DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Recorridos y árbol generador mínimo

30 de mayo de 2023 — Algoritmos y Estructura de Datos III

Integrante	LU	Correo electrónico
Kim, Lucas	456/20	lucasthkim99@gmail.com
Ramírez, Leandro	90/09	leandroroman.cito@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

https://exactas.uba.ar

Índice

1. Introducción

1.1. Presentación y descripción del problema

El informe está destinado a presentar y describir la resolución del punto 3 del trabajo práctico. Este ejercicio consiste en conectar las n oficinas pasadas como parámetro siendo coordenadas, con un máximo de W módems. Y una vez ubicados los módems, conectar las oficinas entre sí con cables y estimar el costo de ello. Es decir, la resolución del problema reside en modelar las oficinas y optimizar el posicionamiento de cableado para ver el mínimo costo que toma conectar las n oficinas. Además, se tuvo que tener en cuenta que dependiendo de como se conectasen los cables, el precio podría variar al existir dos tipos de cables diferentes.

Como input del problema, contamos con la cantidad de oficinas, sus respectivas coordenadas en el eje cartesiano, w como la cantidad de módems, el precio de ambos cables por centímetro y por último el límite de distancia que tienen los cables UTP, para saber si es posible de usar para cada caso o no.

1.2. Como compilar

Se dispone junto al informe, una carpeta comprimida con el código de cada ejercicio. Para compilar, usar un compilador de C++ usando el archivo de texto ej3.cpp. Luego al ejecutar el binario, la instancia a correr se pasa por estándar input o por un archivo.txt usando el operador $\ll$ de la terminal. Ejemplo:

```
g++ ej3.cpp -o ej3
./ej3 < instancia.txt
```

2. Algoritmo

La idea es tratar de modificar el algoritmo de Kruskal de árbol generador mínimo para encontrar un conjunto solución para la instancia de problema. Para ello, tuvimos que modelar el grafo al que se le intentaría correr el algoritmo de Kruskal.

Como vértices tendríamos a las n oficinas dispuestos por sus coordenadas cartesianas y el objetivo sería poder conectarlas de modo que entre cada grupo de oficinas cableadas tuviese un módem. Además, dicho grafo es un grafo completo ya que existe la posibilidad de conectar cada oficina con cualquier otra oficina dentro del grafo.

Sin embargo, el problema no se trata de conseguir el árbol generador mínimo porque no se necesita que haya un camino entre cualquier par de nodos, es decir, que todas las oficinas estén cableadas entre sí. Sino, al disponer de W módems, alcanza con tener "grupos"de oficinas con cables entre sí igual a la cantidad de módems disponibles. Tampoco importará para el problema saber en donde está colocado un módem, tener las n oficinas agrupadas en W grupos donde exista un camino entre cada oficina del mismo grupo es suficiente para asumir que se logró proveer internet a cada oficina.

Llevar esta idea al grafo implícito formado por las oficinas sería querer encontrar el bosque de W componentes conexas, para ubicar en cada componente conexa y tener así una solución del problema. El costo de los cables sería el equivalente a haber usado las aristas para conectar las componentes conexas. Asimismo, calcular el costo una vez elegidas las aristas, sería trivial ya que dependiendo de la longitud de cada arista, esta se multiplica por el precio de los cables UTP o de fibra óptica.

En otras palabras, podemos pasar de un problema de conexión de oficinas a un grafo implícito buscando generar W componentes conexas y que en distancia, las aristas sean las de menor longitud.

Por lo tanto, podemos usar una modificación del algoritmo de Kruskal. Este, busca generar el árbol generador mínimo de un grafo, contemplando las aristas en orden creciente y decidiendo de agregarlas o no sólo si al grafo con la nueva arista tiene menos componentes conexas que anteriormente.

De este modo, podremos modificar Kruskal cortando el algoritmo antes de formar un árbol, tener W componentes conexas, a las cuales se le destinaría un módem y por último, calcular el costo de cable UTP y fibra óptica dependiendo de la longitud de las aristas.

Por lo tanto, el algoritmo seguiría esta forma:

```
process(V,E):
    generar las aristas desde cada oficina a todas las otras(V)
    ordenar de forma creciente E
    iterar en el vector de aristas:
        Si agregar la iésima arista disminuye las c.c: la agrego
    Calcular los costos de cada arista agregada a mi bosque de c.c.
```

2.1. Detalles de implementación

Para la implementación del algoritmo, usamos tuplas para representar las coordenadas cartesianas de las oficinas. Luego las distancias al ser euclídeas, números reales, usamos el tipo de dato double para guardar el costo de las aristas, en este caso, sus distancias.

Nuestro algoritmo, al ser una modificación de Kruskal, fue necesaria la implementación de algún método de ordenamiento, en nuestro caso, la función sort de la biblioteca estándar y la estructura de datos DSU, en su versión optimizada.

2.2. Demostración de correctitud

En esta sección se procederá a la demostración de correctitud del algoritmo propuesto. Para tal fin, necesitaremos demostrar por qué tomar el problema de las oficinas como un grafo implícito es correcto y cómo el algoritmo de Kruskal, un algoritmo para obtener un árbol generador mínimo y la modificación hecha a la misma resolvería el problema planteado.

La instancia del problema a resolver consta de n oficinas, cada una ubicada coordenadas cartesianas (x,y) donde queremos conectarlas con cables. De este modo, al tener que decidir que cables usar y cuales no, estamos implícitamente eligiendo un subgrafo posible dentro del grafo donde cada oficina es un vértice y además es completo ya que es posible conectar con cables cualquier par de oficinas.

Una vez modelado el grafo, podremos definir la función de costo para cada arista del grafo como la distancia euclídea e inclusive, estaríamos definiendo los costos de cada arista, ya que una vez obtenida la distancia, podremos saber si dada la restricción, si es UTP o de fibra óptica.

Luego, tenemos un grafo definido con un conjunto de vértices y uno de aristas donde cada arista tiene un peso definido. Nuestro objetivo es, entonces, poder determinar el subgrafo de G donde la suma del costo de las aristas dentro del subgrafo sea el mínimo, pero a diferencia del problema del árbol generador mínimo, no es necesario obtener un árbol. Esto se debe a que el ejercicio consta de un número W de módems disponibles

donde $W < N$. En el caso de W = 1, el problema se reduce a encontrar el árbol generador mínimo, pero en caso contrario, podemos tener W componentes conexas.

Hasta ahora, pudimos condensar el ejercicio en obtener nuestro grafo y encontrar las W componentes conexas que minimicen el costo de los cables. Además, por la índole del problema estos serían árboles y mas aún, arboles generadores mínimos del subgrafo de G que contiene a esos nodos. Los W AGMs también serán subgrafos del árbol generador mínimo de G,y por ende, usando el invariante de Kruskal podemos demostrar que nuestra modificación del algoritmo resuelve el problema de manera óptima.

Invariante de Kruskal: Dado G = (V, X) un grafo conexo. $T_k = (V, X_{Tk}), 0 \leq k \leq n-1$, es bosque y subgrafo de un árbol generador mínimo de G.

Para cada k-ésima iteración de kruskal donde se decide agregar una arista tal que la cantidad de componentes conexas disminuya, el subgrafo generado es un bosque de N - k árboles y además es un subgrafo de un AGM de G. Esto se debe a que en la iteración 0, contamos con un subgrafo sin aristas y por ende N componentes conexas que por cada iteración tiene una componente conexa menos (unimos en cada iteración las componentes conexas u y v de la arista $e = (u, v)$). Por ende, luego de la iteración (n - w), tenemos un bosque de W componentes que es subgrafo del AGM de G. Luego, al colocar un módem en cada componente, podemos asegurar que las aristas al pertenecer a un subgrafo del AGM de G corresponden al conjunto que hacen al costo de los cables mínimo.

$\#CompConexas_k = N - k \Rightarrow \#CompConexas_{N-w} = N - (N - w) = w$

2.3. Complejidad

Las dos grandes funciones usadas para la resolución del ejercicio son generar las aristas y luego el algoritmo de Kruskal. Generar las aristas es $O(n^2)$ para grafos completos, ya que por cada arista tendremos n - 1 aristas incidentes a la arista. Luego, Kruskal debe su complejidad $O(m * logn)$ a iterar por cada arista ($O(m)$) y luego por cada arista, verificar si pertenecen o no a la misma componente conexa. Dependiendo de como se implementa la estructura de DSU y sus optimizaciones, esta última parte del código podría empeorar la complejidad. En nuestro caso, utilizamos la optimización *Union by size* la cual intenta mantener el árbol de padres para cada componente conexa de forma balanceada. De dicha forma, se logra usar los métodos find y unite en tiempo logarítmico a la cantidad de nodos. $\Rightarrow$ El algoritmo es $O(m * logn)$.

Sin embargo, este implementación de Kruskal es eficiente para grafos ralos ya que en grafos completos donde las arista son $m = O(n^2)$, la complejidad del algoritmo termina siendo, $O(n^2 * logn)$. En conclusión, nuestro algoritmo para resolver el problema de módems es $O(n^2 * logn)$.

2.4. Casos de test y resultados

En esta sección, intentaremos estudiar y comparar empíricamente la complejidad del algoritmo como también comparar distintas implementaciones del algoritmo de Kruskal. En el inciso anterior, mencionamos que la implementación de Kruskal usada en la resolución es una versión optimizada mediante un buen manejo de la estructura de ancestro en DSU. Luego, nos proponemos comparar empíricamente el rendimiento de DSU con optimización versus sin optimización.

Para ello, modificamos el código del ejercicio para usar la biblioteca chrono de C++ para medir el tiempo de ejecución de nuestro algoritmo y fue comparada con la versión

de Kruskal para grafos densos. Para los casos de test, usamos un script en python para generar de forma aleatoria las coordenadas de las oficinas y la cantidad de módems para distintos N.

Ciertamente, deberíamos notar mejor *performance* en la implementación de DSU con *Union by rank* al manejar no determinísticamente las estructuras de datos. En el caso de optar por no usar la optimización, no tenemos asegurado que el árbol de ancestros se encuentre balanceado. En otras palabras, encontrar el nodo representante de cada componente conexa en el peor caso es $O(n)$.

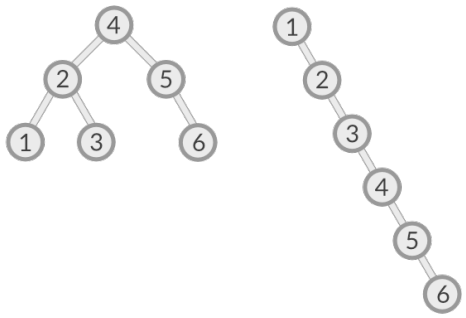
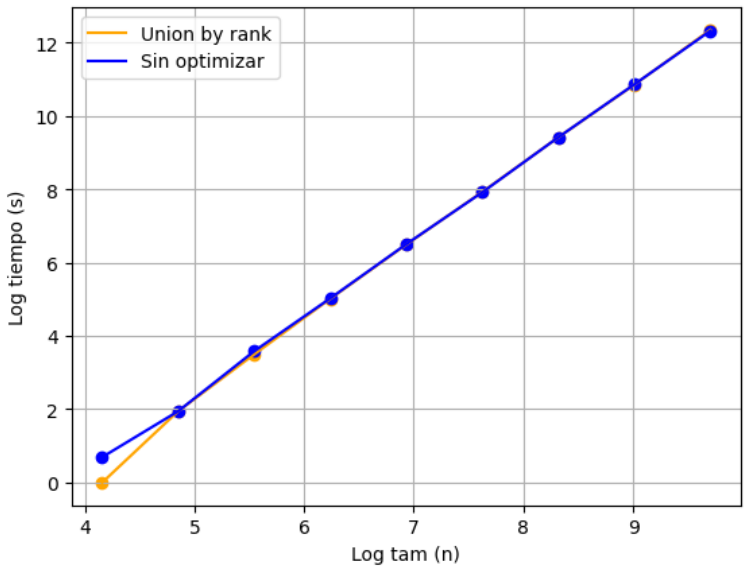


Figura 1: Dos posibles árboles de ancestros para representar los padres de una componente conexa. A la izquierda, un árbol balanceado con Union by rank. A la derecha, un árbol desbalanceado.

Por lo tanto, para cada iteración donde se determina si agregar o no una arista, no podemos asegurar que el método find de DSU sea logarítmico con respecto a la cantidad de componentes conexas, por lo cual el algoritmo termina siendo $O(m * n)$ donde para grafos completos, es $O(n^3)$.



Podemos observar que ambas implementaciones en la práctica no difieren en el tiempo de ejecución tomado a pesar de que teóricamente, la versión sin optimización es asintóticamente peor. Esto puede deberse a que la cota $O(n)$ para encontrar el padre de una componente conexa sólo es lineal en el peor caso, en el cual unir dos componentes conexas,

deja al árbol de ancestros con altura lineal. Sin embargo, este caso sólo depende de como se construye el árbol, lo cual no es determinístico y depende sólo del orden de la instancia.

En conclusión, a pesar de que la versión optimizada de DSU teóricamente mejora el rendimiento del algoritmo, en la práctica puede no presentar ninguna mejora.