Universidad de Concepción
Facultad de Ingeniería
Departamento de Informática y Ciencias de la Computación

Tarea Computacional

Integrantes: Diego Domínguez Rivera.
Javier Arriagada Inostroza.
Isaias Huerta Vargas.

Asignatura: Matemáticas Discretas.

Docente: Lilian Salinas Ayala.

Concepción, Noviembre de 2015.

1.- Índice

2.- Introducción

Nosotros las personas, como usuarios, generalmente ignoramos todo lo que está detrás de los servicios que utilizamos día a día, como el sistema de transporte público, redes telefónicas, de internet, distribución de alimentos, repartos, etc. La verdad es que detrás de estos servicios hay un proceso de desarrollo complejo para que estos funcionen, idealmente, lo más eficiente posible. Así, una etapa importante en el proceso de desarrollo consiste en el modelamiento del problema. Aquí viene al caso el uso de grafos para modelarlos, lo que nos ofrece una visión más gráfica de lo que queremos tratar.

En esta ocasión, el problema propuesto consiste en: dado un mapa, en base a ciudades y rutas, debemos encontrar el mejor lugar para construir una planta de distribución para así repartir un producto a todas las ciudades gastando lo mínimo.

La solución será entregada por un programa escrito en lenguaje C que recibe un archivo de texto con la información de ciudades (nombre, ubicación) y rutas (nombre, punto de partida, punto de llegada, largo) que luego, utilizando el algoritmo de Floyd-Warshall, determinará el camino más corto según las condiciones del grafo que representa el problema.

Este documento está organizado en secciones que explican más detalladamente cómo se abordó el problema.

Las secciones son las siguientes:

- Objetivos
- Modelo
- Implementación
- Conclusiones

3.- Objetivos

Entre los objetivos de este trabajo, podemos destacar:

- Familiarizarse con problemas del mundo real / laboral.
- Utilizar el computador como herramienta eficaz para resolver problemas.
- Aplicar los conocimientos del lenguaje C para poder desarrollar un programa computacional en el que se desarrollen cálculos y acciones para así entregar una solución óptima al problema.
- Entender, manejar y almacenar una base de datos mediante un archivo de texto.
- Analizar y comprender algoritmos específicos de análisis sobre grafos para encontrar el camino mínimo en ellos.
- Hacer uso del conocimiento de grafos, para proponer soluciones a problemas.
- Aprender a redactar informes, para llegar mejor preparados a la tesis.

4.- Modelo

Un pilar fundamental en cualquier problema son los datos que se nos entregan. En este caso, los datos son entregados en un archivo de texto. Este archivo contiene las ciudades con sus coordenadas y luego las rutas.

Las ciudades y las rutas son representadas por structs. El struct “ciudad” contiene el nombre y las coordenadas de la ciudad. El struct “ruta” contiene el nombre de la ruta, lugar de inicio, lugar de término kilómetro donde comienza, kilómetro donde termina, largo y tipo.

Almacenar las ciudades (nodos) es relativamente sencillo pues todas vienen en un mismo formato dentro del archivo (nombre coordenada x coordenada y), por lo que solo se necesitó un arreglo de structs de tipo “ciudad” para almacenar el nombre y las coordenadas de cada ciudad.

El caso de las rutas es más complejo. A diferencia de las ciudades, las rutas se pueden presentar en tres formatos distintos, los cuales son:

- `nombre ciudad ciudad largo` (1).
- `nombre ciudad ruta km_inicio(o termino) largo` (2) (es lo mismo si comienza en una ruta y termina en una ciudad).
- `nombre ruta km_Inicio ruta km_termino largo` (3).

Debido a lo anterior, es necesario hacer distinciones al momento de que el programa realiza la lectura del archivo.

Ahora, de los algoritmos disponibles decidimos utilizar Floyd-Warshall, que busca la distancia mínima entre dos vértices cualesquiera de un grafo. Este algoritmo trabaja con la matriz de adyacencia del grafo.

Ya almacenados los datos, se debe crear la matriz de adyacencia, la cual se llena con los datos de las ciudades y rutas guardados anteriormente. Esta matriz nos informa sobre los vértices adyacentes a un nodo particular mostrando el valor ”inf” si el nodo i no es adyacente al nodo j, “0” si i = j y algún valor positivo (distancia de la ruta) si i es adyacente a j (notar que el costo de ir de i a j es igual al costo de ir de j a i).

Cabe mencionar que además de los nodos que representan ciudades, también debemos representar como nodos las intersecciones de las carreteras para los casos (2) y (3), estos nodos resultantes también estarán presentes en la matriz de adyacencia.

Luego, aplicamos el algoritmo de Floyd-Warshall, que luego de algunas iteraciones, va a determinar la ruta más corta que pase por todos los nodos (esto se explicará detalladamente en la siguiente sección).

5.- Implementación

Se utilizó el algoritmo de **Floyd-Warshall**, que permite calcular la distancia entre dos puntos cualesquiera del grafo. Para ello, se creó una función floyd(n) que recibe como parámetro el número de vértices. Esta función genera una matriz tridimensional `d[][][]` que almacena en dos dimensiones la matriz de adyacencia, mientras que la otra dimensión sirve para guardar las iteraciones de k. La parte principal del algoritmo se encuentra en la siguiente línea:

```
d[k][i][j] = min(d[k-1][i][j], d[k-1][i][k] + d[k-1][k][j])
```

Aquí, por cada iteración de k se va generando una nueva matriz que representa una optimización de las distancias entre vértices. Por ejemplo, para k = 0, se utiliza la primera fila y columna para comparar la celda que se encuentra intersectada. Si la suma de las componentes resulta ser menor al valor de la celda, este valor es reemplazado por el valor de la suma. Además, para cada reemplazo se tiene que modificar también el valor correspondiente en `rec[][]` que indica el camino por el cual se debe pasar para llegar de un vértice a otro.

Con estos datos podemos ahora calcular el **recorrido mínimo** entre cada par de vértices. Para ello, creamos la función recursiva `d_rec()` que recibe como argumentos dos valores, que representan al vértice inicial y final. Aquí analizamos la matriz de recorridos `rec[][]`, en donde `rec[x][y] == y || rec[x][y] == x` quiere decir que son adyacentes, por lo tanto, la distancia es su peso. En otro caso, significa que hay uno o más vértices por donde se debe pasar para unir el vértice de inicio y final, por lo tanto se vuelve a la función hasta que se encuentre el punto en que hayan dos vértices adyacentes. Para extraer el peso se utiliza la matriz `d[n][][]`. Ahora con esta función podemos calcular la **sumatoria de las distancias** para cada vértice. Esto lo hacemos dentro de la función `main()`. Iteramos para calcular la sumatoria del recorrido entre vértice actual y cada uno de los otros vértices. Luego se busca el mínimo y se interpreta para imprimir la ciudad que corresponde.

Para la **conexidad** utilizamos el algoritmo BFS para encontrar las componentes conexas. Las componentes conexas fueron guardadas en el array `visited[]`, y éste último se lo entregamos como parámetro a la función `caminoMasCorto()`. Esta función busca los pares de vértices disconexos óptimos para unir un par de componentes conexas con una carretera, la cual es agregada a `ady[][].`

Cabe mencionar que el programa no es útil para un grafo con intersecciones de carreteras. Los datos de las intersecciones se encuentran guardados, pero no se pudo hacer una correcta implementación con intersecciones de carreteras.

6.- Conclusiones

Finalizando este trabajo, podemos concluir que:

- Si bien los grafos permiten presentar un problema de una manera más fácil, al momento de programarlos se deben considerar las limitaciones de la máquina y el lenguaje que se está utilizando. Particularmente, haciendo uso del Lenguaje C, la representación de grafos es compleja, tal vez los lenguajes de programación orientados a objetos tengan una mayor cantidad de herramientas para hacer este trabajo más “fácil”.

- Consideramos que la tarea es una representación básica de los problemas que pueden ser modelados con grafos, pues generalmente hay muchas variables que pueden afectar en la búsqueda de la solución que, para esta tarea, no fueron consideradas.

- No fue fácil decidir cuál sería el método de almacenamiento de los datos. Finalmente nos quedamos con guardar las ciudades y rutas como structs pues los datos se muestran más organizados

- Si bien el algoritmo de Floyd-Warshall nos dio la solución al problema, existen otros algoritmos que pueden realizar esta tarea más rápido.

- Aparecieron una serie de problemas durante el desarrollo de esta tarea, sin embargo los más importantes fueron cómo determinar los nodos de las intersecciones de las rutas (lo que no se pudo concretar) y la implementación correcta del algoritmo de Floyd-Warshall.