PRÁCTICA Nº 7: 1 sesión

(del 3 al 7 de junio de 2002)

Grafos: Grafo dirigido ponderado y algoritmo de Dijkstra

La vuelta al mundo en 80 días

—¡Veinte mil libras! —Exclamó John Sullivan—. ¡Veinte mil libras, que cualquier tardanza imprevista os puede hacer perder!
—No existe lo imprevisto —respondió sencillamente Phileas Fogg.
—¡Pero, mister Fogg, ese transcurso de ochenta días sólo está calculado como mínimo!
—Un mínimo bien empleado basta para todo.
—¡Pero a fin de aprovecharlo, es necesario saltar matemáticamente de los ferrocarriles a los vapores y de los vapores a los ferrocarriles!
—Saltaré matemáticamente.

0. OBJETIVO

El objetivo de esta práctica es la implementación del TAD **grafo** para que sea utilizada por un programa ya dado de inicio. Este programa implementa el algoritmo de Dijkstra para el cálculo del camino de menor coste temporal de un origen a un destino determinados.

El TAD **grafo** (definición de la clase e programación de métodos) deberá implementarse de las dos formas distintas: mediante matriz de adyacencia y mediante listas de adyacencia.

La práctica se ambienta en el comienzo de la novela de Julio Verne, donde el personaje P. Fogg debe planificar *matemáticamente* su viaje (de ciudad en ciudad, con uno u otro medio de transporte) para dar la vuelta al mundo con un coste de 80 días.

1. ESTRUCTURA DE DATOS: TAD GRAFO DIRIGIDO PONDERADO

Una Red de Comunicaciones puede representarse mediante un grafo **G = (V, A)**,

- donde V es un conjunto de vértices, en el que cada uno simboliza una ciudad,
- y donde A es el conjunto de las aristas comunican una ciudad con otra mediante un medio de comunicación.

En nuestro caso, el grafo habrá de poseer las siguientes características:

- Conexo: para que cualquier ciudad pueda ser accesible de alguna manera.
- Dirigido: para que, de algún modo, podamos asegurar que el recorrido mínimo obtenido avance en husos horarios (vuelta al mundo): desde su origen (Londres) hacia el oriente, hasta su meta (Londres).
- Ponderado (o etiquetado): para almacenar la información sobre la trayectoria entre dos ciudades: tanto su coste temporal en días, como el medio de comunicación empleado (vapor o ferrocarril).

El objeto Red de Comunicaciones será del tipo TAD **grafo**, el cual recogerá información tanto de los vértices (ciudades) como de las aristas (medios de transporte). Ambas implementaciones, la de matriz de adyacencia y la de listas de adyacencia, habrán de ajustarse al modo en el que se invocan los métodos en el programa principal.

A modo de orientación, ambas formas del TAD implementan los conjuntos V y A de diferente manera:

1) Por una parte, la de la MATRIZ DE ADYACENCIA define:

- Un **vector V** de tamaño MAXVER, donde cada elemento contendrá información sobre la ciudad (<nombre de ciudad>). El índice de este vector identificará cada ciudad o lugar (0 Londres, 1 París, 2 Amsterdam, etc.).
- Una **matriz de adyacencia etiquetada A** de tamaño MAXVER×MAXVER para almacenar la información de los trayectos. Cada elemento de la matriz contendrá dos informaciones: <coste temporal> y <tipo de transporte>. Para una matriz A, la arista entre el vértice *i* y el vértice *j* será A[*i*, *j*]. Si no existe una arista entre *i* y *j* el campo de coste temporal tendrá un valor de infinito ∞ (dado que será tipo entero, será suficiente con asignar un número suficientemente grande, el cual puede ser identificado por el valor constante INFINITO).
- Una **variable** que almacenará el número efectivo de ciudades consideradas (≤MAXVER).

2) Por otra parte, la de la LISTA DE ADYACENCIA define:

- Un **vector V** de tamaño MAXVER, donde cada elemento contendrá información sobre la ciudad (<nombre de ciudad>) y la cabeza de una lista ordenada de los vértices adyacentes al elemento. El índice de este vector identificará cada ciudad o lugar (0 Londres, 1 París, 2 Amsterdam, etc.).
- Como se ha indicado, existe una **lista de adyacencia A** para cada uno de los vértices V. Cada nodo de esta lista contendrá dos informaciones: <coste temporal> y <tipo de transporte>. Puede emplearse la implementación realizada en la PRÁCTICA 5, previa modificación para que cada nodo recoja las informaciones indicadas del trayecto.
- Una **variable** que almacenará el número efectivo de ciudades consideradas (≤MAXVER).

2. ALGORITMO DE DIJKSTRA: CAMINO MÍNIMO CON UN SOLO ORIGEN.

A modo de información y como ayuda para la comprensión de lo que realiza el programa principal, se explicará brevemente cómo y qué acciones realiza el algoritmo de Dijkstra.

El algoritmo de Dijkstra proporciona la solución del problema de los caminos más cortos entre un único vértice origen y el resto de vértices destino: tanto el cálculo del costo mínimo, como cada una de sus trayectorias. Como hemos advertido, en nuestro caso el coste de cada trayecto entre vértices es una cuestión de tiempo. También en nuestro caso, sólo un vértice destino será de nuestro interés.

Para realizar el cálculo, el algoritmo de Dijkstra necesita una serie de elementos auxiliares:

- Un conjunto *S* de los vértices cuya distancia más corta desde el origen ya es conocida en una iteración dada del algoritmo. En principio, *S* contiene sólo el vértice de origen. En cada paso, se agrega algún vértice restante *v* a *S*, cuya distancia desde el origen es la más corta posible. Suponiendo que todos los arcos tienen coste no negativo, siempre es posible encontrar un camino más corto entre el origen y *v* que pasa sólo a través de los vértices de *S*, y que se llama *especial*.
- Un vector *D* de vértices para registrar, en cada paso del algoritmo, la longitud del camino especial más corto a cada vértice. Cuando *S* incluya todos los vértices, habríamos llegado al final del algoritmo; en ese momento, todos los caminos entre el origen y el resto de vértices son especiales, y *D* contiene el menor coste del camino entre el origen y cada vértice.
- Un vector *P* de vértices para poder reconstruir el camino más corto del origen a cada vértice. Al término de Dijkstra, *P[v]* contendrá el vértice inmediato anterior a *v* en el camino más corto, y dicho camino podrá reproducirse regresando por los vértices predecesores del vector *P*.

Con estos elementos, el algoritmo de Dijkstra puede formularse del siguiente modo:

```
S ← {0};
Desde i ← 1 hasta (num_ciudades - 1)
       D[i] ← coste (0, i);
       P[i] ← 0;
Desde i ← 1 hasta (num_ciudades - 2)
       Elige un vértice w en V-S tal que D[w] sea un mínimo;
       Agrega w a S;
       Para cada vértice en V-S hacer
              D[v] ← min (D[v], D[w] + C[w, v]);
              Si el mínimo era el segundo término (D[w] + C[w, v]) entonces
                     P[v] ← w;
```

Para definir el conjunto S, se ha utilizado la implementación de *conjunto* de la PRÁCTICA 2. Esta implementación deberá añadirse tal cual al compilar el proyecto.

3. TAREAS A REALIZAR: CALCULO DEL ITINERARIO VUELTA AL MUNDO

1. Localizar en el programa principal los métodos utilizados.
2. Implementación del TAD grafo mediante matriz de adyacencia.
3. Implementación del TAD grafo mediante listas de adyacencia.
4. Tratamiento de errores: Los métodos deberán valorar posibilidades de error (inexistencia de un vértice o arista cuando se pide información sobre él, sobrepasar los límites de la implementación, etc.) y devolverlas en valores booleanos. Podrá manipularse el programa principal sólo con el fin del tratamiento de estos errores (mostrar por pantalla el error, etc.).
5. Se modificará la función Presentación() y se permitirá mejorar la interfaz con el usuario.

Como mejora de la práctica podrá realizarse la siguiente tarea:

- Mostrar la traza de la elección de los nodos en la selección del mejor camino. Podrá mostrarse la evolución de los vectores D (*days*) y P (*path*), y explicar —sólo en los dos primeros casos— el porqué del cálculo de los costes de los caminos especiales. Esta actividad requiere una buena compresión del funcionamiento del algoritmo de Dijkstra, la cual podrá ser ampliada en el libro AHO, HOPCROFT, ULLMAN, *Estructuras de datos y algoritmos*, pp. 205ss. Este apartado se entregaría en un documento aparte.

El programa principal sólo podrá ser modificado en alguno de los sentidos indicados anteriormente, y se indicará de forma clara mediante comentarios las partes del programa que han sido añadidas.

Se supondrá una carga de datos como la de los ficheros adjuntos *ciudades.dat* y *trayectos.dat*, donde el orden de índices de vértices (códigos de ciudades) es ascendente comenzando desde el vértice 0 (Londres - Salida) en incrementos de uno hasta el vértice 53 (Londres - Llegada).

4. REQUISITOS

Recordar que para poder acceder a la sesión de prácticas correspondiente es necesario entregar un pequeño esquema (documentación del programa) de lo que se va a realizar en la práctica. Si no se presenta la documentación ANTES de acceder al laboratorio, no se permitirá la entrada a la sesión de prácticas.

5. ENTREGA DE LA PRÁCTICA

La entrega de esta séptima práctica debe incluir los siguientes archivos:

1.- **`viaje##.cpp:`** Un programa principal modificado.

2.- **`GrafM##.cpp`**, **`GrafM##.h`**: Implementación Matriz Adyacencia del TAD grafo

3.- **`GrafL##.cpp`**, **`GrafL##.h`**: Implementación Lista Adyacencia del TAD grafo

La entrega puede realizarse por e-mail a la dirección del profesor encargado del grupo o en disquete personalmente, siempre ANTES de transcurrida una semana desde la realización de esta sesión de prácticas.

Cualquier práctica entregada fuera de plazo no será admitida para su corrección.

Fecha límite de entrega de prácticas: 14 de Junio de 2002