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

IIC2133 — Estructuras de Datos y Algoritmos
2020 - 2

Tarea 4

Fecha de entrega código: miércoles 2 de diciembre a las 23:59

Objetivos

- Modelar un problema NP-hard y encontrar una solución subóptima usando algoritmos codiciosos

Introducción: Guido's Catch-22

Luego de recuperarse de su crisis existencial y tras la sorpresa de recibir las eficientes rutinas que le crearon para resolver el juego, Guido vuelve a tener la motivación necesaria para intentar nuevamente una fallida actividad del `pass`ado: Realizar una exitosa auto-neuro-cirugía.

Después de haber modelado correctamente la sinapsis entre sus neuronas Guido tiene la seguridad de que agregando más neuronas (y así acortando las distancias para que realicen sinapsis entre sí), su capacidad de pensamiento lo llevaría a resolver problemas de cualquier clase de complejidad. Sin embargo, cual tragedia griega, cae en la revelación de que el proceso de aumentar su capacidad cognitiva requiere de aún más capacidad cognitiva. Pese a esto, Guido sabe que el posicionamiento de las nuevas neuronas a agregar tiene que ser tal de que pueda aumentar lo más posible su capacidad de pensamiento en un tiempo razonable. Es tu labor asistir al auto-neuro-cirujano en esta labor.

Problema: Steiner Tree

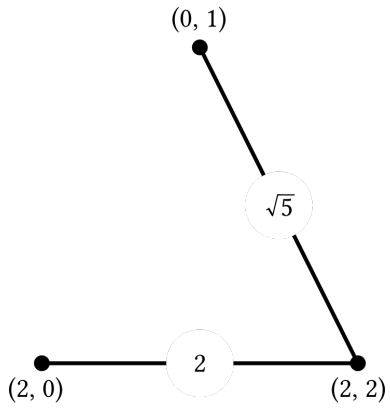
Dado n puntos en un plano $\mathbb{R}^2$, el objetivo es conectar todos los puntos en una sola red tal que el costo total de la red sea mínimo. El costo de conectar dos puntos en el plano es su distancia euclidiana:

$$d(u, v) = \sqrt{(u_x - v_x)^2 + (u_y - v_y)^2}$$

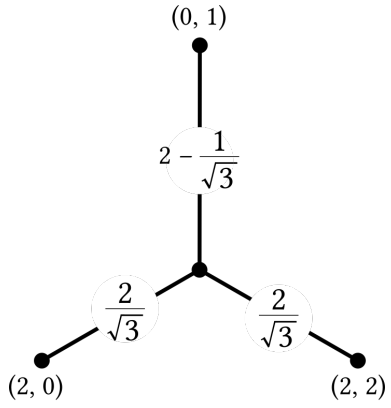
Como se vio en clases, esta red de costo mínimo corresponde al **Minimum Spanning Tree** (MST) del grafo completo en que cada vértice corresponde a uno de los n puntos del plano. El costo de la arista entre cada par de vértices corresponde a la distancia definida anteriormente.

Sin embargo, es posible encontrar una red con un costo aun menor al del **MST** agregando m nuevos puntos al plano, conocidos como **Steiner Points**. Estos nuevos vértices pueden usarse como puntos intermedios para crear una nueva red menos costosa. Este problema se conoce como **Euclidean Steiner Tree Problem**, el cual es **NP-hard** [1].

Por ejemplo, para 3 puntos en el espacio, la red puede ser mejorada agregando un nuevo Steiner point



(a) Red original



(b) Red modificada

Recordemos que el costo total de la red es la suma de todas las aristas. Para este ejemplo el eje Y crece hacia abajo.

El problema entonces consiste en determinar cuantos Steiner points deben ser agregados, y donde agregarlos. Dado que el espacio es en los números reales, la cantidad de opciones posible no está acotada.

[1] El consenso es que no existen algoritmos que puedan resolver problemas NP-Hard en tiempo polinomial, aunque no está demostrado.

Versión rectilínea

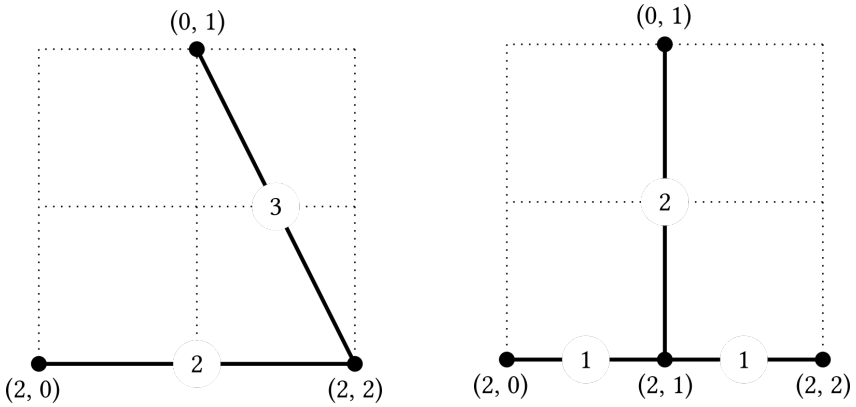
Con el fin de simplificar tu tarea y abstraer las complejidades de implementación, el problema será una simplificación del anterior, conocido como **Rectilinear Steiner Tree Problem**, el cual sigue siendo NP-hard.

La única diferencia es que los puntos se encuentran en el plano $\mathbb{Z}^2$ y la distancia entre dos puntos corresponde a la distancia rectilínea [2]:

$$d(u, v) = |(u_x - v_x)| + |(u_y - v_y)|$$

Este versión del problema es el equivalente a trabajar en una grilla uniforme de puntos, donde tanto los vértices del grafo como los Steiner Points solo pueden estar posicionados en los vértices de la grilla, solo que esta vez la cantidad de opciones disponible está acotada. Los puntos se pueden expresar como (fila,columna).

Veamos, como ejemplo, la red con los mismos vértices que el ejemplo anterior:



(a) Un problema

(b) Una posible solución

Si bien ambas versiones del problema son NP-hard, es posible encontrar soluciones subóptimas en tiempo polinomial utilizando algoritmos codiciosos.

Deberás escribir un programa en **C** que, dada una configuración de vértices en una grilla, sea capaz de reducir el largo total de la red, manteniendo la restricción de pasar por todos los vértices originales. Para esta labor, deberás encontrar puntos Steiner y el árbol de cobertura de costo reducido. Se recomienda investigar la literatura al respecto.

[2] También conocida como distancia Manhattan.

Ejecución

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

```
./steiner input output
```

Donde `input` es la ruta a un archivo con la configuración inicial, y `output` es la ruta al archivo donde tu programa escribirá la solución. En esta ocasión, se les entregará un repositorio que contará con código de `I/O`, `Makefile` y el archivo `main.c`.

Input

El input está estructurado como sigue:

- Una linea con los valores T, N, y D, donde T representa el alto y ancho del tablero, N el número de puntos ya puestos en el tablero y D el costo del MST de los vértices originales. Esta distancia es la que se utilizará como base y es la que tu algoritmo debe mejorar.
- N líneas que describen las posiciones de cada uno de los puntos. Cada una de estas tendrá el formato r_i c_i describiendo respectivamente fila y la columna del punto i

El input para el tablero de ejemplo se vería como:

```
2  3  5
0  1
2  0
2  2
```

Output

El output corresponde a las $n + m - 1$ aristas del MST que cubre los vértices originales y los Steiner points.

Tendrá el siguiente formato:

- Una linea con el valor $n + m$ que indica el número de puntos totales del grafo.
- $n + m - 1$ líneas que representan las aristas entre los vértices. Cada una de las líneas tendrá el formato r_i c_i r_j c_j, donde el punto i de fila r_i y columna c_i está conectado con el punto j de fila r_j y columna c_j.

Es necesario que el output entregado corresponda al MST **de todos los vértices**. El orden de las aristas no importa. Para el tablero de ejemplo, una respuesta válida sería:

```
4
2  0  2  1
2  2  2  1
2  1  0  1
```

Evaluación

Para esta tarea solo habrá nota de código. Como solución a cada test se espera que entregues un árbol de cobertura de los vértices originales junto con los Steiner points. Este será evaluado con una nota del 1 al 8 de la siguiente manera:

- Si el output no corresponde a un árbol de cobertura tendrás un 1 en ese test.
- Si el costo del árbol de cobertura es superior al del MST del grafo original, entonces tendrás un 1 en ese test.
- Si el costo del árbol de cobertura es entre 0 % a 6 % inferior al costo del MST, tendrás una nota linealmente distribuida entre 4 a 8 según el porcentaje de mejora. [3]

Cada test tendrá un máximo de 10 segundos de ejecución; si tu programa tarda más tiempo, será interrumpido y tendrás un 1 en ese test. La nota final de la tarea será el promedio de las notas individuales de cada test.

Entrega

Código: Git – Repositorio asignado. Se revisará el último `commit` en `main` a más tardar el día de entrega a las 23:59, hora de Chile continental.

Bonus

Los bonus solo aplican si la nota base correspondiente es superior o igual a 4.0.

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.

[3] Esto quiere decir que si lo único que haces es calcular el MST del grafo original tendrás un 4.0.