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.

Algoritmos y Estructuras de Datos

PARTE II: ALGORÍTMICA (o ALGORITMIA)

Tema 0. Introducción

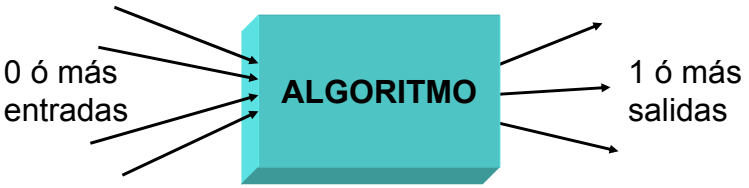
0.1. Definición y propiedades.

0.2. Análisis y diseño de algoritmos.

0.3. Heurísticas para una buena programación.

0.1. Definición y propiedades.

- **Algoritmo:**
  Conjunto de reglas para resolver un problema.
- **Propiedades**
  - **Definibilidad:** El conjunto debe estar bien definido, sin dejar dudas en su interpretación.
  - **Finitud:** Debe tener un número finito de pasos que se ejecuten en un tiempo finito.



0.1. Definición y propiedades.

- **Algoritmos deterministas:** Para los mismos datos de entrada se producen los mismos datos de salida.
- **Algoritmos no deterministas:** Para los mismos datos de entrada pueden producirse diferentes de salida.

- **ALGORITMIA:** Ciencia que estudia técnicas para construir algoritmos eficientes y técnicas para medir la eficacia de los algoritmos.
- **Objetivo**: Dado un problema concreto encontrar la mejor forma de resolverlo.

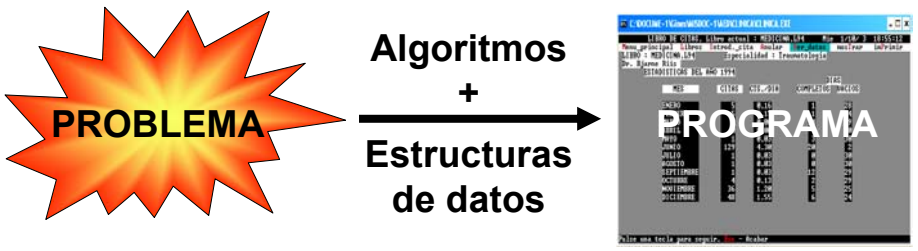
0.1. Definición y propiedades.

Recordamos:
Objetivo de la asignatura

Ser capaz de **analizar**, **comprender** y **resolver** una amplia variedad de **problemas** de programación, diseñando soluciones **eficientes** y de **calidad**.

Pero **ojo**, los algoritmos no son el único componente en la resolución de un problema de programación.

0.1. Definición y propiedades.



Algoritmos + Estructuras de Datos = Programas

- **Estructura de datos:** Parte estática, almacenada.
- **Algoritmo:** Parte dinámica, manipulador.

0.1. Definición y propiedades.

Resolver problemas

¿Cómo se resuelve un problema?

¿Cuándo se dice que la solución es eficiente y de calidad?

¿Qué clase de problemas?

0.1. Definición y propiedades.

ARQUITECTO	INFORMÁTICO
1. Estudio de viabilidad, análisis del terreno, requisitos pedidos, etc.	1. **Análisis** del problema
2. Diseñar los planos del puente y asignar los materiales.	2. **Diseño** del programa (alg. y estr.)
3. Poner los ladrillos de acuerdo con los planos.	3. **Implementación** (programación)
4. Supervisión técnica del puente.	4. **Verificación** y pruebas

0.1. Definición y propiedades.

MÉTODO CIENTÍFICO		INFORMÁTICO
1.Observación.	⟷	1. **Análisis** del problema
2.Hipótesis.	⟷	2. **Diseño** del programa (alg. y estr.)
3.Experimentación.	⟷	3. **Implementación** (programación)
4.Verificación.	⟷	4. **Verificación** y pruebas

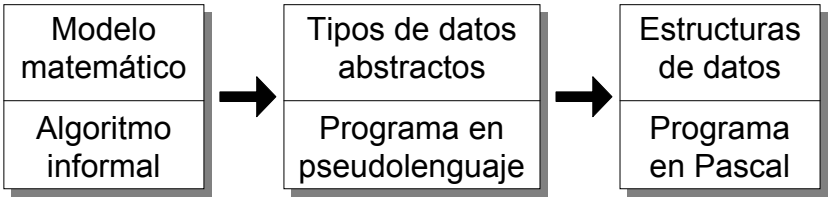
0.1. Definición y propiedades.

Otras ideas...

- **Refinamiento por pasos sucesivos.**
  - Escribir la estructura de la solución en pseudocódigo, de manera muy genérica.
  - Especificar los pasos de forma cada vez más detallada, y precisa.
  - Repetimos el refinamiento hasta llegar a una implementación.

0.1. Definición y propiedades.

- **Proceso de resolución propuesto por Aho.**



- Más en las asignaturas de Ingeniería del Software...

0.2. Análisis y diseño de algoritmos.

ALGORITMIA = ANÁLISIS + DISEÑO

- **Análisis de algoritmos:** Estudio de los recursos que necesita la ejecución de un algoritmo.
- No confundir con análisis de un problema.

- **Diseño de algoritmos:** Técnicas generales para la construcción de algoritmos.
- Por ejemplo, divide y vencerás: dado un problema, divídelo, resuelve los subproblemas y luego junta las soluciones.

0.2. Análisis y diseño de algoritmos.

- **Análisis de algoritmos.** Normalmente estamos interesados en el estudio del tiempo de ejecución.
- Dado un algoritmo, usaremos las siguientes notaciones:
  - $t(..)$: Tiempo de ejecución del algoritmo.
  - $O(..)$: Orden de complejidad.
  - $o(..)$: O pequeña del tiempo de ejecución.
  - $\Omega(..)$: Cota inferior de complejidad.
  - $\Theta(..)$: Orden exacto de complejidad.

0.2. Análisis y diseño de algoritmos.

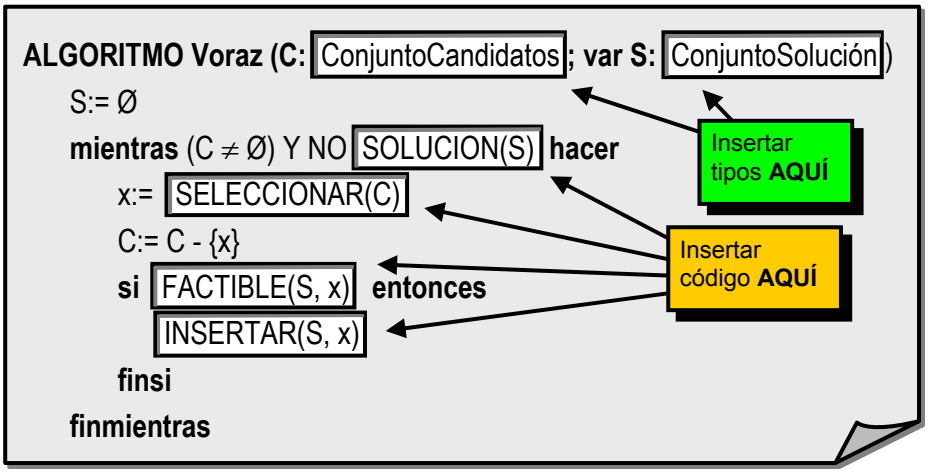
- **Ejemplo.** Analizar el tiempo de ejecución y el orden de complejidad del siguiente algoritmo.

```
Hanoi (N, A, B, C: integer)
    if  N=1 then
        Mover (A, C)
    else begin
        Hanoi (N-1, A, C, B)
        Mover (A, C)
        Hanoi (N-1, B, A, C)
    end
```

- Mecanismos:
  - Conteo de instrucciones.
  - Uso de ecuaciones de recurrencia.

0.2. Análisis y diseño de algoritmos.

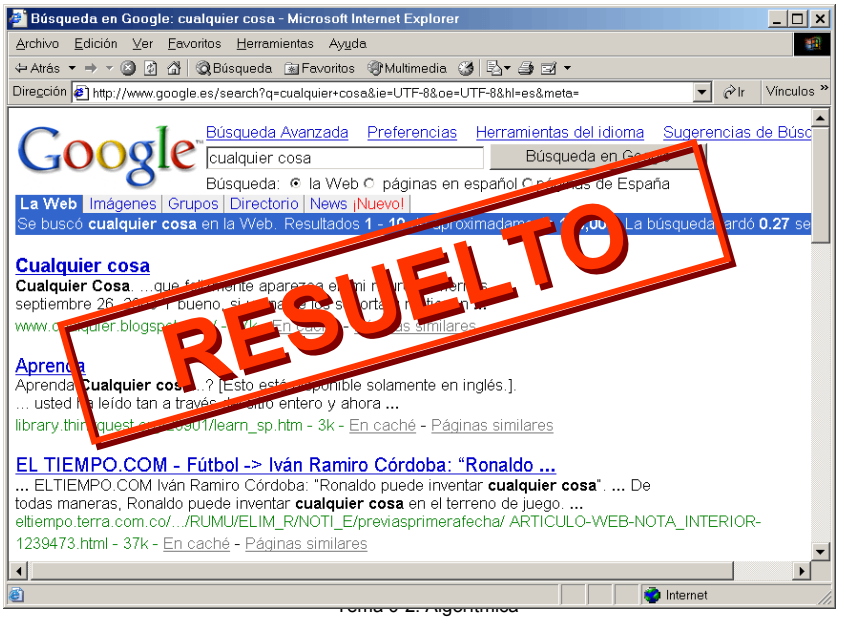
- **Diseño de Algoritmos.** Técnicas generales, aplicables a muchas situaciones.
- **Esquemas algorítmicos.** Ejemplo:



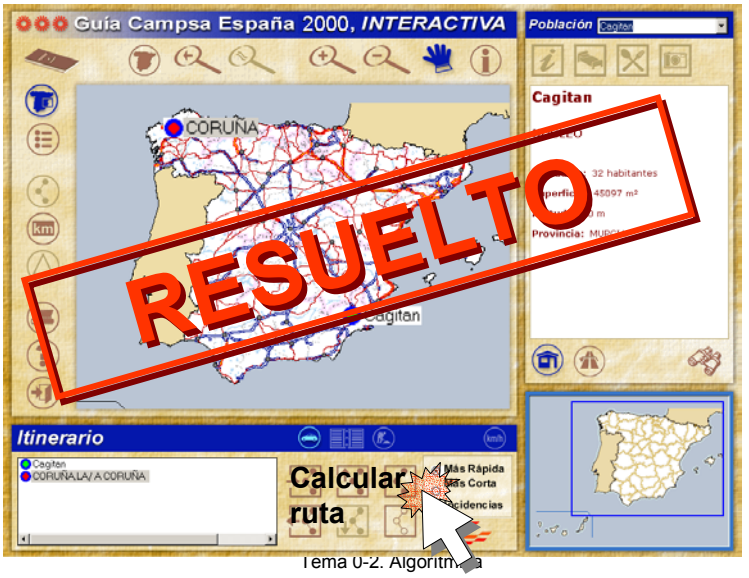
0.2. Análisis y diseño de algoritmos.

¿Qué clase de problemas?

0.2. Análisis y diseño de algoritmos.



0.2. Análisis y diseño de algoritmos.
Planificador de rutas



0.2. Análisis y diseño de algoritmos.

- **EL JUEGO DE LAS CIFRAS.**
  Dado un conjunto de 6 enteros, encontrar la forma de conseguir otro entero, utilizando las operaciones de suma, resta, producto y división entera (y sin usar cada número más de una vez).

0.2. Análisis y diseño de algoritmos.

0.2. Análisis y diseño de algoritmos.

0.2. Análisis y diseño de algoritmos.

- **Caso 1.** 1 1 5 2 10 7
  990
- **Caso 2.** 6 8 10 9 4 75
  835
- Implementación muy rápida.

0.2. Análisis y diseño de algoritmos.

- **RETO.** Implementar un programa para resolver el problema, más rápido que la versión del profesor, y que no pierda ninguna solución.
- **RECOMPENSA.**
  - Un 10 en la tercera práctica de la asignatura (diseño de algoritmos).
  - Un 10 en el ejercicio correspondiente del examen.
  - Más 1 punto de notas de clase.

0.2. Análisis y diseño de algoritmos.

- **PROBLEMA DEL BALÓN DE FÚTBOL.**
  Se pinta una región (conexa) del balón colorando de verde algunos hexágonos o pentágonos del balón.

  La región se describe moviéndose por el contorno de la misma, poniendo 1 ó 2 según el vértice tenga al lado 1 ó 2 trozos de verde.

  A partir de la descripción, calcular el número de trozos verdes, blancos y negros.

0.2. Análisis y diseño de algoritmos.

- **Caso 1**. 1, 1, 1, 1, 1

  Resultado: 1 verde
  11 negros, 20 blancos

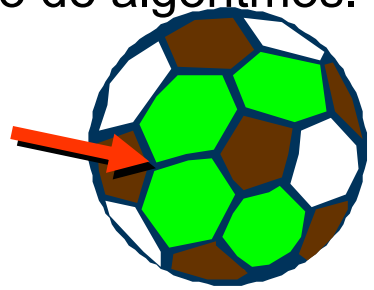
- **Caso 2**. 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1

  Resultado: 3 verdes
  11 negros, 18 blancos

0.2. Análisis y diseño de algoritmos.

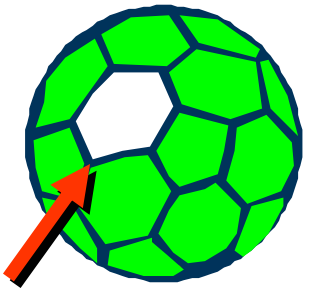
- **Caso 3**. 2, 1, 1, 2, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 2, 1, 1

  Resultado: 4 verdes
  12 negros, 16 blancos



- **Caso 4**. 2, 2, 2, 2, 2, 2

  Resultado: 31 verdes
  0 negros, 1 blanco



0.2. Análisis y diseño de algoritmos.

- **RETO.** Implementar un programa para resolver el problema.
- **RECOMPENSA.** Para el primero que lo resuelva:
  - Un 10 en uno de los problemas de la tercera práctica.
  - Un 10 en el ejercicio correspondiente del examen.
  - Más 1 punto de notas de clase.

0.3. Consejos para una buena programación.

1. **Proceso de análisis/diseño**. No empezar tecleando código como locos/as.
2. **Usar abstracciones**, respetando los dos principios básicos:
   - **Encapsulación**: las funciones relacionadas deben ir juntas (clases, módulos, paquetes, etc.).
   - **Ocultación de la implementación**: Los aspectos de implementación no son visibles fuera del módulo, clase, etc.

0.3. Consejos para una buena programación.

3. **Reutilizar programas, librerías, tipos, etc. existentes.** Y programar pensando en la posible reutilización futura. Un nuevo programa no debe partir desde cero.
4. **No resolver casos concretos, sino el problema en general.** Si no se requiere un esfuerzo adicional, el algoritmo debe resolver un caso genérico.
5. **Repartir bien la funcionalidad.** Repartir la complejidad del problema de forma uniforme. No crear procedimientos muy largos: usar subrutinas. De esta forma se mejora la **legibilidad** del código.