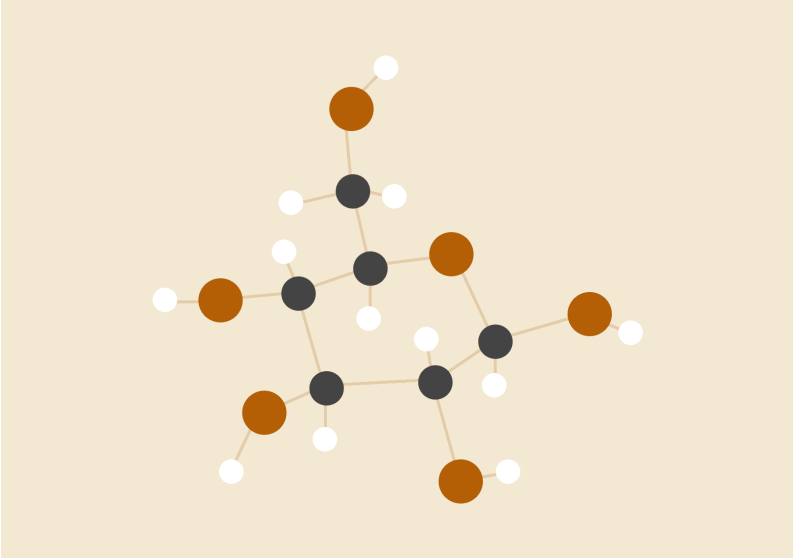
TRABAJO PRÁCTICO ESPECIAL DE PROGRAMACIÓN 3

Entrega de la Segunda Parte



Melisa Burlando <melisa.burlando@gmail.com>

Martina Svedas <svedasmarti@gmail.com>

Profesores a cargo:

- **Améndola, Federico**
- **Casanova, Federico**
- **Merino, Soledad**
- **Vallejos, Sebastián**

02/07/2023

Universidad Nacional del Centro de la Provincia de Buenos Aires

Facultad de Ciencias Exactas

TUDAI

PROBLEMA PLANTEADO

Las autoridades de una ciudad deciden construir una red de subterráneos para resolver los constantes problemas de tráfico. La ciudad ya cuenta con N estaciones construidas, pero todavía no tienen ningún túnel que conecte ningún par de estaciones entre sí. La red de subterráneos que se construya debe incluir a todas las estaciones (es decir, que de cualquier estación H pueda llegar a cualquier otra estación J, ya sea de manera directa o atravesando otras estaciones). Sin embargo, debido al acotado presupuesto, las autoridades desean construir la menor cantidad de metros de túnel posibles. Para esto han calculado cuantos metros de túnel serían necesarios para conectar de manera directa cada par de estaciones existentes.

OBJETIVO

El objetivo de esta segunda parte del trabajo será resolver el problema planteado mediante dos técnicas algorítmicas distintas: Backtracking y Greedy. Luego se deberán comparar los resultados teniendo en cuenta distintas métricas que permitan visualizar, mínimamente, la calidad de la solución y el costo de obtener dicha solución, con ambas técnicas.

RESOLUCIÓN

“¿Cuál fue la estrategia Greedy llevada adelante para resolver el problema? ¿Cuál es el costo computacional de dicha estrategia?”

La estrategia Greedy elegida para resolver él problema planteado fue él algoritmo Prim.

Él algoritmo Prim es un algoritmo cuya función es encontrar él árbol de expansión mínimo de un grafo no dirigido y ponderado, con lo cual es óptimo para encontrar una solución al problema planteado.

Se basa en elegir primeramente un vértice aleatoriamente y almacenar los datos de los arcos qué salen de él en un arreglo de arcos y luego ordenarlos de forma ascendente, esto se realiza de forma iterativa hasta qué todos los vértices del grafo están incluidos en él árbol de expansión, un array de vértices.

Él pseudocódigo correspondiente es él siguiente:

1. Crear un conjunto vacío para almacenar el árbol de expansión mínima (MST).
2. Elegir un vértice inicial arbitrario y agregarlo al MST.
3. Mientras el MST no contenga todos los vértices:

    a. Encontrar él arco de menor peso que conecte un vértice del MST con un vértice fuera del MST.
    b. Agregar él arco al MST.
    c. Agregar el vértice conectado por la arista al MST.
4. Devolver el MST.

Él costo computacional de dicha estrategia es de O(E log V), donde E es el número de aristas y V es el número de vértices en el grafo.

“¿Cuál fue la estrategia Backtracking llevada adelante para resolver el problema? ¿Cuál es el costo computacional de dicha estrategia?”

La estrategia de Backtracking llevada a cabo para la resolución del problema fue cambiando a lo largo del desarrollo, ya que a medida que fueron surgiendo inconvenientes, se los fue tratando de corregir, pero en dos ocasiones hubo que volver a replantear desde cero el diseño del algoritmo al darnos cuenta de que al planteo actual era inviable o muy poco eficiente.

La estrategia final fue elegir todos los arcos del grafo para luego ir seleccionando y pasarlos a una lista de solución parcial, la cual se va usando para intentar llegar a una solución. Esta condición de "solución" se da si encontramos una solución mejor a la anterior (la primera vez no importa) y además validamos que la solución sea conexa, es decir, que además de usar todos los vértices quede todo conectado y no pedazos de caminos separados.

La otra implementación que se intentó, recorría por vértices y obtenía los arcos que partían del mismo. Esta funciona de manera mucho más eficiente pero no estamos logrando llegar al mismo resultado que obtuvimos con en el algoritmo de Greedy, de modo que nos volcamos a esta solución por arcos. (Dejamos en el repositorio esta clase que se llama BacktrackingPorVertices para mostrar qué fue lo que intentamos)

Se hacen dos llamados a backtracking, uno con el arco seleccionado incluído en la solución, y un segundo llamado donde ese arco no es usado.

Al finalizar, se vuelve a poner el arco en la lista de arcos disponibles.

Se intentó un algoritmo de poda más avanzado para intentar evitar repetir escenarios durante la prueba, pero la forma en que se implementó resulta siendo más costosa que el analizar los casos (Vimos que casi se duplican los tiempos de ejecución). Quedó pendiente por si en un futuro se nos ocurre una mejor implementación.

De esta forma la complejidad computacional queda como O(n) donde n es la cantidad de arcos.

TABLA COMPARATIVA

DATASET / ALGORITMO	BACKTRACKING	GREEDY
DATASET 1	Distancia total: 55 kms Ciclos : 125; Tiempo: 3.1888;	Distancia total: 55 kms Ciclos : 7; Tiempo: 0.0981;
DATASET 2	Distancia total: 135 kms Ciclos : 2073; Tiempo: 15.2123;	Distancia total: 135 kms Ciclos: 14 ; Tiempo: 0.1665;
DATASET 3	Distancia total: 440 kms Ciclos : ; Tiempo: ;	Distancia total: 440 kms Ciclos: 45; Tiempo: 1.2351;

Tabla 1: tabla comparativa entre ambos algoritmos

El ciclo en greedy se refiere a la cantidad de veces qué se agrega a solución y en backtracking a la cantidad de veces qué se genera un estado nuevo.

La métrica del tiempo se realizó en una misma computadora bajo las mismas condiciones para qué la comparación entre ambos algoritmos según su tiempo sea la más objetiva posible.

El único inconveniente que tuvimos fue que ninguna de nuestras computadoras pudo correr el dataset 3, pero usamos un cuarto dataset que es más largo que el segundo para chequear que también funcionara con otro.

Por suerte un compañero nos pudo correr nuestro código y confirmarnos que daba bien el dataset 3!

En la tabla podemos observar cómo, si bien por ambos caminos se llega al mismo resultado (esto es gracias a haber utilizado el algoritmo Prim de Greedy, de no haber sido así Greedy nos proveería una aproximación pero no el mejor resultado), el algoritmo Greedy lo hace en un tiempo y ciclos mucho menor al de Backtracking.

GRAFOS DE LOS DATASETS

DATASET1

ESTADO INICIAL:

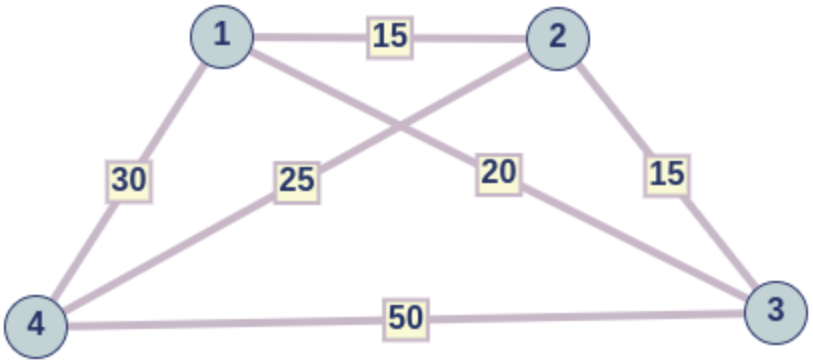


Imagen 1

ESTADO FINAL:

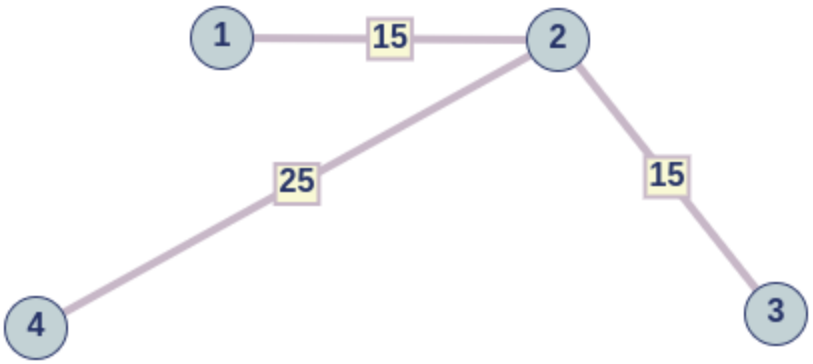


Imagen 2

DATASET 2

ESTADO INICIAL:

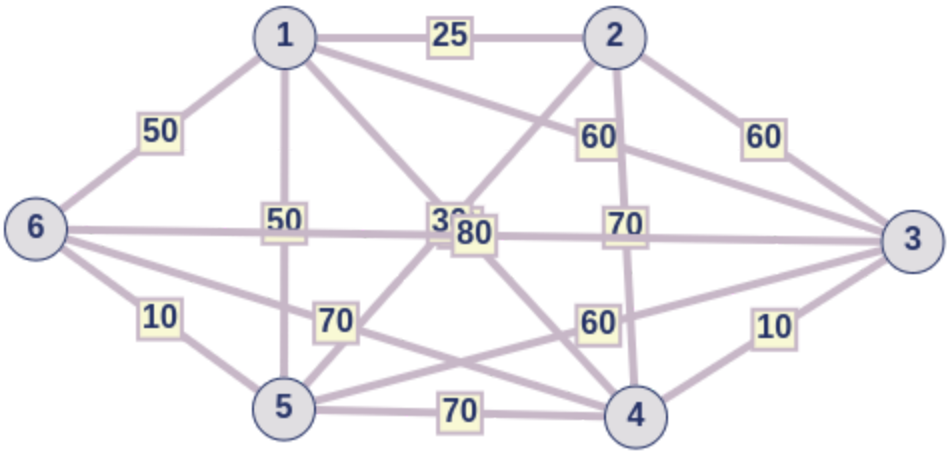


Imagen 3

ESTADO FINAL:

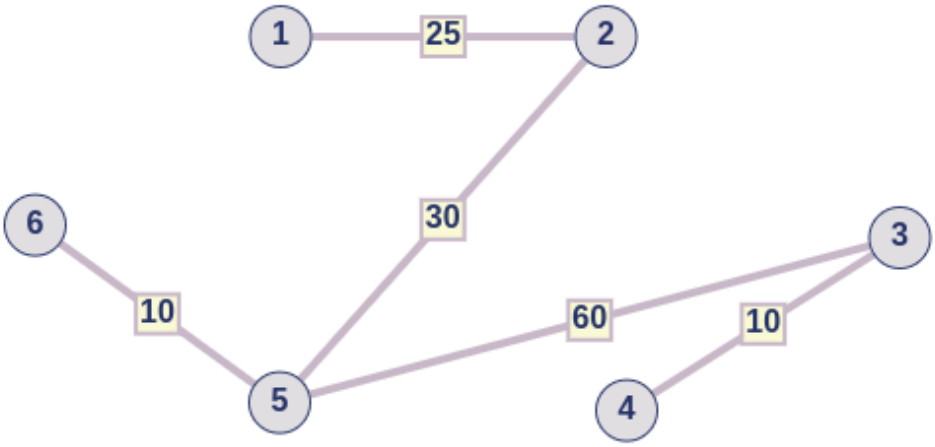


Imagen 4

DATASET 3

ESTADO INICIAL:

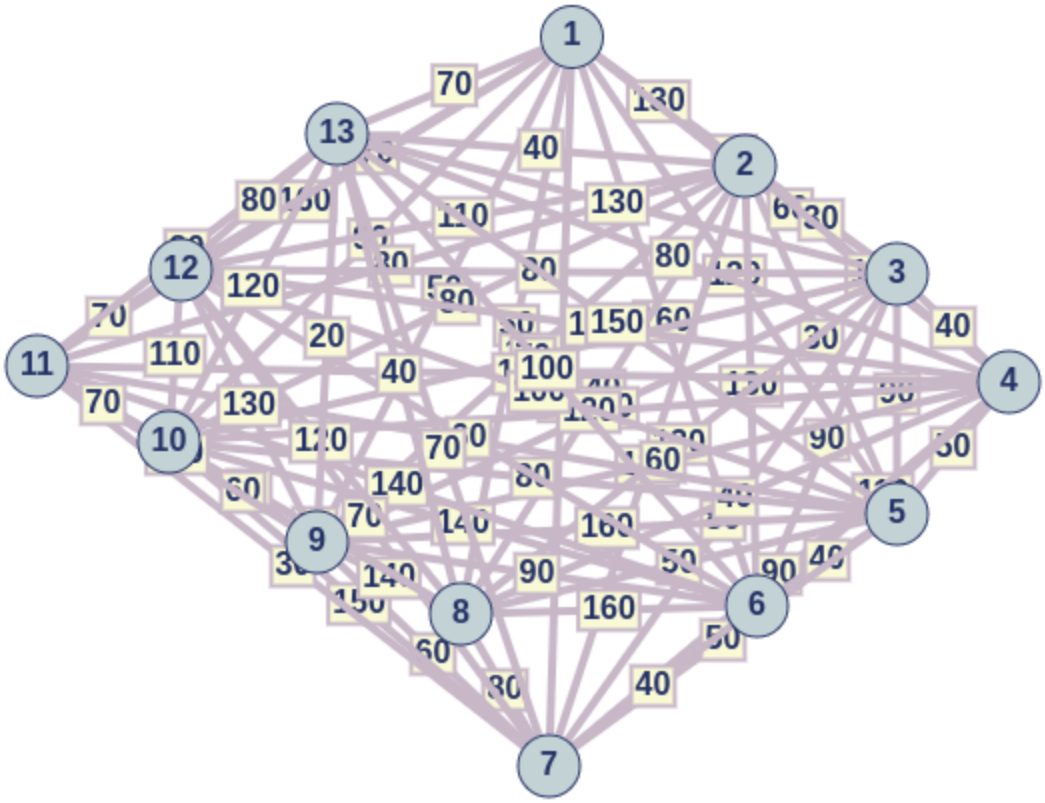


Imagen 5

ESTADO FINAL:

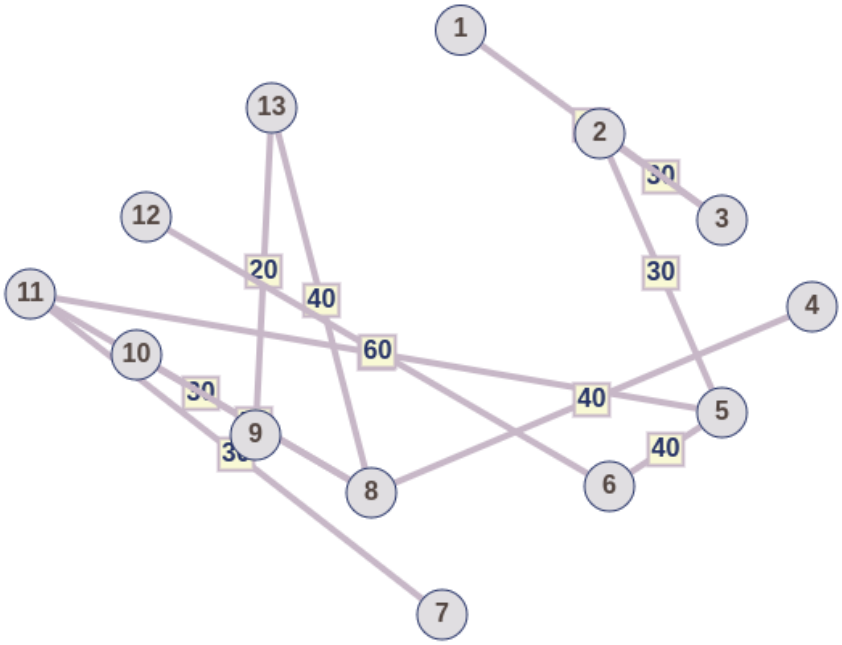


Imagen 6

CONCLUSIÓN

A lo largo del desarrollo del trabajo se buscó implementar un algoritmo qué cumpla con lo requerido de la forma más eficiente posible y qué está solución se muestre correctamente en un tiempo aceptable, especialmente en él caso del backtracking.

Pudimos comprobar qué los resultados tanto del backtracking cómo del greedy son iguales en los 3 dataset otorgados llegando a la conclusión de qué esa solución mostrada es la mejor.

A pesar de qué el backtracking da siempre la mejor solución, en él 3er dataset, el más largo, el tiempo de ejecución resultaba muy excesivo por lo qué se tuvo qué buscar otros criterios de corte para lograr un tiempo más óptimo. Pese a haber encontrado podas eficientes para este problema no hay sido lo suficientemente eficaces ya qué él tiempo de ejecución resulta ser demasiado, además podemos decir qué hay otros casos donde él problema puede ser mucho más largo en el cual no hay podas eficientes qué puedan solucionar él problema del tiempo de ejecución y no resulta conveniente utilizar él método backtracking, por lo qué se implementará greedy para buscar la solución más óptima.