Análisis y Diseño de Algoritmos

Actividad sobre conceptos de Grafos

Autor/es

Pedro Miguel Carmona Broncano

Rubén Marín Lucas

Algoritmos implementados:

Los algoritmos principales que resuelven los dos problemas que se nos plantean son los siguientes:

1. En primer lugar a partir de un conjunto de ciudades (agrupadas en parejas) tenemos que hallar el camino mínimo entre ellas (entre las parejas). Con este fin hemos aplicado el algoritmo de Floyd el cuál nos permite hallar el camino mínimo de cualquier vértice a todos los demás del grafo a tratar:

   void caminoMinimo(mFloat &C, mString &P);

   - Hemos decidido que este era el algoritmo correcto puesto que el otro posible, que sería el algoritmo de Dijkstra solo nos haya el camino mínimo de un vértice a todos los demás, pero como tenemos que hacerlo un número repetido de veces (todas las que nos digan las últimas líneas del documento de texto de entrada), pues al final consideramos más eficiente realizar el algoritmo de Floyd una vez y luego recuperar los caminos.

2. En segundo lugar se pide realizar un problema de optimización en el que tenemos que unir todas las ciudades a partir del menor número de carreteras (y las más cortas posibles), en otras palabras se nos está pidiendo un árbol de expansión de coste mínimo. Por ello, hemos elegido realizar el algoritmo de Prim:

   void arreglarCarreteras(Carretera &gs);

   - En este punto parece más lógico que el grafo sea conexo, ya que en caso contrario no se podría realizar el algoritmo.

➔ También cabe destacar el algoritmo usado para recuperar los caminos intermedios de Floyd:

void Camino(Carretera c, int i, int j, mString &P, ofstream &flujoSalida);

Descripción de la solución

Para comenzar, la problemática consistiría en encontrar la estructura de datos correcta para la representación del sistema de carreteras que se nos pide. Para ello, hemos elegido a un grafo como la estructura adecuada. Los vértices del grafo representarían a las diferentes ciudades y las aristas indicarían la existencia de carretera entre dos ciudades (dos vértices). Además tendría sentido que el grafo fuera:

- No dirigido: Puesto que puedes ir de una ciudad c1 a una ciudad c2, o de la ciudad c2 a la ciudad c1.
- Etiquetado: Debido a que de esta manera puede representarse la distancia entre las ciudades.
- Conexo: Consideramos que todas las ciudades es lógico estuvieran unidas en la red de carreteras. No tendría mucho sentido que desde una ciudad no se pudiera llegar a otra (o incluso a ninguna).

Una vez aclarada la estructura llega la hora de decidir cómo implementarla: Para ello hemos decidido crear una clase (class Carretera) que recoja como atributos un vector de string (vString cVertices) en la que se almacene el nombre de cada ciudad, es decir, la información de cada vértice; y una matriz de float (mFloat mAdyacencia) que representa la matriz de adyacencia de nuestro grafo, ésta almacena la distancia entre las distintas ciudades, en caso de que no exista camino entre dos ciudades se representa con un -1 en la matriz, y la diagonal quedará completa con 0.

Para seguir, el siguiente problema era leer los datos del fichero de texto (datos.in) para después realizar unas operaciones y volcar los resultados en otro fichero de texto (datos.out). De esta manera se ha planteado realizar distintos módulos en main.cpp:

- Un módulo que sirve para leer los datos del fichero de entrada y que carga dichos datos en nuestro grafo (void cargarDatos(Carretera &c, IList<nodo*> *&caminos)).

  - Sirve para leer los datos de este fichero en concreto, porque respeta los espacios en blanco, los saltos de líneas y los valores que nos dicen.
  - Carga los datos en nuestro grafo a partir de los métodos necesarios que contiene nuestra clase.
  - Cómo en las últimas líneas se lee información que será tratada, y sobre la que se realizarán operaciones, consideramos oportuno almacenarla en una estructura de datos auxiliar, con el objetivo de leer el fichero de una sola vez. Con este fin se ha usado un template de una lista con punto de interés en la que se guardarán punteros a elementos de tipo nodo, una estructura auxiliar definida con el objetivo de guardar las ciudades sobre las que se realizatán operaciones (estas operaciones dependen de dos ciudades, caminos entre dos ciudades). Así a la hora de realizar las operacines sólo tendremos que recorrer la lista.(*)

- Módulo que recupera los caminos intermedios hallados en el algoritmo de Floyd de caminos mínimos. Dicho módulo ya escribe en el fichero de texto de salida (void Camino(Carretera c, int i, int j, mString &P, ofstream &flujoSalida))

- Dos algoritmos, uno por cada función que se nos pide, los dos recorren la lista con la información a tratar y llaman a los métodos propios de la clase Carreteras que resuelve los problemas que se nos plantean. Estos algoritmos también se sirven del módulo anterior y escriben la información que se nos pide en el fichero de texto de salida. (void Algoritmo2(Carretera c, IList<nodo*> *caminos, ofstream &flujoSalida))
  void Algoritmo1(Carretera c, IList<nodo*> *caminos, mString &P, mFloat &C, ofstream &flujoSalida)

- (*)La decisión de usar un puntero a una lista con punto de interés, en la que contiene punteros a nodos (ya se ha mencionado que es una estructura que guarda información de dos ciudades) es elegida por la ventaja que nos permite la implementación de estructuras con punteros: Nos permite una implementación "más flexible" en el sentido de que no es necesario reservar un espacio de memoria fijo, como si es necesario en otros tipos de estructuras (cómo pueden ser los vectores, que dependen de una constante), y aunque nos añaden una problemática, como es una mayor lentitud a la hora de acceso al dato, no se ve demasiado influenciado en este caso puesto que el acceso a esta estructura es bastante limitado (una vez por cada algoritmo, un total de 2 veces). Por estas razones pensamos que la implementación elegida es correcta, y porque tampoco hay que olvidar otra de las grandes ventajas que nos permite esta elección, y que es la posibilidad de poder borrar los datos que contiene, además de la propia estructura, ya que no podemos olvidar que está creada con una función secundaria y con el objetivo de mejorar la claridad y complejidad del problema.

- Para los ficheros de datos hemos seguido el guión de la actividad respetando un espaciado entre palabras de la misma línea, y una línea en blanco para separar los distintos bloque de información.

<u>Análisis de la complejidad</u>

Haremos un estudio de complejidad de los 3 algoritmos anteriormente mencionados:

- void caminoMinimo(mFloat &C, mString &P); //Algoritmo de Floyd

  +*Tam_poblema*: Tamaño de la matriz de adyacencia

  +*Operaciones significativas*: Número de comparaciones con la matriz C (suma < C[i][j])

  La comparación se realiza en cada iteración, por tanto no existe caso peor ni mejor, solo existe un caso único: +*Caso único*: T(n) = $n^3 \in O(n^3)$

- void arreglarCarreteras(Carretera &gs); //Algoritmo de Prim

  +*Tam_poblema*: Tamaño de la matriz de adyacencia

+*Operaciones significativas*: Número de comparaciones con menor (menor > mAdyacencia[i][j])

La comparación se realiza en cada iteración, por tanto no existe caso peor ni mejor, solo existe un caso único: +*Caso único*: T(n) = $n^3 \in O(n^3)$

➔ void Camino(Carretera c, int i, int j, mString &P, ofstream &flujoSalida);

+*Tam_poblema*: Tamaño de la matriz de adyacencia

+*Ecuación de recurrencia*: $T(n) = \begin{cases} 1 & n \leq 1 \\ 2T(n-1)+1 & x>1 \end{cases}$

Conclusiones

Consideramos que ha sido una actividad útil porque nos ha hecho estudiar y repasar algunos conceptos de grafos, además de algoritmos relacionados con el tratamiento de la información de grafos. También nos ha servido para repasar complejidad, y para enlazar algunos conceptos relacionados con los tipos de algoritmos que estamos viendo ahora en teoría.

En definitiva, lo hemos considerado como una buena práctica.