A. Contexto

Los sistemas de información geográfica y de planificación de rutas constituyen uno de los dominios de aplicación más interesantes para poner a prueba la potencia de los equipos informáticos más recientes. La existencia de algoritmos clásicos, como el de Dijkstra para el cálculo de caminos mínimos, no evita la necesidad de realizar un diseño muy cuidado y bien estructurado para manejar grandes volúmenes de datos. Así, una ciudad puede tener unos cuantos centenares de calles y carreteras; una región suele tener muchas docenas de ciudades; un país se compone de algunas decenas de regiones; y existen unos pocos centenares de países en el mundo. En conjunto, un mapa detallado puede tener varias decenas de millones de carreteras y lugares, sobre los cuales aplicar un algoritmo de orden cuadrático.

El objetivo de esta práctica es desarrollar el motor de un sistema de planificación de rutas capaz de gestionar varios miles de lugares, con decenas de miles de carreteras, que no están predefinidos sino que pueden cambiar en tiempo de ejecución. El uso de memoria y, sobre todo, el tiempo de ejecución serán los principales factores a considerar en el diseño y estructuración de los datos requeridos en la aplicación.

B. El Problema

Analizar, diseñar e implementar el motor interno de un sistema de planificación de rutas. El programa recibirá en la entrada una serie de comandos, que se leerá siempre de la entrada estándar, produciendo el resultado en la salida estándar. El programa tendrá en todo momento un mapa cargado en memoria, con la posibilidad de añadir, eliminar y consultar lugares (nodos) y carreteras (aristas). Los comandos admisibles son los siguientes:

- **Inicializar el mapa**: elimina todos los lugares y carreteras del mapa, si los hubiera. Ese será también el estado inicial del programa.
- **Insertar y eliminar lugares**: estos comandos permiten añadir y quitar lugares del mapa, respectivamente. Un lugar puede representar una plaza, un pueblo, un cruce de caminos, etc.; en general, será simplemente un nodo del grafo. Tendrá asociado un nombre y cierta información (en forma de una cadena de texto arbitraria).
- **Insertar y eliminar carreteras**: sirven para añadir o quitar carreteras del mapa, es decir, aristas del grafo. Las carreteras siempre van entre dos lugares (un origen y un destino) y tendrán asociado un coste y una información (también una cadena de texto).
- **Consultar lugares y carreteras**: dado un nombre de un lugar o una carretera, obtener la información asociada a los mismos.
- **Listar adyacentes**: dado un lugar del mapa, encontrar todos los lugares que tiene adyacentes, es decir, aquellos con los que es origen de una carretera.
- **Listar lugares**: devuelve todos los lugares del mapa listados por orden alfabético.
- **Calcular ruta**: dados dos lugares, origen y destino, encontrar el camino más corto de uno a otro pasando por carreteras del mapa.

El programa debe estar diseñado cuidadosamente para conseguir la máxima eficiencia de tiempo y de memoria, con especial hincapié en lo primero. Por ejemplo, no será admisible usar una simple lista para implementar todas las operaciones.

C. Formato de entrada y salida

La entrada será un texto ASCII. Cada línea de la entrada contendrá un comando distinto, que se expresarán con una notación funcional:

```
NombreComando(parámetro 1,parámetro 2...)
```

Cada comando tiene predefinido cuántos parámetros necesita (puede que ninguno). Cualquier línea que no se ajuste a ese formato será descartada sin más. En concreto, los co-

mandos admisibles son los siguientes (algunos pueden quedar aquí un poco imprecisos o ambiguos; se concretará su significado en los ejercicios correspondientes del juez on-line):

- **Inicializar**. Elimina todos los lugares y carreteras del mapa, liberando la memoria que hubiera sido reservada. En la entrada aparecerá:
  Inicializar()
  Y en la salida, después de ejecutarlo, aparecerá la línea:
  Mapa inicializado

- **Añadir lugar**. Añade un nuevo lugar al mapa (un nuevo nodo al grafo), con un nombre y un texto asociados. Si ya existe ese nombre de lugar, se modificará el texto asociado. El formato de entrada será:
  AñadirLugar(*nombre***,***información asociada***)**
  Obsérvese que los separadores son la coma y los paréntesis. Cualquier otro carácter puede formar parte del nombre y de la información. El formato de salida será:
  Añadido: *nombre***. Total:** *n* **lugares**
  Donde *n* es el número de lugares del mapa en ese momento.

- **Añadir carretera**. Añade al mapa una nueva carretera entre dos lugares (una nueva arista al grafo). Se supone que la carretera es de un solo sentido (es decir, el grafo es dirigido). El formato de entrada será:
  AñadirCarretera(*origen***,***destino***,***coste***,***información asociada***)**
  origen y *destino* son los nombres de los lugares de origen y destino de la carretera, respectivamente; *coste* es un natural; e *información asociada* es una cadena. Si no existe *origen* o *destino* no hace nada. Si ya existe esa carretera, se modifica el coste y la información. El formato de salida será:
  Añadido: *origen-destino***. Total:** *a* **carreteras**
  Siendo *a* el número de carreteras añadidas al mapa hasta ese punto.

- **Eliminar lugar (opcional)**. Elimina un lugar del mapa. El formato será:
  EliminarLugar(*nombre***)**
  Si existe *nombre* en el mapa, elimina ese lugar y todas las carreteras asociadas al mismo. Si no existe *nombre* no hace nada. En otro caso, la salida será:
  Eliminado: *nombre***. Total:** *n* **lugares y** *a* **carreteras**

- **Eliminar carretera (opcional)**. Elimina una carretera añadida previamente al mapa. En la entrada aparecerá:
  EliminarCarretera(*origen***,***destino***)**
  Si no existe la carretera entre *origen* y *destino* no hace nada. Si no, la salida será:
  Eliminado: *origen-destino***. Total:** *a* **carreteras**

- **Consultar lugar**. Busca la información referente a un lugar, cuyo nombre es pasado como parámetro. El formato de entrada será:
  ConsultarLugar(*nombre***)**
  Si no existe *nombre* en el mapa, escribirá una línea con:
  No encontrado: *nombre*
  En otro caso, la salida será:
  Encontrado: *nombre*
  Información: *información asociada*

- **Consultar carretera**. Busca la información referente a una carretera entre dos lugares. El formato de entrada será:
  ConsultarCarretera(*origen***,***destino***)**
  Si no existe la carretera entre *origen* y *destino* escribirá una línea con:
  No encontrado: *origen-destino*
  En otro caso, la salida será:
  Encontrado: *origen-destino*
  Coste: *coste*

Información: *información asociada*

- **Listar adyacentes**. Busca todos los lugares adyacentes a uno dado, es decir, aquellos con los que tiene una carretera de la cual es origen. El formato de entrada será:
  ListarAdyacentes(*nombre***)**
  Si no existe *nombre* en el mapa, escribirá una línea con:
  No encontrado: *nombre*
  En otro caso, la salida será:
  Encontrado: *nombre*
  Adyacentes: *adyacente1*, *adyacente2*...

- **Listar lugares (opcional)**. Lista todos los lugares del mapa. El formato de entrada es:
  ListarLugares()
  Y el formato de la salida:
  lugar1
  lugar2
  ...
  Total: *n* **lugares**
  Donde los lugares deben estar ordenados alfabéticamente, de menor a mayor.

- **Encontrar caminos mínimos**. Dados dos lugares, encuentra el camino mínimo entre los dos pasando por las carreteras del mapa. Si existe más de uno, devolverá el que pasa primero por los lugares con menor orden alfabético. El formato de entrada será:
  CalcularRuta(*origen***,***destino***)**
  El formato de la salida será:
  origen: **0**
  lugar1: *k1*
  lugar2: *k2*
  ...
  destino: *kt*
  Total: *t* **pasos**
  Donde *lugar1*, *lugar2*... son los nodos intermedios de paso; *k1*, *k2*... son las distancias acumuladas hasta cada nodo intermedio; *t* es el número de carreteras (aristas) del camino; y *kt* es el coste del camino mínimo. Si no existe ningún camino entre origen y destino, el resultado será una línea con el texto:
  No existe ningún camino entre *origen* **y** *destino*

- **Destinos especiales (opcional)**. Dado un lugar *A* cualquiera, denominamos "destino especial" a cualquier lugar *B* del mapa tal que existe algún camino de *A* hacia *B*, pero ninguno de *B* hacia *A*. El formato de entrada será:
  DestinosEspeciales(*origen***)**
  Este comando calculará todos los destinos especiales de *origen*. La salida será:
  Destinos especiales: *destino1*, *destino2*...
  Total: *e* **lugares**

D. Formato acortado

Los comandos de añadir y eliminar lugares y carreteras se podrán escribir también de forma acortada, según la tabla de abajo.

Comando normal	Forma acortada
AñadirLugar(*nombre*,*información asociada***)**	**AL(***nombre***)**
AñadirCarretera(*origen*,*destino*,*coste*,*información***)**	**AC(***origen*,*destino*,*coste***)**
EliminarLugar(*nombre***)**	**EL(***nombre***)**
EliminarCarretera(*origen*,*destino***)**	**EC(***origen*,*destino***)**

Cuando se use la forma acortada, no se deberá producir ninguna salida. Obsérvese que en la forma acortada la *información asociada* es inexistente (es una cadena vacía).

E. Comandos adicionales

Son adicionales (opcionales) los tres siguientes comandos o grupos de comandos:

AD1. Eliminar lugar y eliminar carretera (ambos comandos van juntos).

AD2. Listar lugares (implementado de manera eficiente).

AD3. Cálculo de los destinos especiales de un lugar.

F. Fases de desarrollo

Para una correcta resolución de la práctica, los alumnos deberán cumplir las siguientes fases en los plazos señalados abajo. El cumplimiento de estas fases se validará en el juez on-line de la asignatura, superando los problemas indicados entre paréntesis. Además, se entregará una breve memoria del trabajo realizado en cada fase. Las fases de desarrollo son:

F1. Hasta el 20 de noviembre: resolver los problemas básicos referentes a procesar el texto de entrada (020 y 021), definir el tipo lugar (022), las listas de lugares (023, 024), el tipo carretera (025), y el mapa de carreteras con listas (026).

F2. Hasta el 14 de diciembre: implementar un tipo conjunto/diccionario con tablas de dispersión (abierta o cerrada) para almacenar cadenas (220) o lugares (221), eliminar lugares (222), listar lugares (223), y el mapa de carreteras con dispersión (224).

F3. Hasta el 18 de enero: implementar un tipo conjunto/diccionario mediante árboles para almacenar cadenas (321) o carreteras (322), eliminar carreteras (323), el mapa de carreteras con árboles (324), y el planificador de rutas (325). Esta fase coincide con la entrega final de la práctica.

En todos los casos, el juez on-line permanecerá abierto hasta las 14:30 del día señalado. Y la entrega de la memoria será a lo largo del mismo día, a los profesores de prácticas.

Los grupos que cumplan todos los plazos señalados y de forma satisfactoria, tendrán hasta un +1 en la nota final de la práctica. Los grupos que incumplan uno de los plazos (incluido la **F3**), tendrán que hacer un comando más de los adicionales descritos en el apartado **E** (a elegir); los que incumplan dos plazos harán dos adicionales más; y los que incumplan tres plazos harán tres adicionales más. Los alumnos que entreguen la práctica en junio o en la convocatoria de septiembre, deberán hacer todos los comandos adicionales.

G. Documentación

La documentación a entregar durante las fases intermedias de desarrollo (**F1** y **F2**) contendrá: el listado del código, una descripción somera de los aspectos más relevantes del trabajo realizado, una indicación de los envíos realizados al juez on-line (para cada envío incorrecto, indicar brevemente dónde estaba el fallo), y el número total de horas de trabajo estimado.

La documentación final (fase **F3**) contendrá los siguientes apartados:

1. **Portada**. Nombre de los alumnos y e-mail de cada uno.
2. **Análisis del problema**. Definir los tipos abstractos que aparecen en el problema, y en qué partes aparecen. Analizar, discutir y justificar las diferentes alternativas que se presentan para la implementación de esos tipos según las operaciones.
3. **Diseño de la aplicación**. Mostrar un esquema gráfico global de la estructura de tipos de datos existentes. Detallar la descomposición modular del programa, qué módulos existen, cuál es la responsabilidad de cada uno y la relación de uso. Documentar cualquier otra decisión de diseño que pueda resultar de interés.
4. **Listado del código**. Incluyendo el fichero `makefile` necesario para compilar.
5. **Informe de desarrollo**. Describir cómo ha sido la coordinación y el reparto del trabajo entre los miembros del grupo. Rellenar las tablas de dedicación personal en las dis-

tintas fases del trabajo. Se utilizarán tablas como las explicadas en las páginas 37 y 350 del texto guía, rellenadas con el mayor rigor posible.

6. **Conclusiones y valoraciones personales**.

H. Evaluación de la práctica

H.1. Obligatorio

Para aprobar la práctica se requiere que:

- El programa se pueda ***compilar sin errores*** en las máquinas del laboratorio de prácticas, en la fecha y hora en la que se realice la entrevista final con los alumnos. En particular, el programa estará escrito en C++, y el código se deberá compilar en Linux.
- El programa debe ***funcionar correctamente***, sin colgarse y produciendo ***resultados correctos*** para el conjunto de pruebas que se determinen. Para ello, el profesor puede usar (pero no está limitado a) los casos de prueba incluidos en el juez on-line de la asignatura.
- La ***memoria de la práctica*** debe contener todos los puntos indicados en el apartado **G**, y debe ser entregada en el plazo que se establezca. ¡La documentación entregada no debe contener *faltas de ortografía* (incluida la omisión de tildes)!

H.2. Criterios de valoración

La práctica se puntuará de acuerdo con los siguientes 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. Se deben encontrar los tipos abstractos e implementarlos usando clases, eligiendo las estructuras más adecuadas.
- **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. Usar *makefiles* y compilación separada.
- **Uso del lenguaje**. El código deber ser claro, legible, robusto y eficiente. No crear procedimientos muy largos y complejos. Se valorará el uso de clases genéricas (plantillas) y precondiciones / postcondiciones (asertos).
- **Seguimiento continuo**. El correcto cumplimiento de las fases de desarrollo, marcadas en el apartado F, será un aspecto a favor en la evaluación de la práctica. Se desaconseja la posibilidad de no seguir los plazos a cambio de realizar comandos adicionales.

H.3. Otras cuestiones

La práctica se deberá realizar preferiblemente en **grupos de dos alumnos**. De forma extraordinaria se permiten **grupos de 1 alumno**, pero no se prevé ninguna reducción del trabajo para los mismos.

Para realizar pruebas y para la verificación de las fases de desarrollo, los profesores dejarán en la página web del juez on-line (http://dis.um.es/~mooshak), dentro del concurso "AED 09/10: Practica 1", los problemas mencionados en el apartado **F**. Cada alumno dispondrá de un login y password para acceder a este sistema; el grupo deberá elegir y utilizar una de las cuentas para hacer los envíos al juez.

La fecha de entrega definitiva de esta práctica coincide con la entrega de la última fase, es decir, el 18 de enero de 2010. La forma de hacer las entregas (en papel, por email, a través de SUMA, etc.) la indicará cada profesor de prácticas.

AVISO IMPORTANTE

Las prácticas de todos los grupos, en todas las convocatorias y titulaciones, serán sometidas a un sistema computerizado de **detección de plagios**. Copiar la práctica de otro grupo (en todo o en parte) supondrá el suspenso fulminante de la asignatura en la convocatoria correspondiente, para todos los grupos implicados.