Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-2693 - Laboratorio de Algoritmos y Estructuras III
Trimestre Abril-Julio 2016

Prelaboratorio de la semana 10: Implementación de los algoritmos de Prim, Kruskal y Floyd

1. Introducción

Se quiere que implemente los algoritmos sobre grafos de Prim, Kruskal y Floyd. Para ello se les proporcionará un código base que usted debe completar. Este código contiene, entre otros, los archivos `Prim.java`, `Kruskal.java` y `Floyd.java`, que deben tener la implementación de los algoritmos de Prim, Kruskal y Floyd respectivamente. La implementación de los algoritmos puede estar basada en el pseudocódigo que se encuentra en el libro de Meza y Ortega [1]. El código que se les proporciona contiene implementaciones de grafos no dirigidos y dirigidos con costos (o pesos) en los lados, junto con otras utilidades y un Makefile que compila todos los archivos. Es obligatorio que comprenda todo el código que se le está entregando. Esté código base puede ser utilizado en otras actividades de laboratorio y se le puede solicitar que lo modifique.

2. Actividad a realizar

La primera actividad consiste en implementar los algoritmos Prim y Kruskal para obtener el árbol mínimo cobertor de un grafo no dirigido. La idea es que al ejecutarse el constructor de las clases `Prim` y `Kruskal`, se debe ejecutar los algoritmos de Prim y Kruskal respectivamente, tal que el programa guarda en variables privadas la información del árbol mínimo cobertor obtenido. Ambas clases poseen los siguientes métodos que debe implementar:

`getEdgesMST()`: Retorna el conjunto de lados que forman parte del árbol mínimo cobertor.

`weight()`: Retorna la suma de los costos (o pesos) de los lados que forman parte del árbol mínimo cobertor.

La segunda actividad tiene como objetivo la implementación del algoritmo de Floyd para obtener los caminos de costo mínimo entre cada par de vértices de un grafo dirigido usando el algoritmo de Floyd. Tal como en la actividad anterior, al crear un nuevo objeto el constructor de la clase `Floyd` debe ejecutar el algoritmo de Floyd y guardar los resultados en variables privadas. Los métodos de la clase `Floyd` que debe implementar son los siguientes:

`dist(int s, int t)`: Retorna el costo del camino mínimo entre los vértices. El costo del camino mínimo es la suma de todos los costos (o pesos) de los lados que son parte del camino de costo mínimo entre s y t. En caso de que no exista un camino entre s y t, se debe retornar `Double.MAX_VALUE`.

`path(int s, int t)`: Retorna el camino de costo mínimo entre dos vértices. En caso de no existir un camino, retorna `null`.

La tercera y última actividad es la implementación de dos clientes en los que se demuestren el buen funcionamiento de todos los métodos de las clases implementadas. El primer cliente debe llamarse `TestPrimKruskal.java` que tiene como objetivo probar las clases `Prim` y `Kruskal`. El cliente recibe como argumento un archivo con los datos de un grafo no dirigido con costos en los lados. El formato del archivo que representa un grafo no dirigido es como sigue: en la primera línea indica el número de nodos, la segunda línea presenta el número m de lados, luego siguen m líneas con tres números separados por espacio, en donde el primer y segundo número corresponden a los vértices que componen un lado y el tercer número es el costo (o peso) del lado. Por ejemplo, dado el grafo de la Figura 1, se puede crear un archivo llamado `miniGrafo.txt` con el siguiente contenido:

```
5
6
0 1 1.1
0 2 1.2
0 3 1.5
1 4 4.2
3 4 5.3
2 4 0.3
```

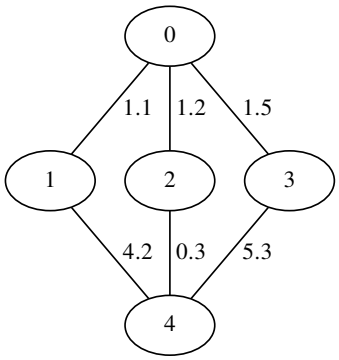

Figura 1: Ejemplo de un grafo no dirigido

Si se quiere probar los algoritmos de Prim y Kruskal con el cliente y el archivo `miniGrafo.txt`, entonces se debe ejecutar:

```
>java TestPrimKruskal miniGrafo.txt
```

El segundo cliente que se debe implementar tiene como finalidad probar mostrar el funcionamiento de los métodos que la clase `Floyd`. El cliente tiene como nombre `TestFloyd.java` y recibe como entrada un archivo que contiene un grafo dirigido. El formato del archivo es análogo al del grafo dirigido con la diferencia que cada línea con los datos de un lado ahora van a corresponder a un arco, donde el primer entero corresponde al vértice de partida y el segundo entero al vértice de llegada. Por ejemplo, dado el grafo dirigido de la Figura 2 y suponiendo que se almacena un archivo llamado `miniDigrafo.txt`, el archivo podría tener el siguiente contenido:

```
5
6
0 1 0.1
0 2 1.2
0 4 3.2
0 3 1.1
3 4 2.3
2 4 3.3
```

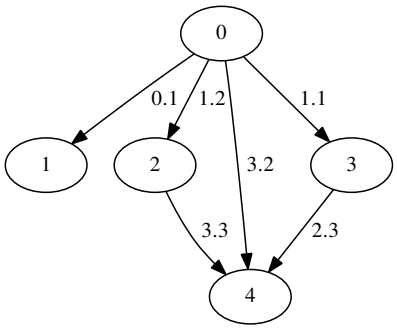

Figura 2: Ejemplo de un grafo dirigido

La ejecución del cliente de `TestFloyd.java` con el digrafo `miniDigrafo.txt`, sería como sigue:

```
>java TestFloyd miniDigrafo.txt
```

Para finalizar se le recuerda que todo el código debe seguir la guía de estilo de `Java`, y debe estar debidamente documentado siguiendo las reglas para la herramienta Javadoc, ver `http://www.oracle.com/technetwork/articles/java/index-137868.html`. Todas las implementaciones deben ser razonablemente eficientes, cumpliendo el orden en tiempo de ejecución esperado para estos algoritmos.

3. Condiciones de entrega

Debe entregar el día 15 de Junio de 2016 antes de las 11:00 am, un archivo comprimido llamado **PreLabSem19-X-Y.tar.gz**, con todos los códigos Java y el archivo Makefile. Las letras **X** y **Y** del archivo comprimido son los número de carné de los integrantes del equipo.

Referencias

[1] Meza, O., and Ortega, M. *Grafos y Algoritmos*, segunda ed. Editorial Equinoccio, 2004.