

Estructuras de Datos no Lineales

Práctica 6

Problemas de grafos I

TRABAJO PREVIO

Antes de asistir a la sesión de prácticas es obligatorio:

1. Imprimir copia de este enunciado.
2. Lectura profunda del mismo.
3. Reflexión sobre el contenido de la práctica y generación de la lista de dudas asociada a dicha práctica y a los problemas que la componen.
4. **Esbozo serio de solución** de los problemas en papel (al menos de los que se hayan entendido).

PASOS A SEGUIR

1. Para cada uno de los problemas escribir un módulo (de nombre por ejemplo `ejercicioN.cpp`) que contenga las funciones requeridas en el enunciado, para lo cual se hará uso de las clases y algoritmos de grafos proporcionados.
2. Escribir un programa de prueba de la solución propuesta para el problema, donde se realicen las llamadas a las funciones correspondientes definidas en el paso anterior, comprobando el resultado de salida para una batería suficientemente amplia de casos de prueba. Esto se puede hacer de dos maneras: Incluyendo la función `main()` en el fichero `ejercicioN.cpp` del paso anterior; o bien, creando un nuevo fichero `.cpp` para la función `main()`, que se compilará por separado y se enlazará con este `ejercicioN.cpp` anterior.

MATERIAL PARA LAS PRÁCTICAS DE GRAFOS (6 a 8)

Para la realización de ésta y siguientes prácticas el estudiante dispone, junto a este enunciado, de diversos ficheros de código C++ con las implementaciones de las distintas estructuras de datos para la representación de grafos, así como de los algoritmos de grafos estudiados en las clases teóricas. Además se han incluido algunas funciones de utilidad para la entrada y salida de grafos y para facilitar la presentación de los resultados de los ejercicios. Todo este material se distribuye entre varias cabeceras, que el alumno deberá incluir en los programas donde las use con los correspondientes `#include`, y ficheros `.cpp`, que habrá que enlazar con dichos programas.

1. **grafoMA. [h|cpp]**
Clase `Grafo`. Grafo no ponderado mediante matriz de adyacencia.
2. **grafoLA. [h|cpp]**
Clase `Grafo`. Grafo no ponderado mediante listas de adyacencia.

3. **grafoPMC.h**

Clase genérica `GrafoP<T>`. Grafo ponderado con costes de tipo `T` representado mediante matriz de costes.

4. **grafoPLA.h**

Clase genérica `GrafoP<T>`. Grafo ponderado con costes de tipo `T` representado mediante listas de adyacencia.

Estas cuatro clases tienen constructores para extraer grafos desde ficheros de texto y también tienen sobrecargado el operador de inserción (`<<`) para mostrar los grafos en flujos de salida. Para más detalles sobre otros métodos y sobre el formato de los ficheros de entrada consultar los comentarios insertados en los ficheros de cabecera.

5. **alg_grafoMA.[h|cpp]**

Algoritmos para grafos no ponderados representados mediante matriz de adyacencia:

- Recorridos en profundidad y anchura.
- Algoritmo de Warshall.

6. **alg_grafoPMC.h**

Algoritmos para grafos ponderados representados mediante matriz de costes:

- Algoritmos de Dijkstra y Floyd.
- Funciones para recuperar los caminos hallados mediante los algoritmos de Dijkstra y Floyd.
- Algoritmos de Prim y Kruskal.

7. **alg_grafo_E-S.[h|cpp]**

Operadores de inserción en flujo de salida para visualizar los resultados obtenidos con los anteriores algoritmos para grafos:

- Sobrecarga del operador de inserción (`<<`) para vectores genéricos, el cual se puede emplear para presentar los resultados del algoritmo de Dijkstra (vectores de costes y vértices).
- Sobrecarga del operador de inserción (`<<`) para matrices genéricas, que se puede usar para visualizar las matrices de costes y vértices que se obtienen con el algoritmo de Floyd.
- Especialización del operador de inserción (`<<`) anterior para matrices de valores lógicos (`bool`). Sirve para imprimir el resultado del algoritmo de Warshall.
- Sobrecarga del operador de inserción (`<<`) para listas de vértices de un grafo, que se puede utilizar para mostrar recorridos de grafos y los caminos de coste mínimo hallados con los algoritmos de Dijkstra y Floyd.

Si se desea un formato de salida diferente, se puede prescindir de estos dos ficheros y definir a conveniencia otras sobrecargas de estos operadores.

8. **matriz.h**

Clase genérica `matriz<T>`. Matriz cuadrada de valores de tipo `T`. Sencilla clase auxiliar para obtener los resultados de los algoritmos de Floyd y Warshall.

La cabecera `matriz.h` ya se incluye en los ficheros de cabecera de los algoritmos de grafos (`alg_grafoMA.h` y `alg_grafoPMC.h`), por lo que no es necesario incluirla en los programas.

9. `pilaenla.h`, `colaenla.h`, `listaenla.h`, `particion.[h|cpp]`, `apo.h`
Clases auxiliares requeridas para la implementación de grafos y de los diversos algoritmos. Estas cabeceras normalmente no habrá que incluirlas en los programas, pues ya están incluidas en los ficheros donde hacen falta, aunque será necesario enlazar `particion.cpp` cuando se use el algoritmo de Kruskall.

PROBLEMAS

`d[w]` coste de cualquier nodo hasta el destino

1. Añadir una función genérica, llamada `DijkstraInv`, en el fichero `alg_grafoPMC.h` para resolver el problema inverso al de Dijkstra, con los mismos tipos de parámetros y de resultado que la función ya incluida para éste. La nueva función, por tanto, debe hallar el camino de coste mínimo hasta un destino desde cada vértice del grafo y su correspondiente coste. D es costes minimo de cualquier nodo hasta destino
operador de asignacion memoria

2. Definiremos el *pseudocentro* de un grafo conexo como el nodo del mismo que minimiza la suma de las **distancias mínimas** a sus dos nodos más alejados. Definiremos el *diámetro* del grafo como la suma de las distancias mínimas a los dos nodos más alejados del pseudocentro del grafo.

Dado un grafo conexo representado mediante matriz de costes, implementa un subprograma que devuelva la longitud de su diámetro.

3. Tu empresa de transportes “PEROTRAVEZUNGRAFO S.A.” acaba de recibir la lista de posibles subvenciones del Ministerio de Fomento en la que una de las más jugosas se concede a las empresas cuyo grafo asociado a su matriz de costes sea acíclico. ¿Puedes pedir esta subvención?

Implementa un subprograma que a partir de la matriz de costes nos indique si tu empresa tiene derecho a dicha subvención.

muy importante

4. Se necesita hacer un estudio de las **distancias mínimas** necesarias para viajar entre **dos ciudades cualesquiera** de un país llamado Zuelandia. El problema es sencillo pero hay que tener en cuenta unos pequeños detalles:

- La orografía de Zuelandia es un poco especial, las carreteras son muy estrechas y por tanto **solo permiten un sentido de la circulación**.
- Actualmente Zuelandia es un país en guerra. Y de hecho hay una serie de ciudades del país que han sido tomadas por los rebeldes, **por** lo que no pueden ser usadas para viajar.
- Los rebeldes no sólo se han apoderado de ciertas ciudades del país, sino que también han cortado ciertas carreteras, (por lo que estas carreteras no pueden ser usadas).
- Pero el gobierno no puede permanecer impasible ante la situación y ha exigido que absolutamente todos los viajes que se hagan por el país pasen por la capital del mismo, donde se harán los controles de seguridad pertinentes.

Dadas estas cuatro condiciones, se pide implementar un subprograma que dados

- el grafo (matriz de costes) de Zuelandia en situación **normal**,
- la relación de las ciudades tomadas por los rebeldes,
- la relación de las carreteras cortadas por los rebeldes
- y la capital de Zuelandia,

true sea infinito, puede estar cortado de A a B y de B a A no por ser dirigido
ciudades tomadas o no, vector de booleanos

no poder entrar a una ciudad corta la columna y no poder salir la fila
aplicar floyd

si todos los viajes tienen q pasar por la capital, se puede hacer en orden cuadratico. obliga a hacer los viajes en dos trozos,
origen-capital, capital-destino

calcule la matriz de costes mínimos para viajar entre cualesquiera dos ciudades zuelandesas en esta situación.

5. Escribir una función genérica que implemente el algoritmo de Dijkstra usando un grafo ponderado representado mediante listas de adyacencia.