Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos
2021 - 1

Tarea 3

Fecha de entrega código e informe: Miércoles 7 de julio

Objetivo

- Implementar un bosque de cobertura mínimo modificando lo aprendido con *Minimum Spanning Tree*.

Introducción

La consultoría a Walter White fue todo un éxito. Además de recuperar la fe en sus habilidades de programación, Kevin ganó el dinero suficiente como para cumplir su objetivo final de este semestre: Fundar una empresa distribuidora de papel que acabe con `Dunder Mifflin`, la empresa que lo despidió tras el desastre bioquímico.
Para cumplir su meta, Kevin se asoció con **Dewey**, un niño que se auto-proclama como un genio. En su primera reunión de directorio, Dewey recalcó la importancia de un plan estratégico para reducir los costos de distribución del papel. Para lograr esto, Kevin propone un sistema de optimización que minimice el costo total de traslado del papel entre las distintas sucursales y los clientes.

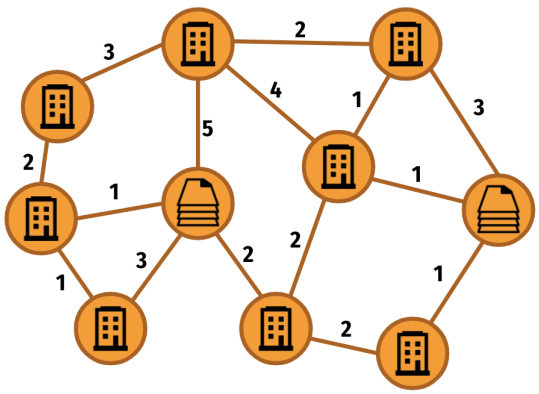
Dewey repartiendo a los trabajadores en las distintas sucursales.

Problema

El problema puede entenderse como una red de nodos que corresponden a dos tipos: clientes y centros de distribución. Además, dos nodos se conectan a través de carreteras, en donde cada una tiene un costo específico. Tu tarea consiste en decidir qué carreteras elegir para que todo cliente se conecte con un centro de distribución y que el costo total sea el mínimo. Se deben precisar los siguientes puntos:

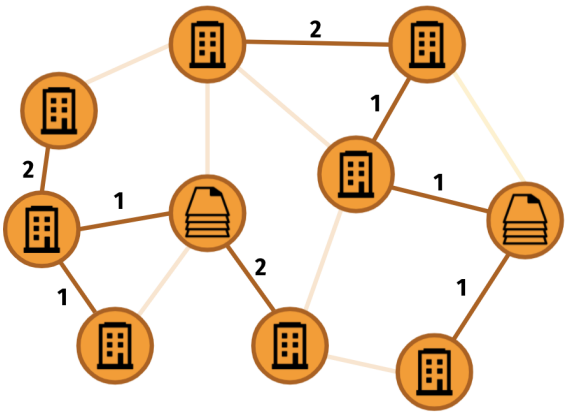
- Un cliente puede conectarse directamente con un centro de distribución (solo una carretera entre ellos) o indirectamente (a través de más de una carretera).
- La red obtenida al elegir las rutas puede ser conexa (un centro abastece a todos los clientes) o no conexa (cada centro abastece a subgrupos independiente de clientes).

Veamos un ejemplo



Red inicial con 2 centros de distribución de papel y 8 clientes.

Tu programa deberá retornar la configuración que minimice el costo total que supone conectar a cada cliente con un centro de distribución.



Red solución con costo mínimo. Bosque de cobertura mínimo formado por dos árboles.

Ejecución

Tu programa se debe compilar con el comando `make` y debe generar un binario de nombre `dewey` que se ejecuta con el siguiente comando:

```
./dewey network output
```

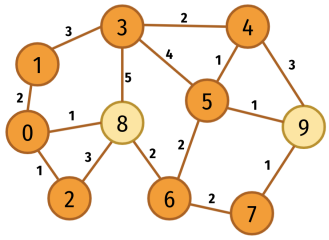
Donde `network` es la ruta a un archivo de texto que incluye la información de la red que se quiere optimizar y `output` el archivo de texto donde se escribe el resultado.

Input

El input está estructurado como sigue:

- Una línea con tres números `C D H` que indican respectivamente la cantidad de clientes (`C`), la cantidad de centros de distribución de papel (`D`), y la cantidad de carreteras (`H`).
- Después vienen `H` líneas que contienen información de cada carretera que conecta a dos nodos. El formato de cada línea es `A B C`, donde `A` y `B` son nodos de la red (ya sean centros de distribución o clientes) y `C` es el costo que implica transitar por dicha carretera.
- Los clientes se identifican con un id i tal que $i \in \{0, ..., \texttt{C - 1}\}$
- Los centros de distribución se representan con un id j tal que $j \in \{\texttt{C}, ..., \texttt{C + D - 1}\}$

Por ejemplo, asignemos una configuración posible a la red que hemos visto anteriormente:



En naranjo, los clientes. En amarillo, los centros de distribución.

Así, el input que recibe el programa podría ser:

```
8 2 15
0 2 1
0 8 1
0 1 2
1 3 3
3 8 5
2 8 3
8 6 2
3 5 4
3 4 2
5 4 1
5 6 2
6 7 2
5 9 1
4 9 3
9 7 1
```

Output

Se espera como output un archivo que contenga en la primera linea el `costo total` y luego las `X` carreteras que fueron seleccionas en el bosque de cobertura mínimo. La numeración de las carreteras se hace respecto a su aparición en el archivo de input (la carretera 0 es la descrita en el ejemplo anterior como la línea `0 2 1`). Para el ejemplo el output esperado es:

```
11
0
1
2
6
8
9
12
14
```

No importa el orden de las carreteras en el output y puede haber más de una solución correcta. En el ejemplo, podemos cambiar `6` por `10` o `11`.

Evaluación

Informe (20 %)

Explica y compara en término de complejidad e implementación[1], dos posibles algoritmos para solucionar el problema (el que tu implementaste y otro) [2].

Código (80 %)

Tu código será evaluado con tests de dificultad creciente. Estos tests están separados en 3 categorías:

- Easy: Para debugear manualmente. No serán evaluados.
- Normal: Usados para testear la correctitud de tu implementación. Serán evaluados.
- Hard: Usados para testear la eficiencia de tu implementación. Serán evaluados.

Para cada test de evaluación, tu programa deberá entregar un output correcto en menos de 10 segundos. Para verificar la correctitud se revisará que tu output cumpla con que (1) cada cliente se conecte directamente o indirectamente a un centro de distribución y que (2) el costo total de la carretera sea el mínimo. Debes tener claro que puede haber más de una configuración óptima para un mismo problema.

Entrega

Código: GIT - Repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59 horas.

Bonus

El siguiente bonus solo aplica si tu nota correspondiente es mayor o igual a 4.

Manejo de memoria perfecto: +5 % a la nota de código

Recibirás este bonus si `valgrind` reporta que tu programa no tiene ni leaks ni errores de memoria en los tests que logres resolver.

[1] De forma teórica y en un máximo de 2 planas
[2] Puedes buscar Kruskal y Prim