DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP 2: Recorridos y árbol generador mínimo

28 de Mayo, 2023 — Algoritmos y Estructuras de Datos III

Integrante	LU	Correo electrónico
Zaid, Pablo	869/21	pablozaid2002@gmail.com
Arienti, Federico	316/21	fa.arienti@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

http://www.exactas.uba.ar

RESUMEN

En la teoría de grafos, el problema del *árbol generador mínimo*[1] —o *minimum spanning tree*, en inglés— se refiere al problema de encontrar, para un grafo conexo $G = (V,\ E)$ con función de peso $w : E \to \mathbb{R}$ asociada, un subgrafo generador conexo y acíclico de G —es decir, un árbol generador— que minimice la suma total del peso de sus aristas.

Existen diversos métodos para la resolución de este problema. Entre ellos, los algoritmos *golosos* de *Prim* y de *Kruskal*, que se basan en la selección de aristas de peso mínimo *seguras*[2] para la construcción de una solución.

El siguiente informe evalúa el problema de los *módems*, explicado en el próximo apartado, y lo reformula como una generalización del problema del *árbol generador mínimo* que aprovecha el invariante del algoritmo de *Kruskal*. Además, evalúa la eficiencia de la solución propuesta de manera empírica.

Palabras clave: *árbol generador mínimo, algoritmo de Kruskal*.

Índice

[1] Ver Thomas H. Cormen; Charles E. Leiserson; Ronald L. Rivest y Clifford Stein. Introduction to algorithms. 2009. Sección 23: *Minimum Spanning Trees*.

[2] Una arista es *segura* si y sólo si, al agregarla a un subgrafo de un árbol generador, el resultado sigue siendo un subgrafo de algún árbol generador.

1. El problema de los módems

El problema de los *módems* que consideraremos tiene la siguiente premisa. Dado un conjunto de N oficinas

$$O := \{o_1 \ ... \ o_N\}$$

donde, para cada $1 \leq i \leq N$, la oficina o_i se representa por su posición $(x_i,\ y_i)$ en el plano cartesiano —cuya unidad es el metro—, queremos encontrar el costo mínimo que debemos pagar en cables UTP y de fibra óptica para conectar todas las oficinas a internet[3].

Para ello, vamos a contar con $W < N$ módems a distribuir entre las oficinas e incurriremos en un costo de U o V pesos, $0 \leq U \leq V$, respectivamente, por cada metro de cable UTP o de fibra óptica utilizado. Dado que los cables UTP tienen ciertas restricciones, estos se podrán utilizar si y sólo si la distancia entre dos oficinas es menor o igual a R.

Por ejemplo, si

$$O = \{(0,\ 0),\ (0,\ 1),\ (1,\ 0)\},\ R = 2,\ W = 1,\ U = 1 \text{ y } V = 1,$$

una solución podría situar al único módem en la oficina $o_1 = (0,\ 0)$ y conectar esta oficina a las restantes con un metro de cable UTP, por un costo total de dos pesos.

1.1. Modelado como un problema de árbol generador mínimo.

Dada la descripción anterior, vamos a mostrar que el problema se puede modelar como una variante del problema del árbol generador mínimo. Lo detallamos a continuación.

Sea G_{M} un grafo pesado completo cuyos vértices son el conjunto de oficinas O. Por cada par de vértices $u,\ v \in O$, definamos el costo de la arista $(u,\ v)$ como

$$w(u,\ v) = \begin{cases} d_{uv} \cdot U & \text{si } d_{uv} \leq R \\ d_{uv} \cdot V & \text{si no} \end{cases}$$

donde d_{uv} es la distancia euclideana entre ambas oficinas. Notar que $w(u,\ v)$ designa la opción más barata disponible entre un cable UTP y uno de fibra óptica.

Luego, de haber, al menos, una oficina con un módem, G_{M} modela una solución no mínima al problema de los *módems* en la que todas las oficinas están conectadas entre si.

Si $W = 1$, sigue que, para mantener a todas las oficinas conectadas y minimizar el costo empleado en los cables, basta con encontrar un árbol generador mínimo de G_{M}. Sin embargo, si $W > 1$, podemos reducir aún más el costo si, en vez de encontrar un árbol generador mínimo, encontramos un bosque generador mínimo de W componentes conexas que sea un subgrafo de G_{M}. Es decir, un bosque de W árboles que incluya a todos los vértices de G_{M} y tenga peso mínimo.

Vamos a demostrar que este bosque es óptimo y que basta modificar al algoritmo de *Kruskal*, tal que termine en la iteración $N - W$, para encontrarlo.

1.2. Demostración de optimalidad.

Supongamos, por absurdo, que un bosque generador mínimo de W componentes de G_{M} no es una solución óptima al problema de los módems. Luego, debe existir otro subgrafo $B \subseteq G_{\mathrm{M}}$ cuyo peso es el mínimo posible y que, una vez dispuestos los W módems, provee de internet a todas las oficinas.

Sin embargo, B también debe ser un bosque generador de W componentes. Esto se debe a que: si no fuera generador, entonces no estaríamos considerando todas las oficinas en nuestro

[3]Notar que una oficina tendrá acceso a internet si y sólo si tiene un módem o está conectada a una oficina con acceso a internet.

modelo; y, si no fuera un bosque de W componentes, entonces podríamos reducir su peso si eliminaramos suficientes aristas —el peso de toda arista es no negativo en G_{M}— hasta formar uno. Notar que tampoco puede tener más componentes, ya que no habría suficientes módems para proveer de internet a cada grupo de oficinas.

En consecuencia, B es un bosque generador de W componentes conexas de G_{M} que tiene un peso menor que un bosque generador mínimo de W componentes de G_{M}. $\rightarrow\leftarrow$ □

1.3. Demostración de correctitud. Vamos a demostrar primero la siguiente proposición. Dado un grafo conexo G, un subgrafo B de un árbol generador mínimo $T \subseteq G$ es un bosque generador mínimo de k componentes de G si se compone de las $n-k$ aristas de peso mínimo de T.

Demostración. Por propiedad de árboles, está claro que si $B \subseteq T$ tiene $n-k$ aristas, entonces B es un bosque generador de k componentes conexas. Vamos a demostrar entonces, por el absurdo, que es mínimo.

Supongamos que existe un bosque generador B' de k componentes de G que pesa menos que B. Luego, podemos construir un árbol generador de G, a partir de B', si le agregamos un conjunto E de $k-1$ aristas de T que unan a las k componentes conexas diferentes en B'. Esto lo podemos hacer ya que, si tal conjunto no existierá, entonces habría, al menos, un par de vértices, ubicados en dos componentes conexas diferentes de B', para los cuales no existe un camino que los una en T. Lo que es absurdo, dado que T es un árbol generador.

Dicho esto, como estas aristas tienen, a lo sumo, el peso de las $k-1$ aristas máximas en T, sigue que $B'+E$ es un árbol generador de G con peso menor que el árbol generador mínimo T. $\rightarrow\leftarrow$ □

En consecuencia, dado que el invariante del algoritmo de *Kruskal* afirma que, para la k-ésima iteración, el grafo B_k construido es un subgrafo de un árbol generador mínimo de G y, en cada iteración, el algoritmo agrega una arista segura de peso mínimo a B_k, para todo $1 \leq k \leq n-1$, entonces B_{n-k} es un subgrafo de un árbol generador mínimo T compuesto por sus $n-k$ aristas de peso mínimo. Luego, B_{n-k} es un bosque generador mínimo de k componentes. □

1.4. El algoritmo. Dicho todo esto, el siguiente pseudo-algoritmo presenta una solución al problema de los *módems*.

```
1  proc modems(O: sec<ℤ × ℤ>, N, W, R, U, V: ℕ0) → ℝ × ℝ:
2      a, b ← 0, 0
3      G ← grafo completo que modela el problema
4      X ← ∅
5      para i en 1 .. N - W:
6          u, v ← minima arista segura en G para B = (V(G), X)
7          X ← X ∪ {(u, v)}
8          si (u, v) es de tipo UDP:
9              a ← a + w(u, v)
10         si no:
11             b ← b + w(u, v)
12     retornar a, b
```

Algoritmo 1. Pseudocódigo para el problema de los *módems*.

El mismo emplea una versión modificada del algoritmo de *Kruskal* que, además de terminar en la iteración $N - W$, acumula el costo incurrido en cada tipo de cable, en las variables a y b, a medida que agrega aristas al bosque B.

1.5. Complejidad temporal y espacial. El algoritmo MODEMS depende casi exclusivamente de la implementación del algoritmo de *Kruskal* que utilicemos. Dado que el grafo G_{M} es completo, la mejor[4] complejidad que podemos lograr corresponde a utilizar la implementación del algoritmo para grafos densos, cuyo costo espacial y temporal es $\Theta(n^2)$. Esta opción es teóricamente la mejor, ya que las implementaciones para grafos *ralos* tienen un costo temporal en $\Theta(m \log n)$, que equivale a $\Theta(n^2 \log n)$ para este caso de uso.

2. Evaluación empírica

Para revisar de manera empírica la diferencia en eficiencia entre una implementación de MODEMS sobre el algoritmo de *Kruskal* para grafos *densos*[5] y una implementación sobre el mismo algoritmo para grafos *ralos*[6], procedimos a implementar[7] ambas versiones en C++ y realizamos una serie de evaluaciones respecto al tiempo de ejecución en función del tamaño de la entrada, para muestras aleatorias de tamaño $N = 1000k$ para cada k natural en el rango $1 \leq k \leq 10$.

Realizamos cada evaluación diez veces para reducir la variación de los resultados y tomamos el promedio aritmético. A su vez, controlamos los parámetros de la función de la siguiente forma. Elegimos $W = N/2$, $R = 1$ y $W = V = 5$, y definimos la posición de cada oficina $o_i = (x_i,\ y_i)$, para todo $0 \leq i \leq N$ y $0 \leq x_i,\ y_i \leq N$ aleatorios, tal que $x_i \neq x_j$ y $y_i \neq y_j$ para todo $1 \leq i,\ j \leq N$ e $i \neq j$.

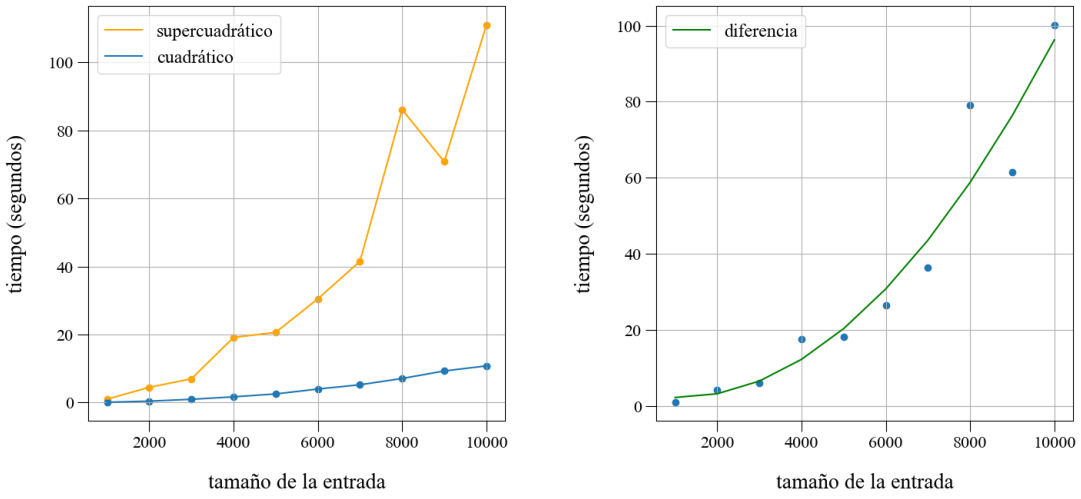

FIGURA 1. Izquierda: tiempo de ejecución de MODEMS en función del tamaño de entrada N para el algoritmo *cuadrático* —en base a *Kruskal* para grafos densos— y el *supercuadrático* —en base a *Kruskal* para grafos ralos—. Derecha: la diferencia absoluta entre ambas mediciones descripta por una regresión cuadrática.

[4]En base a los algoritmos conocidos.

[5]Para este algoritmo, nos basamos en la especificación propuesta por Federico Lebron en su blog.

[6]Ver nota al pie 1. Sección 23.2: *The algorithms of Kruskal and Prim.*

[7]Los experimentos, algoritmos y archivos resultantes se pueden encontrar en *./ej-3/experimentacion.*

Notar que ninguna de estas decisiones afecta la complejidad asintótica del algoritmo. Sin embargo, un analisis en mayor profundidad debería, también, evaluar el comportamiento de ambos algoritmos en función de estos parámetros.

La Figura 1 expone los resultados. Podemos notar que hay una clara diferencia de velocidad a favor del algoritmo cuadrático, incluso en las entradas de menor tamaño. La regresión se realizó con cuadrados mínimos y obtuvo un error de $\approx 752{,}32$.

2.1. Heurísticas para la implementación de disjoint set. Del mismo modo que realizamos la comparación anterior, también vale la pena evaluar la diferencia de tiempos que se pueden obtener al utilizar la estructura de *disjoint set*, sin alguna de sus optimizaciones, en el algoritmo supercuadrático. En este caso obviaremos las heurísticas de *union by rank* y *path compression*.

Esta segunda versión del algoritmo utiliza la implementación de *Kruskal* para grafos *ralos* sobre una estructura ingénua de *disjoint set*: a la hora de unir dos conjuntos, recorremos uno —elemento por elemento—, y lo agregamos al otro. Se puede demostrar, en base al análisis que realiza [Cormen][8], que el algoritmo resultante tiene una complejidad temporal en $\Theta(m \log n + m + n^2)$, correspondiente al costo de ordenar las aristas del grafo de entrada, el costo de recorrerlas y el costo amortizado de realizar $\Theta(n)$ operaciones sobre esta estructura de datos. En comparación, la versión utilizada en la sección anterior tiene una complejidad temporal en $\Theta(m \log n + m\alpha(n))$, donde α es la función inversa de *Ackermann*. Si bien esto no modifica la complejidad del algoritmo, ciertamente resulta posible que afecte su tiempo de ejecución.

En forma análoga a la experimentación anterior, hicimos una prueba empírica con las dos versiones de esta implementación. La Figura 2 muestra los resultados.

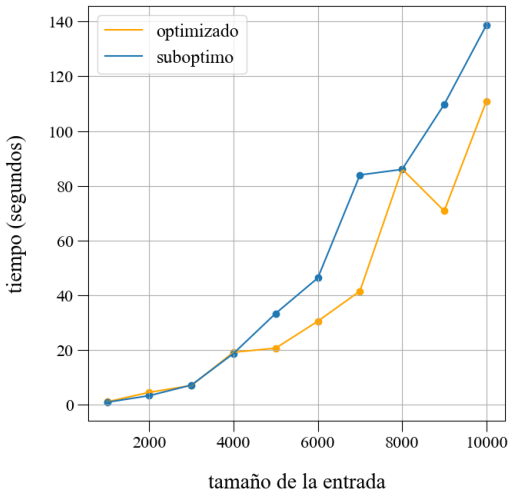

FIGURA 2. Tiempo de ejecución del MODEMS en función del tamaño de entrada n para el algoritmo supercuadrático *optimizado* —con las heurísticas— y *subóptimo* —o, ingénuo—.

Notamos que, en general, la versión con las heurísticas parece funcionar mejor. Esto puede deberse a las constantes *invisibles* asociadas a cada algoritmo, ya que $\Theta(n^2) < \Theta(m\alpha(n))$ si el grafo de entrada es *denso*. Es decir, cuando $m \in \Theta(n^2)$.

[8]Ver nota al pie 1. Sección 21: *Data Structures for Disjoint Sets*.