuntadores
9.4.4. Comparación de las implementaciones
9.5. Modalidades de colas
9.5.1. Dicolas
9.5.2. Colas de Prioridad
10. Pilas
10.1. Descripción del TDA Pila
10.2. Especificación del TDA Pila
10.3. Ejemplos de uso
10.4. Implementaciones del TDA Pila
10.4.1. Implementación basada en el TDA Lista
10.4.2. Implementación con vectores
10.4.3. Implementación con apuntadores
10.4.4. Comparación de las implementaciones
11. Árboles
11.1. Descripción y terminología fundamental
11.2. Especificación del TDA Árbol
11.3. Ejemplos de uso del TDA Árbol
11.4. Implementaciones del TDA Árbol
11.5. Especificación del TDA Árbol Binario
11.6. Ejemplos de uso del TDA Árbol Binario
11.7. Implementaciones del TDA Árbol Binario
11.8. Árboles Parcialmente Ordenados: Colas de Prioridad
11.9. Árboles Binarios de Búsqueda
12. Métodos de Clasificación
12.1. El Método de Shell

12.2. Clasificación Rápida
12.3. Clasificación por Mezcla
12.4. Clasificación por Montículos

PROGRAMA DE PRÁCTICAS

1. Laboratorio Cerrado

1.1. El Entorno de Programación para Modula-2
1.2. Introducción al Lenguaje de Programación Modula-2
1.3. Tipos de Datos Simples y Estructurados
1.4. Estructuras de Control de Flujo
1.5. Procedimientos y Funciones
1.6. Recursividad
1.7. Diseño de Algoritmos y Técnicas de Ordenación
1.8. Implementación de TDAs

2. Laboratorio Abierto

2.1. Realización de un Boletín de Ejercicios en Modula-2.
2.2. Desarrollo, evaluación y prueba de un Proyecto de Programación en Modula-2.

BIBLIOGRAFÍA

Bibliografía Básica

- Martínez, G.; Mateo, A.L; Paredes, S.; Pérez, F.J. Metodología y Tecnología de la Programación. Departamento de Ingeniería de la Información y las Comunicaciones. Facultad de Informática. 2003.
- Jiménez, F.; Sánchez, G. Metodología y Tecnología de la Programación. Implementaciones en Modula-2. Departamento de Ingeniería de la Información y las Comunicaciones. Facultad de Informática. 2003.

Bibliografía Complementaria

- Aho, A.; Hopcroft, J.E.; Ullman, J.D. Estructuras de Datos y Algoritmos. Addison Wesley Iberoamericana. 1988.
- Brassard, J.; Bratley, P. Fundamentos de Algoritmia. Prentice Hall. 1997.
- Calve, J. et al. Algorítmica. Diseño y Análisis de Algoritmos Funcionales e Imperativos. Ra-Ma. 1993.
- Gogesch Micro Systems, Inc. Modula-2 Compiler Reference Manual. Stony Brook Software. 1996.
- Gogesch Micro Systems, Inc. Modula-2 Utilities Reference Manual. Stony Brook Software. 1996.
- Gough, J.; Hynd, J.; Roggenkamp, M. Gardens Point Modula. Language Reference Manual. Programming Language and Systems Group of the Queensland University of Technology. 1992.
- Gough, J.; Hynd, J.; Roggenkamp, M. Gardens Point Modula. Library Definitions Reference Manual. Programming Language and Systems Group of the Queensland University of Technology. 1992.
- Grassmann, W.; Tremblay, J. P.. Matemática Discreta y Lógica. Una perspectiva desde la Ciencia de la Computación. Prentice Hall. 1997.

- Harrison, R. Abstract Data Types in Modula-2. John Wiley & Sons. 1989.
- Roggenkamp, M.; Hynd, J.; Gough, J. Gardens Point Modula. Users Guide Reference Manual. Programming Language and Systems Group of the Queensland University of Technology. 1995
- Weiss, M.A. Estructuras de Datos y Algoritmos. Addison-Wesley Iberoamericana. 1995.
- Welsh, J.; Elder, J. Introducción a Modula-2. Prentice Hall. 1990.
- Wirth, N. Programming in Modula-2. Springer-Verlag. 1982.

A.2. Bases de Datos

CURSO ACADEMICO: 2003-2004
TITULACION: Ingeniería en Informática
CICLO: 1º
CURSO: 3º
PERIODICIDAD: Anual
CARÁCTER: Troncal
CREDITOS: 12 Totales (6 Teóricos + 6 Prácticos)
DEPARTAMENTO: Informática y Sistemas

PROGRAMA DE TEORÍA
I. EL ENFOQUE DE BASES DE DATOS: CONCEPTOS BÁSICOS.
Tema 1. Sistemas de Bases de Datos.
1.1. Bases de datos y sus usuarios.
1.2. Conceptos y arquitectura del sistema de bases de datos.
1.3. Estructura general del sistema de bases de datos.
Tema 2. Proceso de Creación de Bases de Datos.
2.1. Ciclo de vida de un sistema de aplicación de bases de datos.
2.2. Métodos de diseño de bases de datos.
II. BASES DE DATOS RELACIONALES: FUNDAMENTOS Y MÉTODO DE DISEÑO.
Tema 3. Modelo Entidad-Relación.
3.1. Introducción e historia del modelo.
3.2. Conceptos básicos del modelo.
3.3. Extensiones del modelo.
Tema 4. Diseño Conceptual de Bases de Datos.
4.1. Objetivos y etapas del diseño conceptual.
4.2. Análisis de requisitos de datos.
4.3. Diseño del esquema conceptual.
4.4. El enfoque de integración de vistas.
4.5. Características de un buen esquema conceptual de datos.
4.6. Diseño de transacciones.
Tema 5. Modelo Relacional de Datos.
5.1. Presentación y orígenes del modelo relacional.
5.2. Estructura de datos relacional.
5.3. Características generales de integridad de datos: claves.
5.4. Manipulación de datos: lenguajes relacionales.
Tema 6. Integridad en Sistemas de Bases de Datos Relaciones.
6.1. Reglas de integridad: consideraciones generales y componentes.
6.2. Reglas de integridad en SQL-92.

6.3. Comprobación de restricciones.

Tema 7. Diseño Lógico de Bases de Datos.

7.1. Objetivos y etapas del diseño lógico.
7.2. Diseño lógico estándar.
7.3. Diseño lógico específico.

Tema 8. Teoría de la Normalización.

8.1. Motivación.
8.2. Dependencias funcionales.
8.3. Las tres primeras formas normales y la forma normal de Boyce/Codd.
8.5. Enfoques de diseño relacional: Análisis y Síntesis.
8.6. Dependencias multivaloradas y cuarta forma normal.

Tema 9. Estructuras de Almacenamiento y Métodos de Acceso.

9.1. Conceptos generales de organización de ficheros.
9.2. Organización primaria: ficheros no ordenados, secuenciales y dispersos.
9.3. Organización secundaria: ficheros indexados.

Tema 10. Diseño Físico de Bases de Datos.

10.1. Introducción, objetivos y factores que influyen en el diseño físico.
10.2. El proceso de diseño físico.
10.3. Selección y ajuste de la organización de ficheros y estructuras de acceso.
10.4. Ficheros mixtos y desnormalización.

III. ADMINISTRACIÓN DE BASES DE DATOS RELACIONALES.

Tema 11. Seguridad en Sistemas de Bases de Datos.

11.1. El problema de la seguridad.
11.2. Control de acceso.
11.3. Seguridad en SQL-92.
11.4. Otros aspectos de seguridad: auditoría y cifrado.

Tema 12. Procesamiento y Optimización de Consultas.

12.1. Conceptos generales y objetivos del procesamiento y la optimización de consultas.
12.2. Pasos del procesamiento y optimización de una consulta.
12.3. Reglas generales de transformación de expresiones y reglas heurísticas.
12.4. Implementación de operaciones relacionales.

Tema 13. Conceptos de Procesamiento de Transacciones.

13.1. Concepto de transacción.
13.2. Propiedades de una transacción.
13.3. Operaciones de una transacción.
13.4. Estados de una transacción.

Tema 14. Control de la Concurrencia.

14.1. Introducción y problemas de la concurrencia.
14.2. Seriabilidad de los planes de transacciones.
14.3. Técnicas de control de la concurrencia.
14.4. Granularidad de datos.

Tema 15. Recuperación de Fallos.

15.1. Conceptos generales de recuperación.
15.2. Recuperabilidad de los planes de transacciones.
15.3. El proceso de recuperación del fallo de una transacción.
15.4. Técnicas de recuperación de fallos del sistema.

DESCRIPCIÓN DE LAS PRÁCTICAS

Se propondrá una serie de casos de estudio y ejercicios, a través de los cuales los alumnos deberán aplicar los conocimientos y técnicas previamente explicados y discutidos en las clases de teoría.

Programa de prácticas: (el orden de realización de las prácticas no corresponde con el que se muestra)

P. Lenguajes de bases de datos y normalización:

Problemas en el aula de teoría. Intercalados con los temas teóricos correspondientes.

Práctica P1. Consultas en álgebra relacional, cálculo relacional de tuplas y SQL-92.
Práctica P2. Normalización.

D. Diseño de bases de datos:
Práctica D0. Introducción al diseño de esquemas conceptuales.
Práctica D1. Diseño conceptual.
Práctica D2. Diseño lógico.

B. Bases de datos relacionales:
Práctica B1. Consultas en SQL.
Práctica B2. Definición y modificación de datos en SQL.
Práctica B3. Programación en PL/SQL de Oracle.

Tutoriales previos:
Práctica T1. El entorno SQL*Plus de Oracle.
Práctica T2. El lenguaje procedural PL/SQL de Oracle.

Los alumnos deberán entregar informes de realización de las prácticas D1, D2, B1, B2 y B3.

BIBLIOGRAFÍA

a) Bibliografía Básica

- Elmasri, R.; Navathe, S.B. "Fundamentos de Sistemas de Bases de Datos". 3ª edición. Addison-Wesley, Pearson Educación, 2002.
- De Miguel, A.; Piattini, M.; Marcos, E. "Diseño de bases de datos relacionales". RA-MA, 1999.
- Connolly, T.; Begg C.; Strachan, A. "Database Systems: A Practical Approach to Design, Implementation and Management" 2nd edition. Addison-Wesley, 1998.

b) Bibliografía Complementaria (por orden alfabético)

- Atzeni, P.; Ceri, S.; Paraboschi, S; Torlone, R. "Database Systems: Concepts, Languages and Architectures". McGraw Hill, 1999.
- Batini, C.; Ceri, S.; Navathe, S.B. "Diseño conceptual de bases de datos: un enfoque de entidades -interrelaciones". Díaz de Santos, 1994.
- Date, C.J. "Introducción a los sistemas de bases de datos", 7ª edición Prentice Hall. Pearson Educación de México, 2001.
- Date, C.J.; Darwen, H. "A Guide to the SQL Standard". 4th edition. Addison-Wesley, 1996.
- Elmasri, R.; Navathe, S.B. "Sistemas de Bases de Datos: conceptos fundamentales". 2ª ed. Addison-Wesley Iberoamericana, 1997.
- Garcia-Molina, H.; Ullman J.D.; Widom, J. "Database System Implementation". Prentice Hall, 2000.
- Garcia-Molina, H.; Ullman J.D.; Widom, J. "Database Systems. The complete book". Prentice Hall, 2002.
- Hansen, G.W; Hansen J.V. "Diseño y Administración de Bases de Datos". 2ª edición. Prentice Hall, 1997.
- De Miguel, A.; Piattini, M. "Concepción y diseño de bases de datos: del Modelo E/R al modelo relacional". RA-MA, 1993.

- De Miguel, A.; Piattini, M. "Fundamentos y Modelos de bases de datos". 2ª edición. RA-MA, 1999.
- Shasha, D.E. "Database tuning: A Principled Approach". Prentice Hall, 1992.
- Silberschatz, A.; Korth, H.F.; Sudarshan, S. "Fundamentos de Bases de Datos". 3ª edición. Madrid, McGraw-Hill, 1998.
- Silberschatz, A.; Korth, H.F.; Sudarshan, S. "Fundamentos de Bases de Datos". 4ª edición. Madrid, McGraw-Hill, 2002.
- Teorey, T.J. "Database Modeling and Design". 3rd edition. Morgan Kauffman, 1999.
- Ullman, J.D.; Widom, J. "Introducción a los Sistemas de Bases de Datos". Prentice Hall, 1999.

EVALUACIÓN
La evaluación de la **parte teórica** se realizará a través de una prueba escrita al final del curso académico, que consistirá en una combinación de preguntas abiertas (teóricas y problemas) y preguntas de respuesta alternativa (verdadero-falso) y de elección múltiple.
Para la calificación de la **parte práctica** se tendrá en cuenta los informes entregados por los alumnos (prácticas D1, D2, B1, B2 y B3), así como las entrevistas de prácticas realizadas, si han tenido lugar. Para aprobar la parte práctica de la asignatura en una determinada convocatoria, es necesario haber superado todas las prácticas planteadas a lo largo del curso.
Para aprobar la **asignatura** en una determinada convocatoria es necesario superar ambas partes, teórica y práctica, y la puntuación obtenida en cada parte se utilizará para calcular la nota global final. La calificación de la parte superada (ya sea la parte de teoría, la parte práctica en su conjunto, o algunas prácticas concretas) se conservará a lo largo de las tres convocatorias de junio, septiembre y diciembre. En otras palabras, si tras la última de estas convocatorias (diciembre) no se ha conseguido aprobar las dos partes de la asignatura, ambas deberán ser superadas de nuevo a partir de ese momento.

A.3. Programación Orientada a Objetos

CURSO ACADEMICO: 2003-2004
TITULACION: Ingeniería en Informática
CICLO: 1º
CURSO: 3º
PERIODICIDAD: Cuatrimestral
CARÁCTER: Obligatoria
CREDITOS: 6 Totales (3 Teóricos + 3 Prácticos)
DEPARTAMENTO: Informática y Sistemas

OBJETIVOS
Se introduce al alumno en el paradigma de programación orientada a objetos, OO. Los objetivos de la asignatura son:

i) Describir los conceptos que caracterizan al modelo OO

ii) Valorar en que medida las técnicas OO favorecen la calidad del software, analizando sobre todo cómo facilitan la reutilización y extensibilidad,

iii) Contrastar cómo diferentes lenguajes de programación OO (Eiffel, C++, Java y Smalltalk) reflejan los conceptos del paradigma.

iv) Enseñar un lenguaje OO, junto a un entorno de programación.

v) Introducir técnicas de diseño y programación OO

Justificación del programa

La asignatura introduce los conceptos básicos de la programación orientada a objetos: clases, objetos y herencia, mostrando el contraste en la forma en que los cuatro lenguajes de programación OO más extendidos reflejan estos conceptos. Es importante que el alumno no adquiera la visión de la orientación a objetos ofrecida por un determinado lenguaje, sino que comprenda los conceptos de una forma independiente del lenguaje. También se introduce al alumno en el terreno de la construcción de software OO, presentando algunas técnicas de diseño y programación OO que se completarán en cuarto curso en la asignatura de Arquitectura del Software.

En las clases prácticas se presentará un lenguaje de programación OO concreto y se discutirán ejercicios para aplicar los conocimientos adquiridos en teoría. El lenguaje de programación elegido es Java por los motivos que se exponen abajo. Además de estudiar cómo incorpora Java los conceptos que caracterizan la OO, se abordarán los aspectos básicos del lenguaje como: entrada/salida, manejo de cadenas, etc. y otras características avanzadas como son la programación con hilos y el diseño de interfaces gráficas.

Como trabajo práctico el alumno deberá resolver un conjunto de ejercicios de programación y un proyecto de desarrollo de una pequeña aplicación, con el objetivo de llevar al terreno práctico los conceptos explicados y adquirir algunas destrezas relacionadas con la programación OO, así como manejar un entorno de desarrollo.

El lenguaje elegido para la realización de las prácticas debería tener las siguientes buenas propiedades: orientado a objetos puro, reflejar con claridad los conceptos OO, legibilidad, expresividad, pequeño, tipado estáticamente, soporte para la corrección. Además, sería ideal que estuviese muy extendido su uso en las empresas, debería existir un entorno de programación adecuado para la docencia (fácil de aprender y manejar, robusto, requisitos software y hardware mínimos), debería disponerse de abundante y buena documentación y ser de libre distribución. Existen varios lenguajes candidatos, con los cuales se desarrolla la mayor parte de aplicaciones OO: C++, Java, Smalltalk, Eiffel y Delphi (ahora también debemos considerar C#).

En años anteriores hemos usado Smalltalk por acercarse más a ese lenguaje ideal, no optando por Eiffel (muy buenas propiedades, pero entornos que necesitan muchos recursos, no hay versiones de libre distribución y poco extendido), ni C++ (híbrido, muy complicado, poco legible). Al aparecer Java, y a pesar de su rápido éxito, no apostamos por él ya que todavía no era un lenguaje estable, no se disponían de buenos entornos y por la inercia al cambio, ya que entendíamos que Smalltalk permitía cumplir los objetivos y disponíamos de mucha experiencia y documentación. Sin embargo, hace tres cursos nos decidimos por el cambio ya que Java reúne la mayoría de las propiedades del lenguaje ideal, siendo importante la demanda de las empresas de programadores con conocimientos en Java. Además Java es sin duda el lenguaje más utilizado para el desarrollo de aplicaciones web.

PROGRAMA DE TEORÍA

TEMA 1. ORIENTACION A OBJETOS, UNA TECNICA PARA MEJORAR LA CALIDAD DEL SOFTWARE.

1.1. Definición de paradigma de programación. Clasificación
1.2. Algunos comentarios sobre la crisis del software.
1.3. Factores Internos y Externos en la Calidad del Software
1.4. Modularidad.
1.5. Evolución del concepto de módulo en los lenguajes de programación.
1.6. Reutilización del Software: Requisitos para obtener módulos reutilizables.
1.7. Rutinas, Módulos, Sobrecarga y Genericidad como mecanismos de reutilización.
1.8. Problemas con el diseño descendente.
1.9. Diseño orientado a objetos.

TEMA 2. CLASES Y OBJETOS

2.1. Clases
- Estructura
- Dualidad módulo-tipo
- Ocultación de la información.
- Relaciones entre clases: Cliente-Servidor y Herencia
- Visibilidad

2.2. Objetos
2.3. Mensajes
- Sintaxis. Notación Punto
- Semántica

2.4. Semántica referencia versus Semántica almacenamiento
2.5. Creación de Objetos
2.6. Modelo de Ejecución Orientado a Objetos
2.7. Semántica de la Asignación e Igualdad entre objetos
2.8. Genericidad
2.9. Clases y Objetos en Eiffel, C++ y Smalltalk
2.10. Ejemplo: Definición de una clase "Lista Lineal"

TEMA 3. DISEÑO POR CONTRATO: ASERTOS Y EXCEPCIONES

3.1. Clases y Corrección del software: Asertos.
3.2. Asertos en rutinas: Precondiciones y Postcondiciones. Notación especial
3.3. Contrato software.
3.4. Invariante de clase.
3.5. Utilidad de las aserciones.
3.6. Excepciones en Eiffel
3.7. Asertos en Java
3.8. Excepciones en Java

TEMA 4. HERENCIA

4.1. Introducción
4.2. Doble aspecto de la herencia.
4.3. Polimorfismo.
- Tipo estático vs. Tipo Dinámico
- Regla de aplicación de propiedades.
- Compatibilidad de asignación
- Estructuras de datos polimórficas

4.4. Ligadura Dinámica.
4.5. Clases Diferidas
- Clases Parcialmente Diferidas
- Código Genérico
- Herencia, Reutilización y Extensibilidad del software.

4.6. Herencia múltiple.
- Problemas: Colisión de nombres y herencia repetida
- Ejemplos de utilidad
- Resolución de la herencia repetida

4.7. Genericidad Restringida
4.8. Intento de Asignación
4.9. Herencia en Eiffel, C++ y Smalltalk: Estudio Comparativo
- Mecanismos para la adaptación de propiedades
- Herencia múltiple en Eiffel y C++
- Herencia repetida en Eiffel y C++

4.10. Implementación de la herencia
- Herencia Simple en Smalltalk
- Herencia Simple en C++
- Herencia Múltiple en C++

4.11. Herencia y Ocultación de la Información

PROGRAMA DE PRÁCTICAS

El programa de prácticas de la asignatura se desarrollará en varias sesiones de laboratorio. En cada sesión se explicarán los conceptos teóricos y se profundizará en ellos mediante sencillos ejercicios de programación. Para ello se propondrán ejercicios que el alumno deberá realizar de forma individual y entregará en el plazo establecido. Una vez acaben las sesiones de laboratorio se propondrá un proyecto de programación que será realizado en grupos de dos alumnos.

Las sesiones prácticas comenzarán la tercera semana de clase y constarán de unas ocho sesiones de dos horas. A principios de diciembre se propondrá el proyecto final. El plazo para entregar este proyecto acabará el día del examen de teoría en las tres convocatorias.

El lenguaje de programación de las prácticas será Java, utilizando la versión más actual del JDK (Java Development Kit) de Sun. Como entorno de programación se utilizará JBuilder 7 Personal , integrado con el JDK.

Lección 1. El lenguaje Java.
Lección 2. Entorno de programación.
Lección 3. Clases y objetos
Lección 4. Paquetes.
Lección 5. Documentación del código.
Lección 6. Cadenas.
Lección 7. Entrada/Salida.
Lección 8. Clases envolventes.
Lección 9. Herencia.
Lección 10. Genericidad.
Lección 11. Identidad e igualdad.
Lección 12. Asignación y copia.
Lección 13. Excepciones.
Lección 14. Clases abstractas.
Lección 15. Interfaces.
Lección 16. Colecciones.
Lección 17. Serialización.
Lección 18. Clases anidadas.
Lección 19. Modelo de delegación de eventos.
Lección 20. Desarrollo de aplicaciones con interfaces gráficas.

BIBLIOGRAFÍA

Bibliografía de Teoría

- Arnold, K. y Gosling, J., "El lenguaje de programación Java", Addison-Wesley, 3ª Ed., 2001.
- Budd, T.,"An Introduction to Object-Oriented Programming", 3ª ed., Addison-Wesley, 2002.
- B. Liskov y J. Guttag, "Program Development in Java: Abstraction, Specification, and Object-Oriented Design", Addison-Wesley, 2001
- Meyer, B., "Construcción de Software Orientado a Objetos", 2ª edición, Prentice-Hall, 1998.
- Stroustrup, B., "El Lenguaje de Programación C++", Edición Especial, Addison-Wesley, 2000.

Bibliografía de Prácticas

- Arnow, D. y Weiss, G. "Introducción a la programación con Java", Addison-Wesley, 2000.

- Arnold, K., Gosling, J. y Jomes, D. "El lenguaje de Programación Java" , 3ª Edición, Addison-Wesley, 2001.
- J. Bloch, "Effective Java", Addison-Wesley, 2001
- Eckel, B. "Piensa en Java", 2ª Edición, Prentice-Hall, 2002.
- Jaworski, J. "Java 1.2 al descubierto", Prentice-Hall,1998.
- Deitel, H. M. y Deitel, P.J., "Cómo programar en Java", Prentice-Hall, 1998.

EVALUACIÓN
Se realizará un examen teórico de cuestiones teórico-prácticas del mismo tipo que en años anteriores. El alumno dispondrá de una colección de preguntas de otros años. Si un alumno supera el examen teórico (Nota ≥ 5) se mantiene el aprobado en las siguientes convocatorias del curso académico.
No podrán presentar el proyecto de programación los alumnos que no hayan aprobado los ejercicios. Se mantendrá el aprobado en los ejercicios hasta la convocatoria de diciembre. Para la evaluación del proyecto se realizará una entrevista individual con el alumno. Durante esa entrevista también serán evaluados los conceptos del programa de prácticas. Si un alumno supera las prácticas se mantiene el aprobado en las siguientes convocatorias del curso académico.
Para aprobar la asignatura el alumno debe haber superado las prácticas y la teoría. La nota final de la asignatura se calculará como la media ponderada entre la nota de teoría (60 %) y la nota de prácticas (40 %).

A.4. Fundamentos de Ingeniería del Software

CURSO ACADEMICO: 2003-2004
TITULACION: Ingeniería en Informática
CICLO: 1º
CURSO: 3º
PERIODICIDAD: Cuatrimestral
CARÁCTER: Obligatoria
CREDITOS: 9 Totales (6 Teóricos + 3 Prácticos)
DEPARTAMENTO: Informática y Sistemas

VISIÓN GENERAL
Para alcanzar los objetivos de la asignatura comenzaremos mostrando la necesidad de una disciplina como la ingeniería del software, y a continuación estudiaremos brevemente los conceptos fundamentales en sistemas de información, ya que a éstos sistemas será a los que más atención prestaremos en la asignatura. Después dedicaremos mucho tiempo al estudio de los principales procesos del desarrollo de software: análisis, diseño, codificación y prueba, y mantenimiento. A lo largo de la asignatura prestaremos más atención al paradigma estructurado o convencional: el orientado a objetos se estudiará en detalle en segundo ciclo. Nos ocuparemos especialmente del proceso más crítico en el desarrollo: el análisis, presentando distintas técnicas, y centrándonos en el análisis estructurado. A continuación trataremos el proceso de diseño, discutiremos las características generales de una aplicación bien diseñada, y el método de diseño estructurado. Posteriormente estudiaremos el proceso de pruebas del software (no tratamos en esta asignatura la codificación del software). Después de estudiar cada proceso del desarrollo veremos cómo se lleva a la práctica en Métrica 3, que es la metodología

estándar de planificación y desarrollo de sistemas de información de la administración nacional. Una vez que tengamos una visión de los procesos principales del desarrollo, siguiendo una aproximación de abajo a arriba, se presentarán los principales modelos de ciclo de vida del software y algunos métodos de desarrollo. Se finalizará mostrando varios temas de interés en ingeniería del software: mantenimiento del software (incluyendo reingeniería e ingeniería inversa), reutilización, y herramientas CASE (*Computer-Aided Software Engineering*).

PROGRAMA DE TEORÍA

Estructura del temario

El temario de teoría se imparte secuencialmente, con la excepción del Bloque IV, "Métodos de desarrollo", que se va impartiendo gradualmente intercalado con el Bloque II, "El ciclo de desarrollo del software". La idea es enseñar los procesos de Métrica 3 (por ejemplo, "Análisis del Sistema de Información", "Diseño del Sistema de Información", etc.) una vez se ha introducido el conocimiento general acerca de ese proceso en el tema correspondiente del Bloque II.

Bloque I. Introducción.

Tema 1. Introducción a la ingeniería del software.

1.El software.
2.Factores de calidad del software.
3.Problemas en el desarrollo de software.
4.La Ingeniería del Software.
5.Visión general del proceso de la Ingeniería del Software.

Tema 2. Introducción a los sistemas de información.

1.Concepto de sistema
2.Información y datos
3.Sistemas de información
4.Sistemas de información automatizados
5.Sistemas de información empresariales
6.Elementos y estructura de un sistema de información
7.Otros tipos de sistemas de información

Bloque II. El ciclo de desarrollo de software.

Tema 3. Análisis de requisitos.

1.Antes del análisis de requisitos...
2.Actividades generales de análisis de requisitos
3.Técnicas de recogida de la información.
4.Documentos de especificación de requisitos.
5.Análisis estructurado.
6.Casos de uso.
7.Prototipado.

Tema 4. Diseño del software.

1.Modelos de diseño.
2.Diseño estructurado.
3.Métricas de calidad estructural.
4.Heurísticas de diseño.

Tema 5. Prueba del software.

1.Objetivos de la prueba.
2.Importancia de la prueba.
3.Principios de la prueba.
4.El proceso de prueba.
5.Métodos de diseño de casos de prueba.
6.Prueba de interfaces gráficas de usuario.
7.Estrategias de prueba del software.

Bloque III. Modelos de proceso del software.
Tema 6. El proceso software.
1.El proceso software.
2.Estándares en ingeniería del software.
3.Estándares relacionados con el proceso software.
Tema 7. Modelos del ciclo de vida del software.
1.Concepto de modelo del ciclo de vida.
2.Modelos del ciclo de vida
3.Ciclo de vida clásico.
4.Ciclo de vida de construcción de prototipos.
5.Paradigma de programación por transformaciones.
6.Ciclos de vida evolutivos.
7.Técnicas de 4ª generación.
8.Ciclos de vida orientados a objetos.
9.Modelado del proceso software.
Bloque IV. Métodos de desarrollo.
Tema 8. Introducción a los métodos de desarrollo de software.
1.Definición.
2.Beneficios.
3.Adaptación del método.
4.Características deseables.
5.Clasificación.
6.Ejemplos de métodos.
Tema 9. Métrica 3.
1.Objetivos generales del método.
2.Ámbito de aplicación.
3.Alcance del método.
4.Versiones.
5.Objetivos en el desarrollo de la versión 3.
6.Influencias.
7.Modelo de procesos de Métrica 3.
8.Procesos en Métrica 3.
9.Planificación de sistemas de información (PSI).
10.Estudio de viabilidad del sistema (EVS).
11.Análisis del sistema de información (ASI).
12.Diseño del sistema de información (DSI).
13.Construcción del sistema de información (CSI).
14.Implantación y aceptación del sistema (IAS).
15.Mantenimiento de sistemas de información (MSI).
Bloque V. Otras cuestiones en ingeniería del software.
Tema 10. Mantenimiento del software.
1.Tipos de mantenimiento
2.Coste de las actividades de mantenimiento
3.Dificultades del mantenimiento
4.El proceso de mantenimiento en el ciclo de vida.
5.Métodos de mantenimiento del software
6.Mantenibilidad o facilidad de mantenimiento del software.
7.Métricas para mantenibilidad
Tema 11. Reutilización del software.
1.Beneficios de la reutilización
2.Dificultades para la reutilización
3.Assets
4.Niveles de reutilización

5.Modelo de procesos con reutilización
6.Ingeniería de dominios
7.Ingeniería del software basada en componentes
8.Clasificación y recuperación de assets o componentes

Tema 12. Herramientas CASE.

1.Objetivos.
2.Características deseables.
3.Componentes de una herramienta CASE.
4.Taxonomías de CASE.
5.Herramientas CARE.
6.Situación actual.
7.Criterios de selección.

PROGRAMA DE PRÁCTICAS

Las prácticas de la asignatura se organizarán en grupos de dos personas.
Práctica 0: Seminario de System Architect Objetivo: Aprender a manejar los fundamentos de System Architect, la herramienta CASE que vamos a utilizar en prácticas, a través de la introducción de la especificación estructurada de un problema sencillo, que se proporciona ya resuelto.
Práctica 1: Análisis Objetivos: Aplicar Métrica 3 en el análisis de un caso práctico. Dominar los conceptos fundamentales del Análisis Estructurado. Aplicar Análisis Estructurado a la especificación de los requisitos de un problema concreto. Crear un prototipo de interfaz de la aplicación

BIBLIOGRAFÍA

- Piattini, M., et al., Análisis y diseño detallado de Aplicaciones Informáticas de Gestión. 1996: Ra-ma.
- Pressman, R.S., Ingeniería del Software. Un enfoque práctico. (Adaptado por Darrel Ince). 5ª ed. 2001: McGraw-Hill, Interamericana de España S.A.U.
- Yourdon, E., Análisis estructurado moderno. 1993: Prentice-Hall Hispanoamericana.
- MAP, MÉTRICA versión 3. 2001, Madrid: Ministerio de Administraciones Públicas. Secretaría de Estado para la Administración Pública. Consejo Superior de Informática.

EVALUACIÓN

La nota final de la asignatura vendrá dada por la media de las notas de teoría y de prácticas:
- La teoría se pondera al 65 %.
- Las prácticas de laboratorio se ponderan al 35 %.

Para hacer media será necesario aprobar las dos partes de la asignatura. Si no se aprueba totalmente la asignatura, la nota de la parte de teoría o de prácticas aprobada se mantiene hasta la convocatoria de diciembre.

CUESTIONES ADICIONALES

Se asume que el alumno domina la programación en un lenguaje imperativo de tercera generación como Pascal o C, y que posee los conceptos esenciales del diseño de algoritmos y estructuras de datos. El alumno debe estar familiarizado con la creación de modelos conceptuales utilizando diagramas Entidad/Relación, y con los conocimientos sobre modelos lógicos y diseño lógico (que se obtienen en la asignatura "Bases de Datos", que se imparte al mismo tiempo que FIS).

A.5. Algoritmos y Programación Paralela

CURSO ACADEMICO: 2003-2004
TITULACION: Ingeniería en Informática
CICLO: 2º
CURSO: 2º
PERIODICIDAD: Cuatrimestral
CARÁCTER: Optativa
CREDITOS: 6 Totales (3 Teóricos + 3 Prácticos)
DEPARTAMENTO: Informática y Sistemas

PROGRAMA DE TEORÍA

1. Introducción a la NP-Completitud (dos horas)
Ineficiencia e Intratabilidad
No Computablidad e Indecibilidad
2. Algoritmos Probabilistas (seis horas)
Numéricos
Algoritmos de Sherwood
Algoritmos de las Vegas
Algoritmos de Monte Carlo
3. Introducción a los Algoritmos Genéticos (dos horas)
Esquema básico
Operadores genéticos
Fundamentos de los algoritmos genéticos
Ejemplos
4. Cálculo matricial (cinco horas)
Definiciones y propiedades
Transposición de matrices
Multiplicación de matrices
Eliminación Gaussiana
5. Modelos de programación paralela (cuatro horas)
Modelos computacionales
Programación en memoria compartida: OpenMP
Programación por paso de mensajes: MPI
Paradigmas de programación paralela
6. Análisis de algoritmos paralelos (tres horas)
Tiempo de ejecución, speed-up, eficiencia y coste
Estudio de la escalabilidad
Estudios experimentales
7. Algoritmos paralelos (ocho horas)
Algoritmos matriciales
Ordenación
Algoritmos genéticos
Algoritmos sobre grafos
Algoritmos de búsqueda
Programación dinámica

PROGRAMA DE PRÁCTICAS

1. Exposición en clase de algún problema relacionado con el tema uno.

2. Realización del estudió e implementación de dos algoritmos relacionados con los temas dos, tres

y cuatro.

3. Realización de problemas sobre diseño y análisis de algoritmos.
4. Resolución en OpenMP y MPI de un problema, incluyendo estudio teórico y experimental y comparación de los dos estudios.

BIBLIOGRAFÍA
Bibliografía Básica

- Algorithmics. The Spirit of computing (Parte III). David Harel. Addison-Wesley. 1996
- Fundamentos de algoritmia (Capítulo 12). G.Brassard. P. Bratley. Prentice Hall. 1997
- Algorítmica; concepción y análisis (Capítulo 8). Brassard Bratley. Masson. 1991
- Fundamentos de algoritmia (Capítulo 10). G.Brassard. P. Bratley. Prentice. 199)
- Algoritmos genéticos. John H. Holland. Investigación y ciencia, 1992.Pag 38-45.
- Algoritmos genéticos: una estrategia para la búsqueda y la optimización. Francisco J. Marín, Francisco Gracía y Francisco Sandoval. Informática y Automatica, VOL 25-3,4/1992
- Genetic Algorithms + Data Structures = Evolution Programs. Zbiniew Michalewics. Springer-Verlag. 1992
- Matrix Computations. Gene H.Golub, Charles F.Van Loan. The Johns University Peress. 1989
- Matrix Algorithms.Volume I: Basic Decompositions. G.W. Stewart. SIAM. 1998
- Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. McGraw-Hill. 1996
- Ian Foster: Designing and Building Parallel Programs. 1995
- Kumar, Grama, Gupta, Karypis: Introduction to Parallel Computing. Design and Analysis of Algorithms. The Benjamin Cumming Publishing Company. 1994
- Andrews: Foundations of Multithreaded, Parallel, and Distributed Programming. Addison-Wesley. 2000
- Rajkumar Buyya (Editor): High Performance Cluster Computing, Vol 2, Programming and Applications. Prentice Hall. 1999
- Barry Wilkinson, Michael Allen: Parallel programming. Prentice-Hall. 1999

EVALUACIÓN
La asignatura se evaluará sobre los trabajos y prácticas realizadas por los alumnos.

Apéndice B

Tests de evaluación inicial

B.1. Resultados del test de evaluación inicial, curso 2001/02

Test realizado el día 3 de octubre de 2001, sobre un total de 59 alumnos (51 % de los alumnos matriculados). Existen 2 tipos de preguntas. Las 22 primeras son de evaluación de los conocimientos de los alumnos. Las 4 siguientes son de opinión sobre diversos aspectos de la asignatura. Para cada una de las primeras se muestra entre paréntesis el porcentaje de respuestas correctas; para cada respuesta se indica el número de contestaciones, y en negrita se señala la correcta. Para las preguntas de opinión se muestran los porcentajes de cada opción.

1. ¿Qué es un tipo abstracto de datos (TAD)? (68 %)

   40 **a) Básicamente TAD está compuesto por un dominio y un conjunto de operaciones sobre el mismo.**

   1 b) Un TAD es un concepto teórico pero que no tiene una aplicación directa en programación.

   5 c) Lo anterior es falso, puesto que los TAD dan lugar al concepto de tipo de datos.

   8 d) Un TAD es la implementación de una estructura de datos concreta.

2. Ordena las etapas del proceso clásico de desarrollo de aplicaciones, por el orden en que son realizadas. (31 %)

   20 a) Análisis, diseño, especificación, implementación, pruebas, depuración y compilación.

   18 **b) Especificación, análisis, diseño, implementación, depuración y pruebas.**

   7 c) Especificación, implementación, diseño, pruebas y depuración.

   11 d) Análisis, diseño, implementación, depuración, compilación y linkado.

3. ¿Qué podemos decir respecto a la implementación usando estructuras de datos dinámicas o estáticas? (34 %)

   3 a) La implementación dinámica es siempre más eficiente y usa menos memoria, aunque es más compleja de implementar.

   20 **b) Si conocemos a priori el tamaño que tendrán los datos almacenados, la implementación estática suele ser más adecuada.**

   34 c) Las dos respuestas anteriores son correctas.

1 d) Ninguna de las respuestas es correcta, incluida esta misma.

4. Un tipo de datos mutable: (32 %)

7 a) Es aquel cuyo valor está indeterminado, porque puede cambiar una vez creado.

2 b) Es más útil que uno no mutable, porque su valor no es fijo.

19 **c) Tiene operaciones para cambiar el valor de una variable de ese tipo.**

28 d) Todas las respuestas anteriores son válidas.

5. ¿Cuál de los siguientes tipos de datos no se puede considerar como un tipo contenedor? (37 %)

3 a) Pila de pilas.

13 b) Array de booleanos.

22 **c) Registro de enteros.**

8 d) Árbol binario.

6. Dado el siguiente árbol binario de búsqueda, ¿dónde deberían ser colocados los elementos 6 y 14, respectivamente? (42 %)

9 a) A la izquierda del 9 y a la derecha del 11.

6 b) A la derecha del 3 y a la derecha del 12.

9 c) Debajo del 9 y entre el 16 y el 12.

25 **d) En el subárbol izquierdo del 10 y en el subárbol derecho del 10.**

7. De la programación orientada a objetos podemos decir: (46 %)

4 a) El polimorfismo y la ligadura dinámica son dos conceptos que surgen a raíz de la herencia.

5 b) La ligadura dinámica es un concepto independiente de la herencia, puesto que podría darse aunque no la hubiera.

13 c) Una clase puede tener varias superclases y hereda los métodos de ellas, aunque los puede redefinir.

27 **d) Las afirmaciones a) y c) son correctas.**

8. ¿Cuál es el resultado del siguiente algoritmo recursivo, para la llamada Rec(4)? (46 %)

```
function Rec (i: integer) : integer;
begin
   if i<2 then return i+1
   else return 3 + 2*Rec(i-1) + Rec(i-2) + Rec(i-2);
end;
```

27 **a) 71**

1 b) 25

1 c) 83

24 d) Ninguno de las anteriores.

9. Dada la siguiente estructura de clases, en Java: (17 %)

```
class A {
    public int m1 () {
        return 2;
    }
}
class B extends A {
    public A a1;
    public int m1 () {
        return a1.m1()+1;
    }
}
class C {
    static public int m ( ) {
        B ob= new B();
        ob.a1= ob;
        A oa= ob;
        return oa.m1();
    }
}
```

Qué valor devuelve la ejecución del mensaje: C.m()

19 a) 2
14 b) 3
4 c) 1
10 **d) Ninguna de las anteriores. En realidad, no devuelve ningún valor.**

10. En la pregunta anterior, cambiamos la implementación de m1 en la clase B por: (31 %)

```
return super.m1()+1;
```

¿Cuál sería ahora el resultado?

16 a) 2
18 **b) 3**
2 c) 1
7 d) Ninguna de las anteriores. En realidad, no devuelve ningún valor.

11. ¿Qué errores, entre los otros, provocaría la siguiente definición de clases en Java? (24 %)

```
class C1 {
    private int at1;
    protected double at2;
    public final int f () {...};
}
class C2 extends C1 {
    private int at3;
    public int f () {...};
}
class C3 extends C2 {
    public void g ( ) {
        at1= 5;
        at2= 1.2;
    }
}
```

12 a) La clase C3 no puede acceder a la propiedad protegida at2.

9 b) La clase C2 no puede redefinir el método final f.

14 **c) Además de la respuesta anterior, la clase C3 no puede acceder a la propiedad privada at1.**

11 d) Todas las respuestas son correctas.

12. Calcula el resultado de la siguiente expresión: (15 %)

$3\sum_{i=a}^{b} c^i$

1 a) $3(cc^a - c^b)/(c-1)$

9 **b)** $(3c^{b+1} - 3c^a)/(c-1)$

3 c) $6(c^{a+b} - c^a)/c$

18 d) Ninguno de los anteriores.

13. ¿Cuál es el resultado de la siguiente expresión: $lim_{n\to\infty}(n^2-4)/(10n\log_2 n)$? (64 %)

5 a) 1/10

5 b) 0

38 **c)** ∞

2 d) $n/(10log_2 n)$

14. ¿Cuál de las siguientes expresiones con órdenes de complejidad es correcta? (3 %)

34 a) $2n^2 - n + 7 > O(n^2)$

2 **b)** $(n+1)! \in \Omega(n!)$

10 c) $O(n \log n) \subset O(10n \log n)$

0 d) Ninguna de las anteriores.

15. Respecto a los órdenes de complejidad podemos decir que: (34 %)

6 a) Si el orden de complejidad de un algoritmo A es mayor que el de otro B, entonces el algoritmo A siempre tardará más tiempo.

20 **b) Lo anterior sólo es cierto asintóticamente, es decir para tamaños de entrada suficientemente grandes.**

6 c) $\mathrm{O}(10n) = \mathrm{O}(n) \subset \mathrm{O}(n^2) \subset \mathrm{O}(2^n) = \mathrm{O}(2^{10n})$

15 d) Las respuestas b) y c) son correctas.

16. Un algoritmo que usa backtracking recursivo, genera en cada nivel un máximo de a posibilidades y un mínimo de b. La solución del problema está a nivel n. ¿Cuál de los siguientes es un orden de complejidad válido del algoritmo? (3 %)

2 **a)** $O((a+b)n)$

18 b) $\mathrm{O}(n^{max(a,b)})$

7 c) $\mathrm{O}(n^{a*b})$

6 d) Ninguno de los anteriores, puesto que el orden es impredecible.

17. Cualquier algoritmo recursivo puede transformarse en un algoritmo iterativo, además: (51 %)

11 a) La versión iterativa será más eficiente.

5 b) La versión recursiva será más eficiente.

10 c) Ambas versiones son iguales, pero la recursiva usará más memoria.

30 **d) No se puede saber a priori cual será más eficiente.**

18. Dados los algoritmos de ordenación *mergesort*, *quicksort* y *burbuja*: (41 %)

24 **a) Los dos primeros usan divide y vencerás, el tercero no.**

4 b) Los tres son algoritmos de divide y vencerás.

6 c) Sólo *mergesort* se puede considerar como un algoritmo divide y vencerás.

7 d) Los dos últimos usan divide y vencerás.

19. La técnica de backtracking: (12 %)

1 a) Genera un árbol binario, donde se buscan las soluciones a cierto problema.

10 b) Es una técnica recursiva de diseño de algoritmos, aplicable a muchos tipos de problemas.

7 **c) Se basa en la exploración exhaustiva del espacio de soluciones, que implícitamente tiene forma de árbol.**

37 d) Todas las anteriores son correctas.

20. ¿Cuáles son los pasos básicos de un algoritmo que aplique divide y vencerás? (69 %)

7 a) Dividir el array por la mitad, llamar recursivamente a cada parte y combinar las soluciones parciales.

41 **b) Descomponer el problema, resolver los subproblemas hasta llegar a un caso base, combinar las soluciones de los subproblemas.**

3 c) Aplicar el caso base, para tamaño 1, y para tamaño mayor llamar recursivamente al mismo algoritmo.

4 d) Divide y vencerás se puede implementar de forma no recursiva, aunque lo normal es usar una versión recursiva.

21. ¿Cuál de las siguientes afirmaciones es cierta? (78 %)

2 a) Todo programa de tamaño finito termina en un tiempo finito.

7 b) Cualquier programa de tamaño infinito tardaría un tiempo infinito en ejecutarse.

4 c) Ambas afirmaciones son correctas.

46 **d) Ninguna de las anteriores respuestas es correcta.**

22. Decir cuál de las afirmaciones siguientes es cierta, siendo las demás falsas. (36 %)

23 a) Esta afirmación es cierta si la b) no lo es.

21 **b) La respuesta a) o la c) son falsas.**

4 c) La certeza de esta afirmación implica que la b) también lo sea.

7 d) Todas las anteriores respuestas son correctas.

23. ¿Cuál de los siguientes lenguajes preferirías que se usara en la asignatura de Algoritmos Estructuras de Datos, en las explicaciones de teoría?

19 % a) Pascal / Módula.

64 % b) Java.

10 % c) C / C++.

7 % d) Me da lo mismo.

24. ¿De qué tipo te gustaría que fueran las prácticas de la asignatura?

22 % a) Prácticas de desarrollo de ejercicios en papel.

0 % b) Prácticas de desarrollo de un ejercicio de tamaño medio/grande en papel.

53 % c) Prácticas de programación de ejercicios cortos.

22 % d) Prácticas de programación de un problema de tamaño medio/grande.

2 % NS/NC

25. En caso de hacer prácticas de programación, ¿qué lenguaje preferirías que se usara?

20 % a) Pascal / Módula.

68 % b) Java.

10 % c) C / C++.

2 % d) Otro.

26. Aproximadamente, ¿cuántas líneas de código ocupa el programa más largo que has escrito hasta la fecha (sin contar programas en ensamblador)?

32 % a) 100 líneas o menos (2 páginas en papel).
49 % b) Entre 100 y 400 líneas (entre 2 y 8 páginas).
15 % c) Entre 400 y 800 líneas (entre 8 y 16 páginas).
3 % d) Más de 800 líneas (más de 16 páginas).

B.2. Resultados de la encuesta inicial, curso 2002/03

A diferencia de la anterior, esta encuesta se refiere únicamente a la situación y preferencias de los alumnos, pero sin evaluar de forma directa sus conocimientos. La encuesta fue realizada el día 2 de octubre de 2002, sobre una población de 73 alumnos (49 % de los matriculados), durante la primera clase de la asignatura "Ampliación de Algoritmos y Estructura de Datos". Se muestra, para cada respuesta, el porcentaje de alumnos que la contestaron.

1. ¿Cuál de los siguientes lenguajes preferirías que se usara en las prácticas de la asignatura Algoritmos Estructuras de Datos (asignatura de 2º del plan nuevo)?

49 % Java.
23 % Pascal.
19 % C.
7 % C++.
1 % Módula-2.
0 % Visual Basic.
0 % Otro.

2. ¿Cuánto consideras que es tu grado de conocimiento de los lenguajes Pascal/Módula?

0 % Alto.
11 % Medio-alto.
51 % Medio.
32 % Bajo.
7 % Nulo.

3. Responde la misma cuestión anterior para el lenguaje Java.

1 % Alto.
23 % Medio-alto.
44 % Medio.
30 % Bajo.

1 % Nulo.

4. Responde la misma cuestión anterior para el lenguaje C.

5 % Alto.
5 % Medio-alto.
26 % Medio.
42 % Bajo.
21 % Nulo.

5. Aproximadamente, ¿cuántas líneas de código ocupa el programa más largo que has escrito hasta la fecha (sin contar programas en ensamblador)?

26 % 100 líneas o menos (2 páginas en papel).
40 % Entre 100 y 200 líneas (entre 2 y 4 páginas).
16 % Entre 200 y 400 líneas (entre 4 y 8 páginas).
11 % Entre 400 y 800 líneas (entre 8 y 16 páginas).
7 % Más de 800 líneas (más de 16 páginas).

6. Valora tus conocimientos de informática antes de empezar la carrera.

14 % No estaba muy familiarizado con los ordenadores o los usaba sólo para jugar.
67 % Usaba el ordenador a nivel de usuario (acceso a Internet, manejo de Office, etc.).
19 % Sabía programar en un lenguaje, y había hecho mis pequeños programas.
0 % Tenía una experiencia más o menos amplia en programar.

7. ¿Dispones en casa (o residencia) de los siguientes medios?

36 % Ordenador con acceso a Internet de banda ancha.
32 % Ordenador con acceso telefónico a Internet.
30 % Ordenador sin acceso a Internet.
3 % Ninguno de los anteriores.

8. ¿Cuál es tu nota en la asignatura de 1º Programación?

11 % No presentado.
10 % Suspenso.
44 % Aprobado.
25 % Notable.
11 % Sobresaliente o matrícula de honor.

9. ¿Cuál es tu nota en la asignatura de 1º Algoritmos y Estructura de Datos?

3 % No presentado.

18 % Suspenso.
52 % Aprobado.
15 % Notable.
12 % Sobresaliente o matrícula de honor.

10. ¿Qué nota esperas sacar en la asignatura de Ampliación de Algoritmos y Estructura de Datos?

0 % No presentado.
0 % Suspenso.
59 % Aprobado.
25 % Notable.
16 % Sobresaliente o matrícula de honor.

11. ¿Cuál de las siguientes cosas estudiados en Primer Curso consideras más importante?

47 % La recursividad y las iteraciones.
16 % La modularidad, los tipos abstractos de datos.
14 % El manejo del entorno de programación.
10 % El análisis de algoritmos.
7 % Los objetos y las clases en Java.
7 % Los punteros.

12. ¿Cuál de las siguientes cosas estudiados en Primer Curso consideras la segunda más importante?

29 % La modularidad, los tipos abstractos de datos.
25 % La recursividad y las iteraciones.
21 % El análisis de algoritmos.
11 % Los punteros.
8 % Los objetos y las clases en Java.
7 % El manejo del entorno de programación.

13. ¿Y cuál consideras la menos importante?

48 % El manejo del entorno de programación.
19 % Los punteros.
11 % Los objetos y las clases en Java.
11 % El análisis de algoritmos.
8 % La modularidad, los tipos abstractos de datos.
3 % La recursividad y las iteraciones.

14. ¿Compaginas o piensas compaginar los estudios con el desempeño de algún trabajo?

49 % No actualmente, pero puede que sí lo haga.
25 % No, hasta que no acabe la carrera.
22 % Sí, hago algunos trabajos esporádicamente.
4 % Sí, actualmente tengo un trabajo más o menos estable.

Apéndice C

Material de trabajo de clase

C.1. Transparencias de clase del Tema 4 - Grafos

Se muestran algunas transparencias de clase del Tema 4 (Grafos). Se pueden encontrar todas las transparencias de este y otros temas –así como el material de trabajo en clase– en la página web de la asignatura: `http://dis.um.es/~ginesgm/aed.html`

Programa de teoría

Parte I. Estructuras de Datos.

1. Abstracciones y especificaciones.
2. Conjuntos y diccionarios.
3. Representación de conjuntos mediante árboles.
4. **Grafos.**

Parte II. Algorítmica.

1. Análisis de algoritmos.
2. Divide y vencerás.
3. Algoritmos voraces.
4. Programación dinámica.
5. Backtracking.
6. Ramificación y poda.

A.E.D. Tema 4. Grafos. 1

PARTE I: ESTRUCTURAS DE DATOS

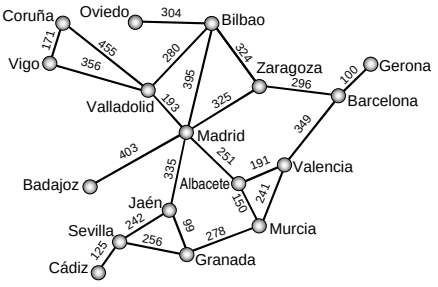
Tema 4. Grafos.

4.1. Introducción, notación y definiciones.
4.2. Representación de grafos.
4.3. Problemas y algoritmos sobre grafos.
- 4.3.1. Recorridos sobre grafos.
- 4.3.2. Árboles de expansión mínimos.
- 4.3.3. Problemas de caminos mínimos.
- 4.3.4. Algoritmos sobre grafos dirigidos.
- 4.3.5. Algoritmos sobre grafos no dirigidos.
- 4.3.6. Otros problemas con grafos.

A.E.D. Tema 4. Grafos. 2

4.1.1. Ejemplos de grafos.

- **Ejemplo:** Grafo de carreteras entre ciudades.



A.E.D. Tema 4. Grafos. 3

4.1.1. Ejemplos de grafos.

- **Ejemplo:** Grafo de carreteras entre ciudades.

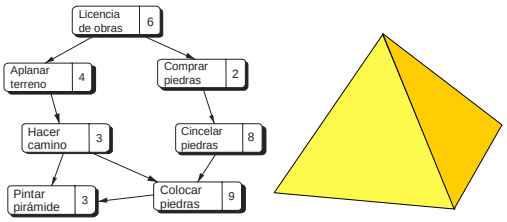
Problemas

- ¿Cuál es el camino más corto de Murcia a Badajoz?
- ¿Existen caminos entre todos los pares de ciudades?
- ¿Cuál es la ciudad más lejana a Barcelona?
- ¿Cuál es la ciudad más céntrica?
- ¿Cuántos caminos distintos existen de Sevilla a Zaragoza?
- ¿Cómo hacer un tour entre todas las ciudades en el menor tiempo posible?

A.E.D. Tema 4. Grafos. 4

4.1.1. Ejemplos de grafos.

- **Ejemplo:** Grafo de planificación de tareas.



4.1.1. Ejemplos de grafos.

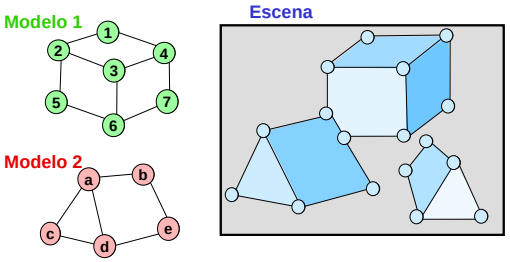
- **Ejemplo:** Grafo de planificación de tareas.

Problemas

- ¿En cuanto tiempo, como mínimo, se puede construir la pirámide?
- ¿Cuándo debe empezar cada tarea en la planificación óptima?
- ¿Qué tareas son más críticas (es decir, no pueden sufrir retrasos)?
- ¿Cuánta gente necesitamos para acabar las obras?

4.1.1. Ejemplos de grafos.

- **Ejemplo:** Grafo asociado a un dibujo de líneas.



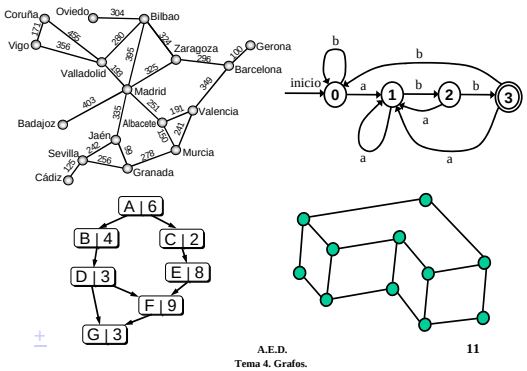
4.1.1. Ejemplos de grafos.

- **Ejemplo:** Grafo asociado a un dibujo de líneas.

Problemas

- ¿Cuántos grupos hay en la escena?
- ¿Qué objetos están visibles en la escena y en qué posiciones?
- ¿Qué correspondencia hay entre puntos del modelo y de la escena observada?
- ¿Qué objetos son isomorfos?

4.1.1. Ejemplos de grafos.

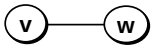


4.1. Introducción y definiciones.

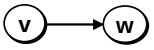
- **Un grafo G** es una tupla G= (V, A), donde V es un conjunto no vacío de **vértices** o **nodos** y A es un conjunto de **aristas** o **arcos**.
- Cada **arista** es un par (v, w), donde v, w V.

Tipos de grafos

- **Grafo no dirigido.**
  Las aristas no están ordenadas:
  (v, w) = (w, v)



- **Grafos dirigidos (o digrafos).**
  Las aristas son pares ordenados:
  <v, w> ζ<w, v>
  <v, w> w = cabeza de la arista, v = cola.



4.1.2. Terminología de grafos.

- **Nodos adyacentes a un nodo v:** todos los nodos unidos a v mediante una arista.
- En grafos dirigidos:
  - **Nodos adyacentes a v:** todos los w con <v, w> A.
  - **Nodos adyacentes de v:** todos los u con <u, v> A.
- Un grafo está **etiquetado** si cada arista tiene asociada una etiqueta o valor de cierto tipo.
- **Grafo con pesos:** grafo etiquetado con valores numéricos.
- **Grafo etiquetado:** G= (V, A, W), con **W**: A o TipoEtiq

4.1.2. Terminología de grafos.

- **Camino de un vértice w_1 a w_q:** es una secuencia $w_1, w_2, ..., w_q$ V, tal que todas las aristas (w_1, w_2), (w_2, w_3), ..., (w_{q-1}, w_q) A.
- **Longitud de un camino:** número de aristas del camino = nº de nodos -1.
- **Camino simple:** aquel en el que todos los vértices son distintos (excepto el primero y el último que pueden ser iguales).
- **Ciclo:** es un camino en el cual el primer y el último vértice son iguales. En grafos no dirigidos las aristas deben ser diferentes.
- Se llama **ciclo simple** si el camino es simple.

4.1.2. Terminología de grafos.

- Un **subgrafo** de G=(V, A) es un grafo G'=(V', A') tal que V' V y A' A.
- Dados dos vértices v, w, se dice que están **conectados** si existe un camino de v a w.
- Un grafo es **conexo** (o **conectado**) si hay un camino entre cualquier par de vértices.
- Si es un grafo dirigido, se llama **fuertemente conexo**.
- Un **componente (fuertemente) conexo** de un grafo G es un subgrafo maximal (fuertemente) conexo.

4.1.2. Terminología de grafos.

- Un grafo es **completo** si existe una arista entre cualquier par de vértices.
- Para **n** nodos, ¿cuántas aristas tendrá un grafo completo (dirigido o no dirigido)?
- **Grado de un vértice v:** número de arcos que inciden en él.
- Para grafos dirigidos:
  - **Grado de entrada de v:** nº de aristas con <x, v>
  - **Grado de salida de v:** nº de aristas con <v, x>

4.1.3. Operaciones elementales con grafos.

- Crear un **grafo vacío** (o con **n** vértices).
- **Insertar** un nodo o una arista.
- **Eliminar** un nodo o arista.
- **Consultar** si existe una arista (obtener la etiqueta).
- **Iteradores** sobre las aristas de un nodo:

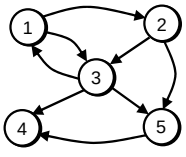
para todo nodo *w* adyacente a *v* **hacer**
acción sobre *w*

para todo nodo *w* adyacente de *v* **hacer**
acción sobre *w*

Mucho menos frecuente

4.2. Representación de grafos.

- **Representación de grafos**:
  - Representación del conjunto de nodos, V.
  - Representación del conjunto de aristas, A.

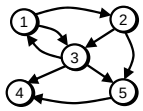


- **Ojo:** las aristas son relaciones "muchos a muchos" entre nodos...

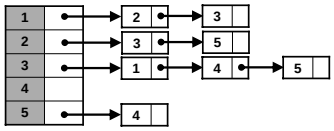
4.2. Representación de grafos.

- **Representación del conjunto de aristas, A.**
  - Mediante **matrices de adyacencia**.

M	1	2	3	4	5
1		T	T		
2			T		T
3	T			T	T
4					
5				T	



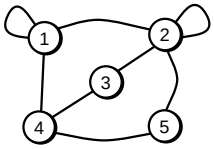
  - Mediante **listas de adyacencia**.



4.2.1. Matrices de adyacencia.

tipo GrafoNoEtiq= **array** [1..n, 1..n] **de** booleano

- Sea M de tipo GrafoNoEtiq, G= (V, A).
- M[v, w] = cierto (v, w) A



M	1	2	3	4	5
1	T	T		T	
2	T	T	T		T
3		T		T	
4	T		T		T
5		T		T	

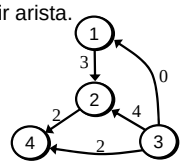
- Grafo no dirigido ˇ Matriz simétrica: M[i, j] = M[j, i].
- **Resultado:** se desperdicia la mitad de la memoria.

4.2.1. Matrices de adyacencia.

- **Grafos etiquetados:**

  tipo GrafoEtiq[E]= **array** [1..n, 1..n] **de** E

- El tipo E tiene un valor NULO, para el caso de no existir arista.



M	1	2	3	4
1		3		
2				2
3	0	4		2
4				

- ¿Cómo serían los iteradores: **para todo** adyacente a, y adyacente de? ¿Y contar número de aristas?
- ¿Cuánto es el tiempo de ejecución?

4.2.1. Matrices de adyacencia.

Uso de memoria

- k_2 bytes/etiqueta
- **Memoria usada:** $k_2 n^2$

Ventajas

- Representación y operaciones muy sencillas.
- Eficiente para el acceso a una arista dada.

Inconvenientes

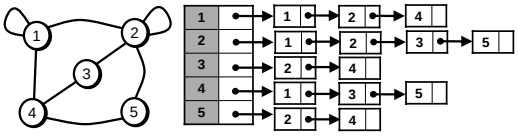
- El número de nodos del grafo no puede cambiar.
- Si hay muchos nodos y pocas aristas ($a<<n^2$) se desperdicia mucha memoria (matriz *escasa*).

4.2.2. Listas de adyacencia.

tipo Nodo= entero (1..n)

tipo GrafoNoEtiq= **array** [1..n] **de** Lista[Nodo]

- Sea R de tipo GrafoNoEtiq, G= (V, A).
- La lista R[v] contiene los w tal que (v, w) A.

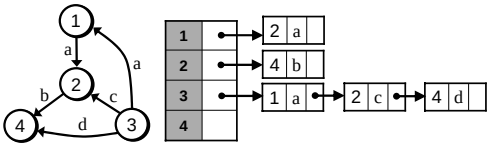


- Grafo no dirigido ˇ Las aristas están repetidas.
- **Resultado:** también se desperdicia memoria.

4.2.2. Listas de adyacencia.

- **Grafos etiquetados:**

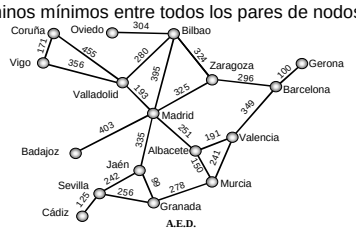
  tipo GrafoEtiq[E]= **array** [1..n] **de** Lista[Nodo,E]



- ¿Cómo serían los iteradores: **para todo** adyacente a, y adyacente de? ¿Y contar número de aristas?
- ¿Cuánto es el orden de complejidad? Se suponen: **n** nodos y **a** aristas.

4.3.3. Problemas de caminos mínimos.

- **Coste de un camino:** suma de los costes de las aristas por las que pasa.
- **Problemas de caminos mínimos:**
  - Camino mínimo entre dos nodos, **v** y **w**.
  - Caminos mínimos entre un nodo **v** y todos los demás.
  - Caminos mínimos entre todos los pares de nodos.

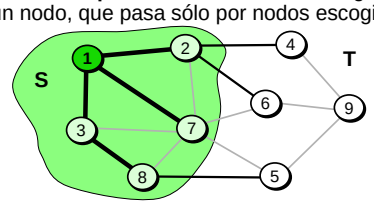


4.3.3.1. Caminos mínimos desde un origen. Algoritmo de Dijkstra

- Supongamos un grafo **G**, con pesos positivos y un nodo origen **v**.
- El algoritmo trabaja con dos conjuntos de nodos:
  - **Escogidos: S**. Nodos para los cuales se conoce ya el camino mínimo desde el origen.
  - **Candidatos: T**. Nodos pendientes de calcular el camino mínimo, aunque conocemos los caminos mínimos desde el origen pasando por nodos de **S**.

4.3.3.1. Caminos mínimos desde un origen.

- **Camino especial:** camino desde el origen hasta un nodo, que pasa sólo por nodos escogidos, **S**.



- **Idea:** En cada paso, coger el nodo de **T** con menor distancia al origen. Añadirlo a **S**.
- Recalcular los caminos mínimos de los demás candidatos, pudiendo pasar por el nodo cogido.

4.3.3.1. Caminos mínimos desde un origen. Algoritmo de Dijkstra

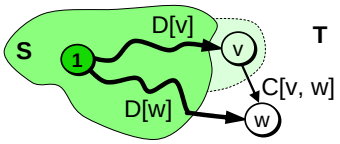
- **Inicialización: S**= {1}, **T**= {2, ..., n}, caminos especiales mínimos = caminos directos.
- **Repetir** n-1 veces:
  - Seleccionar el nodo **v** de **T** con el camino especial más corto.
  - **Proposición:** el camino mínimo para este nodo **v**, coincide con su camino especial.
  - Recalcular los caminos especiales para los nodos de **T**, pudiendo pasar por **v**.

4.3.3.1. Caminos mínimos desde un origen. Implementación del algoritmo de Dijkstra

- Suponemos que el origen es el nodo 1.
- **D: array** [2..n] **de** real. **D[v]** almacena el coste del camino especial mínimo para el nodo **v**.
- **P: array** [2..n] **de** entero. **P[v]** almacena el último nodo en el camino especial mínimo de **v**.
- **Inicialización:** D[v]:= C[1, v], P[v]:= 1
- **Nodo seleccionado:** nodo de **T** con mínimo D[v]
- **Actualización:** para todos los **w** de **T** hacer

```
si D[v] + C[v, w] < D[w] entonces
        D[w]:= D[v] + C[v, w]
        P[w]:= v
finsi
```


4.3.3.1. Caminos mínimos desde un origen.



- **Camino especial para w:**
  - Sin pasar por **v**: D[w]
  - Pasando por **v**: D[v] + C[v,w]
  - Nos quedamos con el menor.
- Si el menor es pasando por **v** entonces: **P[w]= v**.
- Camino especial para **w**:
  1 o ... o P[P[P[w]]] o P[P[w]] o P[w] o w

C.2. Boletín de ejercicios del Tema 4 - Grafos

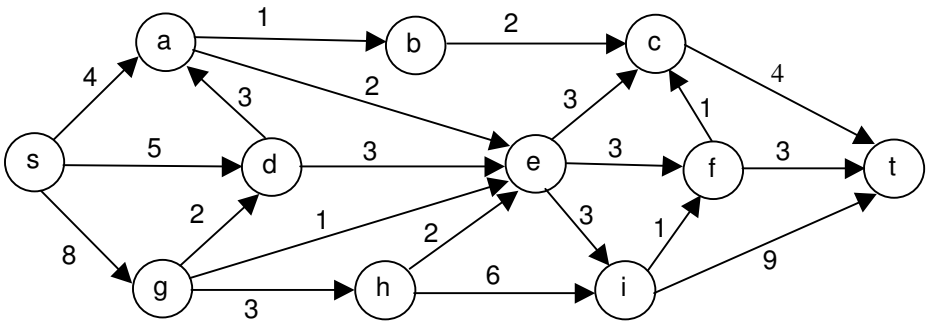
Se muestran algunos de los ejercicios propuestos en el boletín correspondientes al Tema 4. El boletín original contenía otros 20 ejercicios más, omitidos por razones de espacio. La indicación "(EX)" indica que el ejercicio está extraído de algún examen anterior, mientras que "(TG ...)" significa que está planteado y resuelto en el texto guía de la asignatura.

98. Dado un árbol de expansión resultante de un recorrido sobre un grafo no dirigido, ¿qué tipo de arcos (aparte de los del árbol) pueden aparecer si el recorrido es una búsqueda en profundidad o una búsqueda en anchura? ¿Qué arcos aparecerán si el recorrido (en profundidad o en anchura) es aplicado sobre un grafo dirigido?

99. Puesto que la búsqueda primero en profundidad es equivalente a un recorrido en pre-orden de un árbol, un programador decide implementar el equivalente a un recorrido en post-orden (orden posterior). Para ello, modifica el procedimiento `bpp` haciendo que la marca de visitado sea establecida al final del mismo. Además, añade también al final una instrucción `write(v)` para que se muestre el orden de visita de los nodos. ¿Es correcta esta modificación para realizar un recorrido en orden posterior? En caso negativo, ¿cómo se solucionaría?

100. ¿Cuál es el número máximo de arcos que puede tener un grafo no dirigido sin ciclos? ¿Y cuál será para un grafo dirigido acíclico (GDA)?

101. (EX) Mostrar una ordenación topológica para el siguiente grafo dirigido acíclico. ¿La ordenación obtenida es única? En caso negativo mostrar otra ordenación válida.



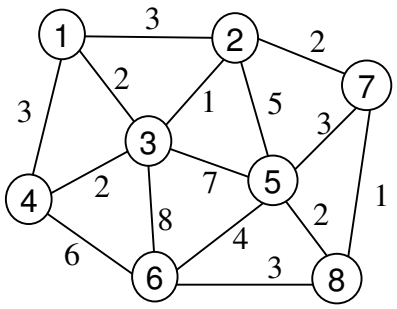
102. Suponer el grafo no dirigido de la siguiente figura. Mostrar:

    a) El bosque de expansión en profundidad, empezando en `a` y en `d`.

    b) El bosque de expansión en amplitud, empezando en `a` y en `d`.

    c) El árbol de expansión de coste mínimo utilizando el algoritmo de Prim.

    d) El árbol de expansión de coste mínimo utilizando el algoritmo de Kruskal. ¿Son iguales las soluciones obtenidas en ambos algoritmos? En caso contrario, ¿son válidas las distintas soluciones? ¿Por qué?



103. Implementar la prueba de aciclicidad en un grafo no dirigido. ¿Se puede hacer en un tiempo $O(n)$? **Sugerencia:** tener en cuenta el resultado del ejercicio 100.

104. Modificar el procedimiento anterior para que, en caso de encontrar que existe un ciclo, devuelva en una lista los elementos que forman el ciclo encontrado. Se supondrá que tenemos un tipo ListaNodos, con operaciones Anula (Lista), InsertaCabeza (nodo, Lista) e InsertaCola (nodo, Lista), para añadir un nodo al principio y al final de la lista, respectivamente.

105. Escribir un procedimiento que realice una búsqueda primero en profundidad sobre un grafo dirigido, y que cada vez que encuentre una arista que no es del árbol calcule de qué tipo es (de avance, retroceso o cruce) y lo muestre por pantalla.

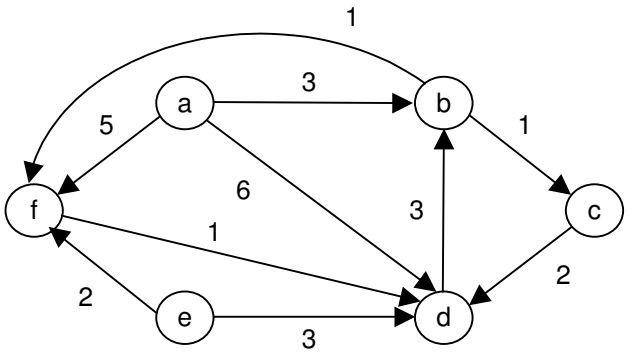
106. Diseñar un algoritmo para contar el número de ciclos simples existentes en un grafo no dirigido. ¿Cuál es el orden de complejidad de este algoritmo? ¿Cuántos ciclos pueden haber como máximo en un grafo cualquiera?

107. ¿Cómo se podría modificar el procedimiento bpa (utilizado en la búsqueda primero en anchura) para realizar un recorrido en profundidad, realizando un cambio mínimo en el algoritmo? **Sugerencia:** considera la estructura de datos utilizada.

108. (EX) Modificar el procedimiento de búsqueda primero en profundidad para que cuente el número de árboles del bosque de expansión en profundidad. Además, se debe almacenar en un array MA[1..n] el número de árbol al que pertenece cada nodo. Se supone que el primer árbol generado es el 1, luego el 2 y así sucesivamente.

109. Demostrar que el algoritmo de Dijkstra no funciona cuando las aristas pueden tener coste negativo, aun cuando no existan ciclos en el grafo. Sugerencia: dar un contraejemplo de un grafo dirigido sin ciclos en el que el algoritmo de Dijkstra no dé el resultado correcto.

110. Utilizar el algoritmo de Dijkstra para encontrar los caminos más cortos que van desde el nodo a hasta los restantes nodos, en el siguiente grafo dirigido. Mostrar los valores S, D y P para todos los pasos de ejecución del algoritmo. A partir del resultado, encontrar cuál es el camino más corto desde a hasta d.

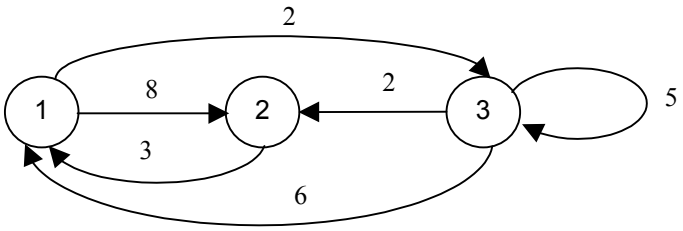


111. (TG 5.2) En general, dados dos nodos, v y w, en un grafo con pesos puede existir más de un camino mínimo entre ambos (aunque necesariamente todos tendrán el mismo coste). Modificar los algoritmos de Dijkstra y Floyd para contar el número de caminos mínimos distintos existentes entre un nodo v y el resto de nodos, o entre todos los nodos, respectivamente.

112. ¿Cuál es el orden de complejidad del algoritmo para encontrar los componentes fuertemente conexos en un grafo dirigido? Suponer que el grafo tiene n nodos y a aristas, siendo $a > n$.

113. Aplicar el algoritmo para obtener los componentes fuertemente conexos para el grafo del ejercicio 110. A partir del resultado obtenido, mostrar el grafo reducido correspondiente.

114. Mostrar el resultado de la aplicación del algoritmo de Floyd sobre el siguiente grafo dirigido.



115. Se define la excentricidad de un nodo v como la distancia máxima de los caminos mínimos desde todos los nodos hasta v. Por otro lado, se define el centro de un grafo como el nodo con menor excentricidad. Dar un algoritmo para calcular el centro de un grafo. ¿Cuál es el centro del grafo del ejercicio 114?

116. (EX) Modificar el algoritmo de Dijkstra para que, además de calcular los caminos mínimos entre un nodo y los demás, también calcule en otro array L el número de aristas por las que pasa cada uno de los caminos mínimos. Decir únicamente las partes que se deben modificar del algoritmo.

117. (EX) Modificar el algoritmo de Dijkstra para que, además de calcular los caminos mínimos entre un nodo y los demás, también calcule otro array L de booleanos, indicando para cada nodo si su camino mínimo desde el origen es único o no. Decir únicamente las partes que se deben modificar del algoritmo.

118. Demostrar que el grafo reducido de un grafo dirigido G cualquiera (el que representa cada componente conexo de G con un nodo) es siempre un GDA.

119. Un grafo no dirigido $\mathsf{G} = (\mathsf{V}, \mathsf{A})$, se dice que es bipartido si V se puede partir en dos subconjuntos $\mathsf{V1}$ y $\mathsf{V2}$, de manera que para toda arista (v, w) de A, el nodo v pertenece a uno de los conjuntos ($\mathsf{V1}$ ó $\mathsf{V2}$) y w pertenece al otro. Proporcionar un algoritmo con tiempo $O(n+a)$ para comprobar si un grafo es bipartido o no. **Sugerencia:** Usar una bpp para asignar una partición (1 ó 2) a cada nodo visitado.

121. (EX) Para desarrollar la especificación formal del TAD grafo dirigido y etiquetado disponemos de los siguientes conjuntos:

    G Conjunto de grafos dirigidos y etiquetados

    M Conjunto de nodos de los grafos

    E Conjunto de valores de las etiquetas en las aristas

    N Conjunto de naturales

    B Conjunto de booleanos

    U Conjunto de mensajes de error

    Escribir la parte de sintaxis correspondiente a las operaciones que son los constructores del tipo. Poner también la sintaxis y la semántica de las principales operaciones de modificación y consulta sobre el grafo.

122. Suponer que estamos trabajando con GDA. ¿Es posible mejorar la eficiencia obtenida con el algoritmo de Dijkstra para este tipo de grafos? ¿Cómo sería la modificación de este algoritmo para conseguir la mejora? **Idea:** haz uso de la ordenación topológica, para seleccionar los nodos entre los candidatos.

C.3. Ejercicios del Seminario de especificaciones algebraicas

1. (1 punto) Comprobar el resultado de las siguientes expresiones usando las especificaciones formales definidas en el punto 4 del guión del prácticas[1]. Decir cuántos axiomas es necesario aplicar en cada caso:

    a) push(tope(push(a, crearPila)), pop(push(e, push(i, crearPila))))

    b) igual(sucesor(sucesor(cero)), suma(sucesor(cero), sucesor(cero)))

    c) tope(pop(pop(push(i, push(o, push(u, crearPila))))))

    d) igual(e, tope(pop(push(e, pop(crearPila)))))

2. (2 puntos) Añadir a la especificación formal del TAD **Natural** las operaciones: predecesor, resta, producto, potencia, factorial, esMenor, esMenorIgual y esPar. Convertir las siguientes expresiones a la notación definida y comprobar el resultado que se obtiene, indicando el número de axiomas aplicados.

    a) $2^2 * (3-1)$ ¿$3 + 3^2 < 3 * 2^3$? ¿$4! = 2^{2^2}$? ¿Es par $(3! - 3^2)$?

    b) $(3-2)^{(2+1)}$ ¿$(4+2) * 2 * 1 < 4!$? ¿$2^{(1+2)} = 2!$? ¿Es par $(2 + 2 - 2 * 3)$?

    c) $3 + 2 * 2 - 1!$ ¿$(2-2)^3 < 2^{3-3}$? ¿$3! = 1 - 2 - 3$? ¿Es par $(2^{1+1} - 1^2)$?

    d) $(2 * 3)^{(3-1-1)}$ ¿$2^2! == 2^2$? ¿$(2+1)! < (2-3)^{2+1}$? ¿Es par $(4! - 3!)$?

[1] En este punto del guión se definen los tipos: natural, letra y pila.

3. (3 puntos) Escribe una especificación formal para el TAD **lista de letras**. La especificación debe incluir las operaciones: vacia (devuelve una lista vacía), insertar (inserta un elemento al principio de una lista), concatenar (unir dos listas), longitud (devuelve la longitud de una lista), primero, ultimo, cabecera (devuelve todos los elementos menos el último), cola (devuelve todos los elementos menos el primero) e invertir (invierte el orden de los elementos de una lista). Probar la especificación con al menos 4 expresiones de ejemplo, en las que aparezcan todas las operaciones definidas.

4. (4 puntos) Escribe la especificación formal del TAD **árbol binario de letras**. La especificación debe incluir las operaciones: crear (crea un árbol vacío), construir (crea un árbol, dada la raíz y dos subárboles), hijoIzq, hijoDer, raiz (operaciones de consulta), altura (calcula la altura del árbol), numNodos (calcula el número de nodos del árbol), esAVL (comprueba si cumple la condición de balanceo de los árboles AVL), esPerfBal (comprueba si cumple la condición de los árboles perfectamente balanceados), preOrden, inOrden y postOrden (recorre un árbol y almacena el resultado en una lista de letras). Escribe las expresiones correspondientes a crear dos árboles binarios que contengan las vocales del nombre y apellidos de cada miembro del grupo. Se requiere que uno de ellos sea un AVL y otro no. Dibujar a mano la estructura de los árboles creados. Probar los resultados obtenidos de aplicar las 7 últimas operaciones anteriores sobre esos árboles.

   Ejemplo. Alumno: Ginés García Mateos Expresión: construir(a, construir(i, crear, construir(e, crear, crear)), construir(a, construir(i, crear, crear), construir(e, construir(a, crear, crear), construir(o, crear, crear))))

C.4. Enunciado de la Práctica 1 - Análisis y diseño de estructuras de datos

A. Contextualización

Ante el imparable crecimiento de Internet, el uso de herramientas de búsqueda se hace cada vez más necesario; y no sólo a nivel global, sino a nivel interno de cada empresa u organización. El problema subyacente en este tipo de herramientas es un típico caso de búsqueda de ciertos patrones entre una serie de datos, pero condicionado por los grandes volúmenes de información manejados: varios millones de páginas, cientos de palabras por página y algunos miles de consultas por minuto.

Por este motivo, hemos recibido el encargo de una empresa de desarrollo de software de crear un motor de búsqueda eficiente para aplicación de buscador web. La empresa, por su parte, se encargará del desarrollo de la interface gráfica con el usuario. Nosotros debemos ajustarnos a las funciones de entrada y salida del motor de búsqueda.

B. Enunciado del problema

El buscador trabaja con un **fichero maestro**, en el que se listan todas las páginas web disponibles para la búsqueda. Este fichero contiene otro tipo de información sobre las páginas, como el título y la relevancia (o interés de la página, definida en términos absolutos) de cada página. Además del fichero maestro, el contenido en sí de las páginas está disponible localmente, en forma de un fichero de texto por cada página web. En el fichero maestro se indica el nombre del fichero local en el que se encuentra cada página.

En relación al manejo del fichero maestro, se requieren operaciones para: cargar un fichero maestro para su utilización en las búsquedas posteriores, insertar una nueva página, eliminar una página y guardar un fichero maestro en disco con los cambios realizados.

En cuanto a la consulta de páginas web, se deberá realizar una implementación eficiente de las operaciones de consulta por palabras. La consulta recibirá como entrada una lista con una o varias palabras. El resultado de la búsqueda dependerá del **tipo de consulta**. Se definen tres tipos de consulta: AND- buscar las páginas que contengan todas las palabras de la lista; OR- buscar las páginas que contengan alguna de las palabras de la lista; COMILLAS- buscar las páginas que contengan todas las palabras de la lista y además que aparezcan consecutivamente en el mismo orden.

Opcionalmente, la empresa nos sugiere la posibilidad de recibir un suplemento extra por aplicar nuestros conocimientos en teoría de autómatas para implementar consultas donde se puedan combinar varios de los operadores anteriores (y, posiblemente, junto con un operador NOT).

C. Especificaciones del producto

El motor de búsqueda desarrollado debe ajustarse a las siguientes especificaciones. Estas especificaciones son de obligado cumplimiento, salvo las que se indiquen como opcionales. Se entiende que todo lo que no esté especificado se deja a la libre elección del implementador, en función de su criterio profesional. En cualquier caso, las decisiones a este respecto deberán estar justificadas adecuadamente.

C.1. Formato del fichero maestro El fichero maestro es un fichero de texto ASCII en el que se listan todas las páginas sobre las que el servidor puede realizar las búsquedas. El contenido de estas mismas páginas está disponible como ficheros accesibles locamente, uno por página, que llamaremos ficheros de página.

Para cada página web disponible, existirá una línea de descripción en el fichero maestro con la información relativa a la misma. La línea tendrá el siguiente formato:

```
DIR_URL NOMBRE_FICH_PAGINA RELEVANCIA TITULO_PAGINA
```

Todos los campos anteriores están separados por espacios en blanco o tabuladores (uno o varios). Un carácter "`#`" al comienzo de una línea indicará un comentario que se extiende hasta el final de la línea. También se ignorará cualquier línea que no contenga todos los campos. El significado de los campos es el siguiente:

DIR_URL Dirección URL de la página. Normalmente será del tipo: `http://www...`

NOMBRE_FICH_PAGINA Nombre del fichero local donde se encuentra el contenido de la página. La página puede estar ubicada en un subdirectorio, en cuyo caso este nombre contendrá la ruta relativa. Por ejemplo, el nombre puede ser: `index.htm` o `elmundo.es/index.html`. En cualquier caso, el nombre no puede incluir espacios en blanco (ni tampoco la URL).

RELEVANCIA Número entero que indica la importancia o interés de la página web, definida de manera absoluta. Cuanto mayor sea, se considera que la página será más interesante. El método por el que se ha asignado este valor nos es indiferente. El interés de este valor es que, al devolver los resultados de una búsqueda, las páginas deben estar ordenadas de mayor a menor relevancia.

TITULO_PAGINA Descripción textual y acortada del contenido de la página. Este campo sí puede contener espacios en blanco, de manera que se supone que el campo se extiende hasta el final de la línea correspondiente.

C.2. Formato de los ficheros de página Al igual que el fichero maestro, los **ficheros de página** son también archivos de texto ASCII. Tendrán normalmente extensión .html, .htm, .txt o cualquier otra. Un fichero de página contiene palabras y otras cosas que no son palabras (símbolos especiales, números, etiquetas HTML, etc.). Podemos desechar todo lo que no sean palabras. Definimos las palabras de la siguiente manera: una palabra es una sucesión se letras, separadas por cosas que no son letras. Las letras incluyen la ñ y las vocales con tilde. Por otro lado, la búsqueda de palabras debe ser independiente de mayúsculas/minúsculas, o de que las vocales lleven tilde o no.

C.3. Interface del programa Todas las interacciones entre el programa y el usuario se realizarán utilizando la entrada y la salida estándar. No se creará un interface complejo con el usuario, ya que esta parte es responsabilidad de la empresa contratante. En este sentido, el programa debe poder admitir un uso *off-line*, esto es, redireccionando la entrada y la salida estándar a ficheros.

En particular, la interface del programa será una línea de comandos que deberá aceptar las siguientes órdenes:

>> **load FICHERO_MAESTRO**
Leer el fichero maestro con nombre FICHERO_MAESTRO. Esta operación servirá para inicializar en memoria las estructuras de datos necesarias para el programa. Si hay otro fichero cargado en ese momento, deberá liberarse la memoria que ocupaba. Al acabar la lectura se deberá mostrar información resumida con, al menos: número de páginas leídas, número de palabras (total y distintas) y tiempo total de carga. Opcionalmente se mostrará la ocupación de memoria, en bytes.

>> **insert URL NOMBRE RELEVANCIA TITULO**
Insertar una nueva página para la búsqueda. Los parámetros, el formato y su significado son los mismos que los que se indican en el fichero maestro. Al finalizar se mostrará el número de palabras leídas (total y distintas). Opcionalmente se mostrará también la nueva ocupación total de memoria, en bytes.

>> **remove NOMBRE** (comando opcional)
Eliminar una página, pasando como parámetro el nombre del fichero de página. Se podrá mostrar la nueva ocupación total de memoria.

>> **save FICHERO_MAESTRO** (comando opcional)
Guardar la lista de páginas disponibles (teniendo en cuenta las inserciones y eliminaciones realizadas) en el archivo FICHERO_MAESTRO, con el formato especificado del fichero maestro.

>> **search and PALABRA$_1$ PALABRA$_2$ ... PALABRA$_K$**
Buscar todas las páginas que contengan todas las palabras especificadas como parámetros. El resultado deberá seguir el siguiente formato. La primera línea será un número real, indicando el tiempo total de búsqueda, en milisegundos, medido con la mayor precisión posible. La siguiente línea indicará el número total de páginas encontradas que se ajustan a los criterios de búsqueda. A continuación vienen las páginas resultantes de la búsqueda. Para cada página aparecerá una línea, que contendrá un número consecutivo (empezando por 1, 2, ...), la URL de la página, la relevancia y el título. Las páginas deben estar ordenadas de mayor a menor valor de relevancia. Ejemplo de búsqueda y resultado:

```
>> search and gines garcia mateos

0.251
```

```
3

1. http://dis.um.es/~ginesgm 165 Algoritmos y Estructuras

2. http:// acm.uva.es/cgi-bin/list.cgi 86 Author Ranklist

3. http://link.springer.de/link/2091.htm 12 SpringerLink - Chapter
```

Opcionalmente, después de la línea correspondiente a cada página puede mostrarse otra línea con un trozo del texto de esa página, donde aparezca alguna de las palabras buscadas. Esta opción se puede aplicar también a las otras operaciones de búsqueda.

>> **search or PALABRA$_1$ PALABRA$_2$ ... PALABRA$_K$**
Igual que **search and**, pero el resultado son las páginas que contengan alguna de las palabras pasadas como parámetro.

>> **search com PALABRA$_1$ PALABRA$_2$ ... PALABRA$_K$** (comando opcional)
Igual que **search and**, pero además las palabras deben aparecer consecutivamente y en el mismo orden en las páginas resultantes.

>> **search EXPRESION** (comando opcional)
Buscar páginas por palabras, utilizando una expresión donde pueden aparecer varias palabras, combinadas con los operadores AND, OR, COM, NOT y paréntesis.

>> **help**
Mostrar información sobre todos los comandos disponibles. Documentar especialmente los comandos nuevos, en caso de que se haya introducido alguno.

>> **quit**
Salir del programa, liberando la memoria que se haya utilizado. Este comando no implica guardar el fichero maestro.

C.5. Examen de AAED de septiembre de 2003

Ampliación de Algoritmos y Estructuras de Datos
Examen. 10 de Septiembre de 2.003

PRIMERA PARTE

1. (1.25 puntos) El tiempo de ejecución de determinado algoritmo viene dado por la siguiente ecuación de recurrencia:

$$t(n) = \begin{cases} 2n^2 + 8 & \text{Si } n \quad 4 \\ 4\,t(n-1)\ \text{í}\ 5\,t(n-2) + 2\,t(n-3) + \log_2 n + 5 & \text{Si } n > 4 \end{cases}$$

   Encontrar el orden exacto, , , del tiempo de ejecución del algoritmo. Sugerencia: en caso de no poder despejar el tiempo, probar buscando cotas superiores e inferiores del mismo.

2. (1.25 puntos) Considerar el siguiente algoritmo para generar todas las combinaciones de **n** elementos de 4 en 4. Se supone que **marca** y **conv** son arrays globales de tamaño **n**, inicializados con valor 0. La llamada inicial al procedimiento es: **Combinaciones(1)**.

```
procedure Combinaciones (k: integer)
(1) for i:= 1 to n do begin
(2)     if marca[i] = 0 then begin
(3)         conv[k]:= i
(4)         if k = 4 then
(5)             escribir (conv[1], conv[2], conv[3], i)
(6)         else begin
(7)             marca[i]:= 1
(8)             Combinaciones(k+1)
(9)             marca[i]:= 0
(10)        end
(11)    end
(12) end
```

   Calcular el tiempo de ejecución del algoritmo. Expresar el orden de complejidad del algoritmo usando la notación asintótica que consideres más adecuada.

3. (3.5 puntos) Contesta las siguientes preguntas de forma breve, clara y razonada.

   a) (0.8 puntos) Definimos el TAD **Sensor** con las siguientes operaciones: *Crear*, *Más*, *Menos* y *Estado*. La operación *Crear* recibe un entero, que es el "tope" del sensor, y devuelve un sensor nuevo. Las operaciones *Más* y *Menos* reciben un sensor y devuelven otro sensor. La operación *Estado* es de consulta: devuelve ON si el número de veces que se ha aplicado *Más*, menos el número de veces que se ha aplicado *Menos* es mayor o igual que el tope del sensor; en caso contrario devuelve OFF. Escribir una especificación formal, algebraica o constructiva, del tipo **Sensor**.

Ampliación de Algoritmos y Estructuras de Datos
Examen. 10 de Septiembre de 2.003

PRIMERA PARTE

b) (0.7 puntos) En una aplicación de base de datos representamos la información mediante árboles AVL, almacenados en disco. Cada sector de disco ocupa 512 bytes, en los cuales caben hasta 64 nodos del árbol. Estamos interesados en contar las operaciones de escritura en disco, que son las más costosas. Determinar, de manera razonada, cuántas escrituras (es decir, operaciones de escribir en bloques de disco distintos) se pueden realizar como máximo en el peor caso en una inserción de un elemento en el árbol AVL.

c) (0.6 puntos) Demostrar una de las dos siguientes afirmaciones.
   i. En un sistema monetario con monedas de valores (1, 2, 2^2, 2^3, ..., 2^k), el algoritmo voraz para el problema del cambio de monedas encuentra siempre la solución óptima.
   ii. El algoritmo voraz para el problema de la mochila 0/1 (es decir, seleccionando los objetos por orden de máximo beneficio por unidad de peso y metiéndolos si caben enteros) no garantiza la solución óptima.

d) (0.6 puntos) Indicar cuál/cuáles de las siguientes afirmaciones son falsas y justificar por qué. No hace falta justificar las ciertas.
   i. Las tablas de dispersión son adecuadas para recorrer un conjunto de elementos secuencialmente por orden de clave.
   ii. En dispersión cerrada, aunque se conozca el número de elementos que serán almacenados, se puede mejorar la eficiencia usando más cubetas.
   iii. La única ventaja de las tablas de dispersión respecto a los árboles AVL es que necesitan menos memoria, al usar menos punteros.

e) (0.8 puntos) Cierto problema se puede resolver utilizando la siguiente ecuación de recurrencia:

   Dinamico (0, j) = j^2, para todo j *(caso base)*
   Dinamico (i, j) = a*Dinamico (i-1, j) + b*Dinamico (i-1, j+1), para i>0

   Siendo a y b constantes. El resultado final del problema es: Dinamico (N, 0). Escribir un algoritmo de programación dinámica para resolver el problema. ¿Cuáles son las dimensiones de la tabla que se debe usar?

Ampliación de Algoritmos y Estructuras de Datos
Examen. 10 de Septiembre de 2.003

SEGUNDA PARTE

4. (2.0 puntos) Para una boda tenemos una lista de **n** personas que pueden ser invitadas o no. Pero no todas pueden serlo, ya que existen enemistades conocidas entre ellas, que debemos evitar. Las enemistades conocidas vienen dadas como pares ($\mathbf{a_1}$, $\mathbf{b_1}$), ($\mathbf{a_2}$, $\mathbf{b_2}$),…, ($\mathbf{a_k}$, $\mathbf{b_k}$), indicando cada par **i**-ésimo que $\mathbf{a_i}$ y $\mathbf{b_i}$ son enemigos conocidos. Aplicando transitividad, los enemigos de una persona **i** son todos los enemigos conocidos de **i** y los enemigos de sus amigos. Decimos que los amigos de una persona son todos los enemigos de sus enemigos.

   Una persona asistirá a la boda si es invitada y son invitados todos sus amigos y no es invitado ninguno de sus enemigos. El objetivo del problema es decidir la lista de invitados de manera que se maximice el número de asistentes a la boda.

   Diseñar un algoritmo para resolver el problema, justificando razonadamente su funcionamiento. Acotar el orden de complejidad del algoritmo diseñado. Mostrar la ejecución sobre el siguiente ejemplo.

   Posibles invitados: **n** = 8

   Enemistades conocidas: (1,2), (5,7), (6,8), (1,4), (2,7)

   Nota: Aplicando las definiciones es posible, en algún caso, que dos personas sean a la vez amigos y enemigos. En ese caso ninguno de ellos puede ser invitado.

5. (2.0 puntos) El problema del empaquetamiento en recipientes consiste en lo siguiente. Tenemos **n** objetos de pesos $\mathbf{w_1}$, $\mathbf{w_2}$,…, $\mathbf{w_n}$, y un número ilimitado de recipientes con capacidad máxima **R** (siendo $\mathbf{w_i}$ **R**, para todo **i**). Los objetos se deben meter en los recipientes sin partirlos, y sin superar su capacidad máxima. Se busca el mínimo número de recipientes necesarios para colocar todos los objetos.

   Diseñar un algoritmo por la técnica de ramificación y poda para encontrar la solución óptima del problema. Indicar la forma del árbol, la representación de la solución, la manera de calcular las cotas, el esquema del algoritmo y las funciones del mismo. Mostrar la ejecución del algoritmo sobre algún ejemplo sencillo.

C.6. Página web de la asignatura

08BZ - ALGORITMOS Y ESTRUCTURAS DE DATOS

Año 2003/2004
Curso Segundo
Ingeniero en Informática
Universidad de Murcia

AVISOS | APUNTES | PRACTICAS | PR. OPTATIVAS | TUTORIAS | EXAMENES | EVALUACION | CONVALIDACIONES | BIBLIOGRAFIA

AVISOS PARA LOS ALUMNOS

- Está disponible el examen de febrero de 2004 (primer parcial).

- Se han añadido algunas indicaciones imporantes sobre cómo realizar la Práctica 1. Se recomienda leerlas antes de entregar las prácticas.

- **¡¡Retos!!** Sigue abierto el reto del juego de las cifras (ver en la sección Prácticas Optativas). El primero que lo supere tendrá la recompensa indicada. El reto del balón de fútbol ha sido resuelto. Ver la solución (por Bruno Recasens).

- Ya están disponibles las transparencias de los dos primeros temas (tema 0 y 1) de la Parte II.

- **Atención:** El Primer Parcial de la asignatura será el sábado 28 de febrero, por la mañana (ver el llamamiento). La entrega de la Práctica 1 se retrasa hasta el 5 de marzo.

- En la sección de Prácticas se ha dejado el programa Generador para crear casos de prueba para la Práctica 1 (para generar conjuntos de páginas web creadas aleatoriamente) y las tablas de desarrollo del proyecto (PDF, Word97), necesarias para la memoria.

- Si buscas información de la asignatura relativa a las convocatorias del curso 2002/2003 usa esta página.

APUNTES

Parte I: Estructuras de datos

Tema 1. Abstracciones y especificaciones.
transparencias: PDF, PowerPoint97; ejercicios: PDF, Word97

Tema 2. Conjuntos y diccionarios.
transparencias: PDF, PowerPoint97; ejercicios: PDF, Word97

Tema 3. Representación de conjuntos mediante árboles.
transparencias: PDF, PowerPoint97; ejercicios: PDF, Word97

Tema 4. Grafos.
transparencias: PDF, PowerPoint97; ejercicios: PDF, Word97

Parte II: Algoritmos

Tema 0. Algorítmica.
transparencias: PDF, PowerPoint97

Tema 1. Análisis de algoritmos.
transparencias: PDF, PowerPoint97

Tema 2. Divide y vencerás.

Tema 3. Algoritmos voraces.

Tema 4. Programación dinámica.

Tema 5. Backtracking.

Tema 6. Ramificación y poda.

PRÁCTICAS

- Las prácticas consistirán en el análisis, diseño, implementación y prueba de algunos problemas de algoritmos y de estructuras de datos, usando el lenguaje de programación C++.

- Los 6 créditos de prácticas se reparten en 1,5 créditos en el seminario de introducción a C/C++, y 4,5 créditos repartidos en dos o tres prácticas en la modalidad de laboratorio abierto.

- Como NO hacer unas prácticas de programación.

- **Seminario 1**: Programación en C
  Sesión 1: PDF, Word97
  Sesión 2: PDF, Word97
  Sesión 3: PDF, Word97
  Sesión 4: PDF, Word97

- **Seminario 2**: Programación en C++
  Sesión 1: PDF, Word97, TXT
  Sesión 2: PDF, Word97, TXT
  Sesión 3: PDF, Word97, TXT

- **Práctica 1: Implementación y manejo de estructuras de datos**
  Indicaciones importantes
  Enunciado de la práctica: PDF, Word97
  Programa generador de casos de prueba: Generador, Leeme
  Tablas de desarrollo del proyecto (necesarias para la memoria): PDF, Word97
  Fecha de entrega: Viernes 5/Marzo/2004

- **Práctica 2**: Eficiencia, evaluación y predicción

- **Práctica 3**: Resolución de problemas

PRACTICAS OPTATIVAS

El juego de las CIFRAS

Dado un conjunto de 4, 5, 6, ... ó 10 enteros (entre 1 y 100) encontrar la forma de conseguir otro entre 1 y 2000, utilizando las operaciones de suma, resta, producto y división entera. Cada número del conjunto sólo se puede usar una vez.
Versión a retar: cifras.exe (executable Win95)

Reto: Diseñar, implementar y documentar una solución al problema más rápida que el programa "cifras.exe", y que no pierda ninguna solución.
Evaluación: Tras la correspondiente entrevista con el profesor, la evaluación si se supera el reto será: un 10 en la tercera práctica de la asignatura, un 10 en un ejercicio correspondiente del examen, más 1 punto de notas de clase en esa parte. Si alguien implementa una solución pero no es más rápida que "cifras.exe", podría valer como uno de los ejercicios de la tercera práctica (también, después de una entrevista). **Máximo: 2 personas por grupo.**

El juego del BALON DE FUTBOL **RETO RESUELTO**

Descripción del problema
Solución (por Bruno Recasens)
Reto: Diseñar, implementar y documentar un programa que resuelva el problema.
Evaluación: Tras la correspondiente entrevista con el profesor, para el primero (alumno o grupo) que resuelva el problema la evaluación si se supera el reto será: un 10 en uno de los problemas de la tercera práctica de la asignatura, un 10 en un ejercicio correspondiente del examen, más 1 punto de notas de clase en esa parte. **Máximo: 2 personas por grupo.**

OLIMPIADAS REGIONALES DE PROGRAMACION

Normas
Para realizar esta práctica los alumnos deberán participar en las II Olimpiadas Regionales de Programación, y/o en el Curso de Promoción Educativa de preparación.

Evaluación

La realización de esta práctica no es obligatoria para aprobar la asignatura.
La nota obtenida se sumará a la Nota Final de la asignatura, siempre que se superen los mínimos en las notas de examen y de prácticas (ver sección Evaluación).
Para los alumnos que realicen el Curso de Preparación para el Concurso Internacional de la ACM para Estudiantes Universitarios, se sumará 0,5 puntos.
Para los alumnos que participen en las II Olimpiadas Regionales de Programación, resolviendo por lo menos 1 problema, se sumará 0,5 puntos.

SEMINARIO DE ESPECIFICACIONES FORMALES

Notas de la práctica
(alumnos que han entregado ficha)

Guión: PDF, Word97
Programa usado: Maude
Especificaciones de ejemplo: Natural, Letra, Pila, Ejemplo

Fechas
Realización: 9/12/2003 y 10/12/2003, en horas de prácticas (laboratorio 1/1)
Entrega: 19/12/2003

Normas
Esta práctica se realizará individualmente o en grupos de 2.
Para evaluar la práctica se requiere que los miembros del grupo asistan al seminario correspondiente (martes 9 de diciembre y miércoles 10) realizando por lo menos el primer ejercicio.
Se pasará lista al final del seminario.
Todos los ficheros de la práctica deberán entregarse en papel, antes del 19 de diciembre, y deberán estar accesibles dentro de una de las cuentas de los alumnos del grupo.
Revisión de la práctica: en horarios normales de tutorías.

Evaluación
La realización de esta práctica no es obligatoria para aprobar la asignatura.
Los alumnos con nota mayor o igual a 5 (sobre 10) no deberán realizar en el examen parcial o final los ejercicios correspondientes al tema 1, parte 1. La nota obtenida en esta práctica se guardará como la nota de los ejercicios del examen correspondiente.
Ver más detalles de la práctica en la guía del seminario.

TUTORIAS

Miércoles: 10:30-13:30

Jueves: 10:30-13:30

Lugar: Despacho E-20 (3ª planta, Fac. Informática)

EXAMENES

Primer Parcial Febrero-04
Fecha: sábado 28/Febrero/2004
Hora: 10:00 am
Lugar: Aulario General, Aulas 0.05 y 0.06
Duración aprox.: 3 horas
Modalidad: Teórico/Práctico
Observaciones: Se aconseja traer calculadora y DNI al examen
Notas del Examen
Examen

EVALUACION

- La **Nota Final** será un promedio entre la **Nota Total de Prácticas** y la **Nota Total del Examen**. La Nota Total de Prácticas contará un 35% y la de Examen un 65%. Ambas partes deben estar aprobadas por separado.
- La Nota Final podrá incrementarse hasta en 1 punto, con la realización de **actividades de carácter optativo** (ver sección Prácticas Optativas).
- La **Nota Total de Prácticas** será un promedio de la nota obtenida en cada una de las dos/tres prácticas de la asignatura.
  - La **Nota Total de Prácticas** podrá incrementarse hasta en 1 punto (sobre 10), con la realización de alguna parte opcional.
  - Para calcular la **Nota Total de Prácticas**, cada una de las dos/tres partes (por separado) debe tener como mínimo un 4 sobre 10. En otro caso las prácticas se consideran no aprobadas.
  - La **Nota Total de Prácticas** se guarda para convocatorias posteriores si es mayor o igual que 5 sobre 10.
- La **Nota Total del Examen** se obtendrá promediando la **Nota del Primer** y el **Segundo Parcial**.
  - El **Primer Parcial** corresponde al bloque de "Estructuras de Datos" y contará un 40%. El **Segundo Parcial** corresponde al bloque de "Análisis y Diseño de Algoritmos" y contará un 60%.
  - La Nota de cada parcial puede incrementarse hasta en 1 punto con la realización de **Ejercicios en Clase**, siempre que sea mayor o igual que 4 sobre 10.
  - Los alumnos que no aprueben el Primer Parcial deberán realizar el **Examen Final** (con toda la asignatura), y la nota obtenida será la **Nota Total del Examen**.
  - Para calcular la **Nota Total del Examen** se requiere que la nota de cada parte sea

como mínimo de 4 sobre 10 (tanto en los Parciales como en el Examen Final). En otro caso el examen se considera no aprobado.

- La **Nota Total del Examen** se guarda para convocatorias posteriores si es mayor o igual que 5 sobre 10 (tras sumar las Notas de Clase). Si el Primer Parcial está aprobado pero no el Segundo, no se guarda la nota para las convocatorias de Septiembre y Diciembre.

BIBLIOGRAFIA

Texto Guía

- **García Mateos, G., Giménez Cánovas, D., Cervera López, J., Marín Pérez, N.:** Algoritmos y Estructuras de Datos. Diego Marín, 2003.

Básica

- **Aho, A.V.; Hopcroft, J.E.; Ullman, J.D.:** *Estructura de datos y algoritmos.* Addison-Wesley, 1988.
- **Brassard, G.; Bratley, P.:** *Fundamentos de Algoritmia.* Prentice-Hall, 1998.
- **Weiss, M.A.:** *Estructuras de datos y algoritmos.* Addison-Wesley, 1995.
- **Collado, M; Morales, R.; Moreno, J.J.:** *Estructuras de datos. Realización en Pascal.* Díaz de Santos, 1987.
- **Baase, S.; Van Gelder, A.:** *Computer Algorithms. Introduction to Design and Analysis.* Addison-Wesley, 2000.
- **Horowitz, E.; Sahni, S.:** *Fundamentals of Data Structures.* Computer Science Press, 1976.
- **Wirth, N.:** *Algoritmos y estructura de datos.* Prentice-Hall, 1987.

Complementaria

- **Heileman, G.L.:** *Estructuras de Datos, Algoritmos y Programación orientada a Objetos.* McGraw-Hill, 1997.
- **Hoorobeek, I.V.:** *Algebraic Specifications. From Many-Sorted Algebras to a Practical Specification Language.* K.U. Leuven, Dept. of Computer Science, 1985.
- **Joyanes, L.; Zahonero, I.:** *Estructura de datos.* McGraw-Hill, 1998.
- **Mehlhorn, K.:** *Data Structures and Algorithms.* Springer-Verlag, 1984.
- **Peña, R.:** *Diseño de programas.* Formalismo y abstracción. Prentice-Hall, 1997.

On-line

- **Ginés García Mateos:** Exámenes de convocatorias anteriores (con soluciones, a partir de 2001). Diciembre/2003, Septiembre/2003, Marzo/2003, Diciembre/2002, Septiembre/2002, Marzo/2002, Marzo- Septiembre- Diciembre/2001. Marzo- Septiembre- Diciembre/2000. Marzo- Septiembre- Diciembre/1999.
- **Dictionary of Algorithms and Data Structures**
- **Algorithms Animation (AlgAnim)**

Facultad de Informática. Despacho E-20.
Campus de Espinardo. Universidad de Murcia.
30071 Murcia (SPAIN)
Teléfono: +34 968 36 46 08
Fax: +34 968 36 41 51
E-mail: ginesgm@um.es

Bibliografía

"Y, ¿para qué vale un libro", pensó Alicia,
"que no tiene dibujos ni diálogos?"
Lewis Carroll, Alicia en el país de las maravillas

[Aho'74] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman. *The design and analisys of computer algorithms*. Adison-Wesley, 1974.

[Aho'83] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman. *Data structures and algorithms*. Adison-Wesley, 1983.

[Aho'88] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman. *Estructuras de datos y algoritmos*. Addison-Wesley, 1988.

[Baase'83] Sara Baase. *Computer algorithms. Introduction to design and analysis*. Addison-Wesley, 1983.

[Baase'00] Sara Baase, Allen Van Gelder. *Computer Algorithms. Introduction to Design and Analysis*. Addison-Wesley, 2000.

[Ben-Ari'98] Mordechai Ben-Ari. *Constructivism in Computer Science Education*. 29th SIGCSE Technical Symposium on Computer Science Education, Atlanta (USA), 1998.

[Berlín'03] *Realising the European Higher Education Area*. Conference of European Ministers responsible for Higher Education, Berlín (Alemania), 2003.
URL: `http://bologna-berlin2003.de/`

[Bloom'56] B.S. Bloom, D.R. Krathwohl. *Taxonomy of Educational Objectives. Handbook I: Cognitive Domain*. New York: David McKay Company Inc., 1956.

[Bloom'64] B.S. Bloom, D.R. Krathwohl, B.B. Masia. *Taxonomy of Educational Objectives. Handbook II: Affective Domain*. New York: David McKay Company Inc., 1964.

[Bolonia'99] *The Bologna declaration on the european space for higher education: an explanation*. Confederation of EU Rectors' Conferences and the Association of European Universities (CRE), 1999.
URL: `http://europa.eu.int/comm/education/socrates/erasmus/bologna.pdf`

[Brassard'90] Gilles Brassard, Paul Bratley. *Algorítmica. Concepción y análisis*. Masson, 1990.

[Brassard'96] Gilles Brassard, Paul Bratley. *Fundamentals of algorithmics*. Prentice-Hall, 1996.

[Brassard'97] Gilles Brassard, Paul Bratley. *Fundamentos de algoritmia*. Prentice-Hall, 1997.

[Campos'95] Javier Campos Laclaustra. *Estructuras de datos y algoritmos*. Colección Textos Docentes, Prensas Universitarias de Zaragoza, 1995.

[CareerSpace'01] *Curriculum development guidelines. New ICT curricula for the 21st century: designing tomorrow's education*. The Career Space Consortium, 2001.
URL: `http://www.career-space.com/cdguide/index.htm`

[CC'1991] *Computing Curricula 1991. Report of the ACM/IEEE-CS Joint Curriculum Task Force*. IEEE Computer Society Press, Silver Spring, 1991.
URL ACM: `http://acm.org/education/curr91`
URL IEEE: `http://www.computer.org/education/cc1991`

[CC'2001] *Computing Curricula 2001 (Computer Science). Report of the ACM/IEEE-CS Joint Curriculum Task Force*. IEEE Computer Society Press, Silver Spring, 2001.
URL ACM: `http://www.acm.org/sigcse/cc2001`
URL IEEE: `http://www.computer.org/education/cc2001`

[CMU'02] *Bachelor of Science program in Computer Science*. Carnegie Mellon University, 2002.
URL: `http://www.csd.cs.cmu.edu/education/bscs`

[Collado'87] Manuel Collado Machuca, Rafael Morales Fernández, Juan José Moreno Navarro. *Estructuras de datos, realización en Pascal*. Ediciones Díaz de Santos, 1987.

[Cormen'90] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[Cormen'01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms, second edition*. The MIT Press, 2001.

[Davis'83] Martin D. Davis, Elaine J. Weyuker. *Computability, Complexity, and Languages*. Academis Press, 1983.

[Deitel'03] H.M. Deitel, P.J. Deitel. *Cómo programar en C++*. Prentice Hall, 2003.

[Doménech'99] F. Doménech Betoret. *Proceso de enseñanza/aprendizaje universitario*. Universitat Jaume I, Castellón, 1999.

[Dromey'82] R.G. Dromey. *How to solve it by computer*. Prentice Hall, 1982.

[Drozdek'01] Adam Drozdek. *Data Structures and Algorithms in Java*. Brooks/Cole, 2001.

[Fernández'02] José Luis Fernández Alemán. *Proyecto Docente*. Universidad de Murcia, 2001. Presentado en concurso de acceso a plaza de Profesor Titular de Escuelas Universitarias del Área de Lenguajes y Sistemas Informáticos, con docencia en Programación.

[Franch'94] Manuel Franch Gutiérrez. *Estructuras de datos, Especificación, diseño e implementación*. Ediciones UPC, 1994.

[Galve'93] Javier Galve, Juan C. González, Ángel Sánchez, J. Ángel Velázquez. *Algorítmica, diseño y análisis de algoritmos Funcionales e Imperativos*. Ra-Ma, 1993.

[García'01] Jesús J. García Molina. *"¿Es conveniente la orientación a objetos en un primer curso de programación?"*. En: VII Jornadas de Enseñanza Universitaria de la Informática (JENUI'2001). Palma de Mallorca, Julio, 2001.
URL: `http://dis.um.es/~jmolina/publi.html`

[García-Valcárcel'01] Ana García-Valcárcel Muñoz-Repiso. *"La función docente del profesor universitario"*. En: *Didáctica universitaria*. Editorial La Muralla, S.A., 2001.

[García'03] Ginés García Mateos, Joaquín Cervera López, Norberto Marín Pérez, Domingo Giménez Cánovas. *Algoritmos y Estructuras de Datos, Volumen I: Estructuras de Datos*. Diego Marín, Colección textos-guía, 2003.

[Garey'79] Michael R. Garey, David S. Johnson. *Computers and intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[Giménez'90] Domingo Giménez Cánovas. *Proyecto Docente*. Universidad de Murcia, 1990.
Presentado en concurso de acceso a plaza de Profesor Titular de Escuelas Universitarias del Área de Ciencias de la Computación e Inteligencia Artificial, con docencia en Algoritmos y Estructuras de Datos.

[Giménez'97] Domingo Giménez Cánovas. *Proyecto Docente e investigador*. Universidad de Murcia, 1997.
Presentado en concurso de acceso a plaza de Profesor Titular de Universidad del Área de Ciencias de la Computación e Inteligencia Artificial, con docencia en Algorítmica.

[Giménez'03] Domingo Giménez Cánovas, Joaquín Cervera López, Ginés García Mateos, Norberto Marín Pérez. *Algoritmos y Estructuras de Datos, Volumen II: Algoritmos*. Diego Marín, Colección textos-guía, 2003.

[Gómez'01] Mercedes Gómez Albarrán. *"Metodología basada en descomposición funcional y orientación a objetos en la introducción a la programación"*. En: VII Jornadas de Enseñanza Universitaria de la Informática (JENUI'2001). Palma de Mallorca, Julio, 2001.

[Gonnet'91] Gaston H. Gonnet, Ricardo Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison Wesley, 1991.

[Gonzalo'62] Leónides Gonzalo Calavia. *Historia de la pedagogía*. Editorial Tibi Magister, 1962.

[Gonzalo'98] Julio Gonzalo Arroyo, Miguel Rodríguez Artacho. *Esquemas algorítmicos: enfoque metodológico y problemas resueltos*. Universidad Nacional de Educación a Distancia, 1998.

[GuíaFIUM'03] *Guía académica de la Facultad de Informática, curso 2003/04*. Universidad de Murcia, 2003.
URL: `http://www.um.es/informatica/alumnos/guias.htm`
URL: `http://www.fi.um.es/estudios`

[GuíaUM'03] *Guía de la Universidad de Murcia, curso 2003/04*. Universidad de Murcia, 2003.
URL: `http://www.um.es/siu/guias.htm`

[Harel'92] David Harel. *Algorithms. The Spirit of Computing(2nd Edition)*. Addison-Wesley, 1992.

[Heileman'98] Gregory L. Heileman. *Estructuras de Datos, Algoritmos y Programación Orientada a Objetos*. McGraw-Hill, 1998.

[Hernández'01] Roberto Hernández, Juan Carlos Lázaro, Raquel Dormido, Salvador Ros. *Estructuras de Datos y Algoritmos*. Prentice Hall, 2001.

[HistoriaUM'98] *Universidad de Murcia. Pasado, presente y futuro*. Unidad de Comunicación e Imagen, Vicerrectorado de Extensión Cultural y Proyección Universitaria, Universidad de Murcia, 1998.
URL (extracto): `http://www.um.es/historiaUMU`

[Hopcroft'79] John E. Hopcroft, Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addisson-Wesley, 1979.

[Hopcroft'01] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. *Introducción a la Teoría de Autómatas, Lenguajes y Computación*. Addisson-Wesley, 2001.

[Horowitz'78] Ellis Horowitz, Sartaj Shani. *Fundamentals of Computer Algorithms*. Pitman, 1978.

[Horowitz'82] Ellis Horowitz, Sartaj Shani. *Fundamentals of Data Structures*. Computer Science Press, 1982.

[Horowitz'95] Ellis Horowitz, Sartaj Shani, D. Mehta. *Fundamentals of Data Structures in C++*. W.H. Freeman & Co., 1995.

[Humphrey'01] Watt S. Humphrey. *Introducción al proceso software personal*. Pearson Educación, 2001.

[Johnson'90] David S. Johnson. *A Catalog of Complexity Classes*, in Handbook of Theoretical Computer Science, Volume A, Algorithms and Complexity, Jan van Leeuwen ed., pp 68-161. Elsevier, 1990.

[Joyanes'00] Luis Joyanes Aguilar *Programación en C++. Algoritmos, estructuras de datos y objetos*. McGraw-Hill, 2000.

[Kernighan'91] Brian W. Kernighan, Dennis M. Ritchie. *El lenguage de programación C*. Prentice-Hall, 1991.

[Kignston'90] Jeffrey H. Kingstone. *Algorithms and data structures: design, correctness, analysis*. Addisson-Wesley, 1990.

[Knuth'85] Donald E. Knuth. *El arte de programar ordenadores. Vol 1: algoritmos fundamentales*. Reverté, 1985.

[Knuth'87] Donald E. Knuth. *El arte de programar ordenadores. Vol 3: clasificación y búsqueda*. Reverté, 1987.

[Knuth'97] Donald E. Knuth. *The Art of Computer Programming. Vol 1: Fundamental Algorithms, Third Edition*. Addison-Wesley, 1997.

[Kopec'99] Danny Kopec, Richard Close, Jim Aman. *"How should data structures and algorithms be taught"*. ITiCSE'99, Cracow, Polonia, 1999.

[Kozen'91] Dexter C. Kozen. *The Design and Analysis of Algorithms*. Springer, 1991.

[Kruse'89] Robert L. Kruse. *Estructura de datos y diseño de programas*. Prentice-Hall, 1989.

[Langsam'97] Yedidyah Langsam, Moshe J. Augenstein, Aaron M. Tenenbaum. *Estructuras de datos con C y C++*. Prentice-Hall, 1997.

[Lewis'81] Harry R. Lewis, Christos H. Papadimitriou. *Elements of the theory of computation*. Prentice-Hall, 1981.

[Levitin'03] Anany Levitin. *The Design and Analysis of Algorithms*. Addison Wesley, 2003.

[Linger'79] R.C. Linger, H.D. Mills, Bernard I. Witt. *Structured programming - theory and practice*. Addison Wesley, 1979.

[López'03] Pedro Enrique López de Teruel Alcolea. *Proyecto Docente*. Universidad de Murcia, 2003. Presentado en concurso de acceso a plaza de Profesor Titular de Escuelas Universitarias del Área de Arquitectura y Tecnología de Computadores, con docencia en Fundamentos de Computadores e Introducción a los Computadores.

[LOU'01] *Ley Orgánica 6/2001, de 21 de diciembre, de Universidades*. B.O.E. de 26 de diciembre de 2001, Serie A, Num. 45-13.

[LRU'83] *Ley Orgánica 11/1983, de 25 de agosto, de Reforma Universitaria*. B.O.E. de 1 de septiembre de 1993.

[Manber'89] Udi Manber. *Introduction to Algorithms. A Creative Approach*. Addison-Wesley, 1989.

[Marcelo'01] Carlos Marcelo García. *"El proyecto docente: una ocasión para aprender"*. En: *Didáctica universitaria*. Editorial La Muralla, S.A., 2001.

[Marín'90] Norberto Marín Pérez. *Proyecto Docente*. Universidad de Murcia, 1990.
Presentado en concurso de acceso a plaza de Profesor Titular de Escuelas Universitarias del Área de Lenguajes y Sistemas Informáticos, con docencia en Algoritmos y Estructuras de Datos.

[Martí'04] Narciso Martí, Yolanda Ortega, Alberto Verdejo. *Estructuras de datos y métodos algorítmicos: ejercicios resueltos; 1ª edición*. Prentice Hall, 2004.

[Martínez'01] Nicolás Martínez Valcárcel *El Proyecto Docente*. Instituto de Ciencias de la Educación. Universidad de Murcia, 2001.

[MCYT'03] *Relación de indicadores de la Sociedad de la Información en España y varios países de la OCDE, 1995-2003* Ministerio de Ciencia y Tecnología, 2004.
URL: `http://www6.mcyt.es/indicadores/tic/indice_tic.htm`

[MECD'03] *La Integración del Sistema Universitario Español en el Espacio Europeo de Enseñanza Superior. Documento-Marco*. Ministerio de Educación, Cultura y Deporte, febrero de 2003.
URL: `http://www.um.es/estructura/rectorado/rector/integracion-eees.html`

[Mehlhorn'84] Mehlhorn. *Data Structures and Algorithms. Vol 1-3*. Springer, 1984.

[MemoriaUM'02] *Memoria Académica del Curso 2002/2003*. Universidad de Murcia, Claustro Universitario, 2003.

[Meyer'88] Bertrand Meyer. *Object Oriented Software Construction*. Prentice Hall, 1988.

[Meyer'94] Bertrand Meyer. *Towards an object-oriented curriculum*. Journal of Object-Oriented Programming, pp 76–81, March, 1994.

[Meyer'99] Bertrand Meyer. *Construcción de Software Orientado a Objetos. Segunda edición*. Prentice Hall, 1999.

[deMiguel'99] Mario de Miguel Díaz. *Los objetivos formativos en la enseñanza universitaria*. I Simposium Iberoamericano sobre Didáctica Universitaria, Santiago de Compostela, diciembre de 1999.

[Montoya'01] Francisco Montoya Dato. *Proyecto Docente y Reseña de Actividad Investigadora*. Universidad de Murcia, 2001.
Presentado en concurso de acceso a plaza de Profesor Titular de Universidad del Área de Lenguajes y Sistemas Informáticos, con docencia en Metodología de la Programación.

[Nievergelt'93] Jurg Nievergelt, Klaus H. Hinrichs. *Algorithms and Data Structures: with Applications to Graphics and Geometry*. Prentice Hall, 1993.

[Ortín'02] Maria José Ortín Ibañez. *Proyecto Docente*. Universidad de Murcia, 2002.
Presentado en concurso de acceso a plaza de Profesor Titular de Escuelas Universitarias del Área de Lenguajes y Sistemas Informáticos, con docencia en Bases de Datos.

[Peña'98] Ricardo Peña Marí. *Diseño de Programas. Formalismo y Abstracción*. Prentice-Hall, 1998.

[Pérez'97] L. Pérez *La evaluación dentro del proceso enseñanza-aprendizaje*. Academia para el desarrollo de la educación, n.º 11 (septiembre-octubre 1997).
URL: `http://www.hemerodigital.unam.mx/ANUIES/ipn/academia/11/sec_4.htm`

[Praga'01] *Towards the European Higher Education Area*. Communiqué of the meeting of European Ministers in charge of Higher Education in Prague.
URL: `http://europa.eu.int/comm/education/prague.pdf`

[Rabhi'99] Fethi Rabhi, Guy Lapalme. *Algorithms, a Functional Programming Approach*. Addison-Wesley, 1999.

[Ramsden'92] Paul Ramsden. *Learning to Teach in Higher Education* London: Routledge, 1992.

[Rodríguez'00] S. Rodríguez. *La evaluación del aprendizaje en los estudiantes*. Actas del Congreso Internacional de Docencia Universitaria e Innovación, Barcelona, 2000.

[Rumbaugh'91] James R. Rumbaugh, Michael R. Blaba, William Lorensen, Frederick Eddy, William Premerlani. *Object oriented modeling and design*. Prentice Hall, 1991.

[Sedgewick'88] Robert Sedgewick. *Algorithms*. Addison-Wesley, 1988.

[Sedgewick'92] Robert Sedgewick. *Algorithms in C++*. Addison-Wesley, 1992.

[SFPlecc'02] *Lección magistral*. Servei de Formació Permanent, Universidad de Valencia, 2002.
URL: `http://www.uv.es/~sfp/pdi/dos.doc`

[SFPeval'02] *Evaluación de estudiantes*. Servei de Formació Permanent, Universidad de Valencia, 2002.
URL: `http://www.uv.es/~sfp/pdi/tres.doc`

[Skiena'98] Steven S. Skiena. *The algorithm design manual*. Springer-Verlag, 1998.

[Sorbona'98] *Joint declaration on harmonisation of the architecture of the european higher education system*. By the four ministers in charge for France, Germany, Italy and the United Kingdom. 25 de mayo de 1998.
URL: `http://www.bologna-berlin2003.de/pdf/Sorbonne_declaration.pdf`

[Stroustrup'98] Bjarne Stroustrup. *El lenguaje de programación C++*. Addison Wesley, 1998.

[Tenenbaum'88] Aaron M. Tenenbaum, Moshe J. Augenstein. *Estructuras de datos en Pascal*. Prentice-Hall, 1988.

[TitulUM'03] *Guía de titulaciones de la Universidad de Murcia, curso 2003/04*. Universidad de Murcia, 2003.
URL: `http://www.um.es/infosecundaria/titulaciones`

[Troya'84] José María Troya Linero. *Análisis y diseño de algoritmos*. VI Escuela de Verano de Informática. AEIA, 1984.

[UM-lab'03] *La inserción laboral de los titulados de la Universidad de Murcia*. Vicerrectorado de Calidad y Convergencia Europea, Unidad para la Calidad, Universidad de Murcia, 2003.

[UNESCO'94] *A modular curriculum in Computer Science*. United Nations Educational, Scientific and Cultural Organization (UNESCO) and International Federation for Information Processing (IFIP), 1994.
URL: `http://www.unesco.org/education/pdf/54_63.pdf`

[UNESCO'98] *Declaración mundial sobre la Educación Superior en el Siglo XXI*. UNESCO, París, 1998.

[Valcárcel'01] María Victoria Valcárcel. *Presentación y explicación de contenidos: la clase magistral*. Departamento de Didáctica de las Ciencias Experimentales, Universidad de Murcia, 2001.

[Weiss'93] Mark Allen Weiss. *Data structures and algorithm analysis in C*. Benjamin Cummings, 1993.

[Weiss'95] Mark Allen Weiss. *Estructuras de datos y algoritmos*. Addison-Wesley, 1995.

[Weiss'00] Mark Allen Weiss. *Estructuras de datos en Java*. Addison-Wesley, 2000.

[Wilder'71] R.L. Wilder *El nuevo profesor universitario*. Enseñanza universitaria. Reforma y métodos. Morris, W. (ed.), México, Pax, 1971.

[Wirth'80] Niklaus Wirth. *Algoritmos+Estructuras de datos=Programas*. Ediciones del Castillo, 1980.

[Wirth'87] Niklaus Wirth. *Algoritmos y estructuras de datos*. Prentice-Hall, 1987.

[Wood'87] Derick Wood. *Theory of Computation.* John Wiley & Sons, 1987.

[Zabalza'93] M. A. Zabalza. *Criterios didácticos para elaborar planes de estudios.* III Jornadas Nacionales de Didáctica Universitaria, Las Palmas, Servicio de Publicaciones de la Universidad, 1993.

[Zabalza'99] M. A. Zabalza. *La calidad de la docencia Universitaria.* Conferencia nº 3. Symposium Iberoamericano de Didáctica Universitaria. Santiago de Compostela, 1999.