Algoritmos y Estructuras de Datos III

Trabajo Práctico 2 - Recorridos y árbol generador mínimo

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Informe

Integrante	LU	Correo electrónico
Carrasco, Andrés Nicolas	1905/21	nicocarrasco703@gmail.com
Moraut, Tobias	1507/21	tobiasmoraut7@gmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Introducción

1.1. Enunciado

Se quiere proveer de internet a N oficinas del subsuelo del pabellón $0 + \infty$. Para ello, se compraron W modems, los cuales pueden instalarse en cualquier oficina, brindándole acceso a internet.

Como $W < N$ se deberán comprar también cables para conectar algunas oficinas entre si. Se pueden comprar cables UTP o de fibra óptica, los cuales tienen un costo por metro de U y V , respectivamente. Los cables UTP tienen la condición particular de que solo pueden usarse entre dos oficinas si estas están a menos de R centímetros. Dadas las posiciones de las oficinas en un eje cartesiano (expresadas en centímetros), la cantidad de modems adquiridos, el precio por metro de los cables UTP y de fibra óptica, y la distancia R, debemos encontrar cuál es la mínima cantidad de dinero que debe pagarse en cables para conectar todas las oficinas. Puntualmente, queremos saber cuánto debe gastarse en cables UTP y cuánto en cables de fibra óptica.

2. Algoritmo

2.1. Explicación

2.1.1. Parametros de entrada y salida

Entrada

Como primer parametro, se toma un entero C que indica la cantidad de casos de test que se desea resolver. Luego, cada caso está compuesto por cinco enteros: N, R, W, U y V, donde $1 \leq N \leq 1000$ denota la cantidad de oficinas del pabellón (o, en otras palabras, cuantos vértices tendrá el grafo), $1 \leq R \leq 10000$ la restricción de distancia a partir de la cual no se pueden usar cables UTP, $W < N$ la cantidad de modems adquiridos y $1 \leq U \leq V \leq 10$ el precio por centímetro de los cables UTP y de fibra óptica. Siguen N lineas consecutivas, las cuales contienen cada una, 2 enteros x e y. Los enteros (x, y) de la i-ésima línea indican la posición de la i-ésima oficina dentro de un eje cartesiano, expresada en centímetros.

Salida

Por cada caso de test provisto, se retorna una linea que indica el número de test que estamos observando, junto con el costo total de los cables UTP en primer lugar, y luego el costo total de los de fibra óptica. Ambos precios están reducidos a 3 cifras decimales.

2.1.2. Descripción del algoritmo

Comenzamos tomando los valores de entrada para el grafo, tal como se explicitó anteriormente. Procedemos a crear un grafo, armamos vértices que están identificados con coordenadas de un eje cartesiano. A su vez, creamos los vecinos de cada vértice y les asignamos una arista que los conecte. Esta arista tendrá peso, el cual estará definido por la distancia entre los dos nodos que conecta multiplicada por el costo del tipo de cable que representa. Al tener los cables UTP una restricción que establece que deben tener una longitud menor a cierta cota y, a su vez, su costo será siempre menor o igual a los cables de fibra, definimos que si el cable es efectivamente menor a dicha cota. La arista será la distancia multiplicada por el costo de UTP, y si el cable es de mayor longitud, el peso de la arista tendrá el valor de su longitud multiplicado el costo de los cables de fibra. Al realizar esto también le asignamos un identificador (un valor `char`) a la arista que indica si esta representa a un cable UTP o uno de fibra.

Luego, corremos el algoritmo de Kruskal sobre el grafo generado. Este nos devuelve su árbol generador mínimo, al que llamaremos T. Ahora, en T tenemos las conexiones entre oficinas más baratas, estos costos serán los mínimos que gastaremos si deseamos tener todas las oficinas conectadas.

Sin embargo, sabemos que tenemos W modems, por lo que no requerimos que todas las oficinas estén conectadas (en otras palabras, que el grafo que modela a las oficinas sea conexo). Luego, eliminamos las $W-1$ aristas más pesadas, quedándonos así con un grafo disconexo pero con el costo minimizado de cada tipo de cable.

Por ultimo, recorremos las aristas del grafo resultante, verificando el tipo de cable cada una. Dependiendo el caso, sumaremos al costo total del tipo de cable, el costo de cada arista.

2.2. Justificación de correctitud

Como vimos anteriormente, la resolución está basada principalmente en el algoritmo de Kruskal, retornando este un AGM. El invariante de Kruskal nos dice que, sea el grafo G en la iteración i tenemos un bosque T de i aristas que, a su vez, es un subgrafo de un AGM de G. Este invariante está demostrado en la teórica y nos garantiza que, al finalizar las iteraciones, tendremos un AGM de G.

En nuestro algoritmo, Kruskal nos devuelve un vector de aristas que son parte del AGM, de forma ordenada. El siguiente paso que realizamos al finalizar Kruskal y obtener dicho vector (al que a partir de ahora lo llamaremos V) de aristas es eliminar las $W-1$ aristas más grandes. Eliminamos esta cantidad y no W ya que como un AGM es un árbol, tiene un total de $N-1$ aristas, y nosotros deseamos obtener al final un bosque con exactamente W componentes conexas para así, instalar un modem en cada una de ellas. De esta manera, no solo nos queda un bosque de W componentes conexas, sino que las aristas que estén presentes dentro de cada componente conexa serán las más baratas posibles que garanticen la conectividad (esto lo sabemos gracias a Kruskal).

Luego, la respuesta final que deseamos consiste en sumar los costos en los distintos tipos de cable, UTP y fibra. Queremos devolver el mínimo que gastaremos con cada uno, y, tal como vimos, Kruskal nos garantizaba el mínimo costo de aristas. Esto abarca a ambos tipos de cable, y no podremos tener cables de menor costo ya que, de lo contrario, no hubiera sido posible generar un AGM con menor costo al cual luego eliminarle las aristas mas pesadas. Así, terminamos recorriendo V luego de ser modificado y sumamos el precio de cada cable dependiendo de su tipo, dejándonos el precio total gastado de cada cable al terminar de recorrerlo.

3. Experimentación con distintas implementaciones de Kruskal

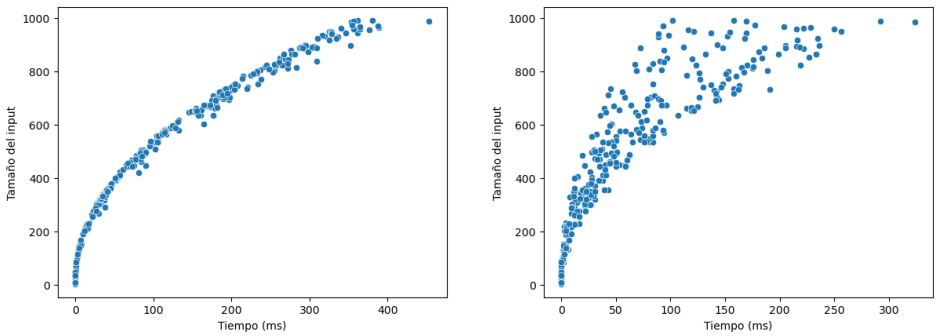
3.1. Kruskal con DSU vs. Kruskal para grafos densos

Nuestro algoritmo se basa en la implementación de Kruskal con *Disjoint Set Union* (DSU). Sin embargo, al utilizar la implementación de Kruskal para grafos densos[1], obtenemos un aumento de performance.

Analizando las complejidades de ambas implementaciones, podemos observar que Kruskal para grafos densos tiene una complejidad de $O(n^2)$. Por otro lado, vemos que Kruskal con DSU tiene una complejidad de $O(m \cdot \log(n))$.
No obstante, como armamos un grafo denso (pues cada oficina debe estar conectada con las demás), tenemos que $m = O(n^2)$[2], por lo tanto $O(m \cdot \log(n)) = O(n^2 \cdot \log(n)) > O(n^2)$.

En la práctica, esta mejoría se hace evidente. Al correr el algoritmo con ambas implementaciones en el juez, obtuvimos que Kruskal con DSU pasa los test en $0.25s$, mientras que Kruskal para grafos densos tarda $0.04s$.

Podemos visualizar la diferencia de performance entre ambas implementaciones al observar los siguientes gráficos:



(1) Kruskal con DSU (2) Kruskal para grafos densos

Figura 1: Gráficos de tamaño del input vs. tiempo de ejecución

Luego de comparar los tiempos de cada caso de test entre las dos ejecuciones, pudimos concluir que, en promedio, el algoritmo que utiliza la implementación de Kruskal para grafos densos resuelve el mismo test un 44 % más rápido que el implementado con DSU.

[1] Basada en la implementación de https://fedelebron.com/a-dense-version-of-kruskals-algorithm#dense-implementation
[2] Abuso de notación

3.2. Kruskal con DSU: Comparativa con distintas optimizaciones

Para esta comparativa, utilizamos distintas implementaciones de DSU.

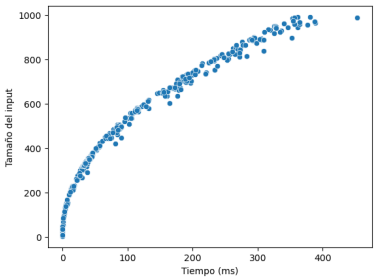
(a) DSU con *union by rank* y *path compression*

(b) DSU con *union by size* y *path compression*

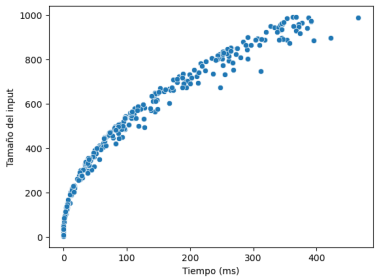
(c) DSU con *union by rank* sin *path compression*

(d) DSU sin ninguna optimización

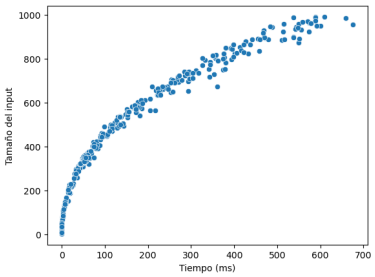
Luego de correr varios casos de test, obtuvimos diversos resultados para cada implementación. Podemos ver en los siguientes gráficos los distintos tiempos de ejecución



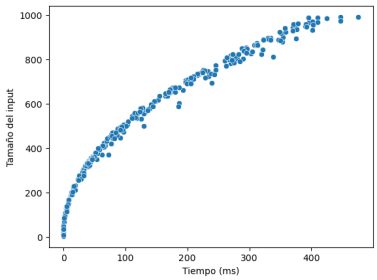
(a) DSU con *union by rank* y *path compression*



(b) DSU con *union by size* y *path compression*



(c) DSU con *union by rank* sin *path compression*



(d) DSU sin ninguna optimización

Figura 2: Comparativa gráfica entre las distintas optimizaciones de DSU

Al calcular la diferencia de los tiempos entre cada implementación. Pudimos concluir lo siguiente

	Tiempo[3]	Diferencia aproximada[4]
(a)	0.25	-
(b)	0.27	+5 %
(c)	0.28	+11 %
(d)	0.47	+45 %

[3]Tiempo en pasar el juez
[4]Porcentaje promedio de tiempo adicional para cada test, en comparación con DSU optimizado