DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2: Modems

Algoritmos y Estructiras de Datos III

Integrante	LU	Correo electrónico
Lucia Ariadna Fernandez	832/19	lucif268@gmail.com
Julieta Giri	311/20	julietagiri.uba@gmail.com

Resumen : El objetivo del presente trabajo práctico es plantear y resolver un ejercicio de programacion utilizando la tecnica de algoritmos golosos. Se describira el problema, se desarrollara la solucion conjunta a un pseudocodigo de la misma para luego demostrar su correctitud y exponer un conjunto de instancias para probar su complejidad luego de analizarla.

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

1. Introduccion

El presente informe describirá y resolverá el ejercicio de modems del segundo trabajo práctico de la materia Algoritmos y estructura de Datos III. El objetivo del mismo es explicar el problema y resolverlo utilizando el algoritmo de kruskal para AGMs.

1.1. Presentación y descripción del problema

El ejercicio plantea el problema de proveer de internet a las oficinas del pabellón 0+Infinito de la facultad minimizando el costo total de la instalación, utilizando una cantidad limitada de modems que serán instalados en oficinas claves, y luego conectando los modems en esas oficinas a otras oficinas con cables UTP o cables de fibra óptica. El costo de utilizar cables UTP (U) siempre será menor o igual al costo de utilizar cables de fibra óptica (V), de forma que siempre que sea posible se elegirá utilizar cables UTP para minimizar el costo total, pero habrá casos en los que utilizar estos cables no será posible, ya que éstos solo pueden usarse si la distancia entre las dos oficinas que ese cable conecta es menor o igual a R centímetros, siendo R uno de los inputs del problema que tendrá un valor entre 1 cm y 10.000 cm.
Las oficinas están posicionadas en un eje cartesiano y expresadas en centímetros con números enteros, y la cantidad de oficinas sobre las que se trabajará (N) estará en un rango de 1 a 1000, y la cantidad de modems disponibles para la instalación (W) será estrictamente menor a la cantidad de oficinas y mayor o igual a 1, de forma que nunca podrá instalarse un modem en cada oficina y siempre será necesario utilizar algún tipo de cable. Además, el costo de utilizar cualquiera de los dos cables siempre será $1 \leq U, V \leq 10$.
Finalmente, el objetivo es encontrar cuál es la mínima cantidad de dinero que debe pagarse en cables para conectar todas las oficinas y devolver cuánto debe gastarse en cables UTP y cuánto en cables de fibra óptica.

2. Resolución del problema y pseudocodigo

Dado el problema de minimización del costo de los cables, podemos modelar la situación tomando cada oficina como un nodo y la distancia entre cada una como el peso de una arista.

Una vez obtenidos los datos de entrada; N, R, W, U, V y la lista de coordenadas de cada oficina que llamaremos *Posiciones* lo primero que debemos hacer es calcular las distancias entre ellas. Nótese que como no sabemos en qué oficina particular se colocaran los modems debemos saber las distancias de todas con todas, lo cual nos lleva a trabajar con un grafo completo. Guardamos las distancias entre todas las oficinas en una lista que llamaremos *distancias*.

Luego, lo que tenemos como ya dijimos es un grafo pesado completo, cuyos nodos son las oficinas y los pesos de las aristas son las distancias entre todas ellas.

El enunciado nos pide que dados W modems, con WN devolvamos el mínimo costo de cables ya sean UTP o de fibra óptica necesarios para que todas las oficinas tengan conexión a internet, dado que UTP es siempre más barato que fibra pero tiene la restricción de que la distancia debe ser menor a R. Para lograrlo, en principio corremos el algoritmo de Kruskal sumando los gastos de cables de fibra o te UTP cuando corresponda; dada la lista de aristas ordenadas, tomamos la de menor costo que no hayamos ya puesto en el árbol y que no genere ciclos entre las que ya pusimos y si su costo es menor a la cota de UTP entonces lo conectamos ese tipo de cable, si no cumple con la restricción no nos queda otra que poner fibra óptica. De esta manera vamos conectando todas las oficinas con el tipo de cable más conveniente dada su distancia.

Una vez que terminamos el algoritmo, lo que nos queda es un grafo conexo sin ciclos, de costo mínimo, es decir un **Árbol Generador Mínimo** que conecta todas las oficinas usando la opción de cable más barata. Pero esto no es exactamente lo que queremos, pues equivale a tener un solo modem que da internet a todas, no estamos aprovechando la cantidad de modems disponibles.

Dado el modelado de la situación propuesto, podemos ver que poner un modem en alguna oficina implica que ya no sea necesario que esté conectada a otra que le dé internet, luego, colocar un modem implica dejar a ese nodo como una componente conexa extra, así haciendo que nuestro árbol generador mínimo se vuelva solo un grafo disconexo sin ciclos.

Lo único que resta es determinar en que oficina nos conviene colocarlo, y como colocarlo en algún lado implica que saquemos cables, lo más conveniente es ponerlo en la oficina que sea parte de la arista más grande que nos quedo en el árbol para poder seguir minimizando el costo. En general las aristas que se irán son las que usaban fibra óptica, pues en las distancias más grandes no nos queda otra que usar ese tipo de cable que además es el más caro. Por otro lado además de eliminar las aristas más pesadas del AGM lo que debemos hacer es restar el costo del presupuesto que veníamos sumando en el algoritmo de kruskal; por lo tanto mientras recorremos las aristas más pesadas para sacarlas y poder la cantidad de modems permitida, determinamos cuál era el tipo de cable elegido para esa y restamos su costo del presupuesto.

De esta manera nos quedamos con un grafo disconexo sin ciclos con las menores distancias entre las oficinas y cada componente conexa representa que alguna de ellas, no importa cuál, es la que tiene el modem que alimenta el resto.

Algorithm 1 kruskal : (**array** distancias)

```
1: sort(distancias)
2: for arista in distancias //asumiendo que es arista segura
3: if distancia < R then
4:     gastoUTP ← gastoUTP + costoUTP * distancia
5: else
6:     gastoFibra ← gastoFibra + costoFibra * distancia
7: AGM.agregar(arista)
8: return AGM
```

La implementación de kruskal encuentra una arista segura, es decir la menor distancia entre oficinas que no genere un ciclo y dado su valor decide si aumenta el presupuesto de gasto de cables UTP o cables de fibra, teniendo en cuenta la restricción R.

Algorithm 2 modems : (**array** coordenadas)

```
1: distancias ← calcularDistancias(coordenadas)
2: AGM ← kruskal(distancias)
3: for i in W
4: if distancia[n-1-i] < R then
5:     gastoUTP ← gastoUTP − costoUTP * distancia[n-1-i]
6: else
7:     gastoFibra ← gastoFibra − costoFibra*distancia[n-1-i]
8: return gastoUTP, gastoFibra
```

Una vez que removemos W cantidad de aristas de nuestro AGM tenemos un costo de cables aun menor al que nos retornó kruskal, ya que utilizamos eficientemente la cantidad de modems que nos habían dado.

3. Experimentación

Para la posterior experimentación se utilizaron tres implementaciones distintas de programas que resuelven el problema planteado:

3.1. Kruskal para grafos densos

Esta implementación sigue el algoritmo de Kruskal pero como veremos, resulta más eficiente para grafos con más cantidad de nodos.

Esencialmente, se manipula el grafo utilizando una matriz de adyacencias, lo cual genera un acceso rápido a la distancia entre dos nodos adyacentes. Particularmente en este caso sabemos que estamos trabajando con un grafo completo y por lo tanto esta estructura tendrá llenas todas sus posiciones.

Por otro lado se mantiene un vector de longitud N con la arista de menor costo para cada nodo; la posición i contiene el menor costo de todas las aristas conectadas al nodo i dado que i es un representante de su componente conexa. Decimos que i es un representante si es raíz del subárbol generado por Kruskal hasta la iteración actual. De esta manera, al agregar una arista lo que estamos haciendo es conectar dos componentes en una, hasta completar el árbol. Ésta optimización reduce el costo de tener que recorrer toda la matriz en busca de la arista más barata, de $O(n^2)$ a $O(2n)$ una para recorrer el vector buscando la arista más barata y otra para mantenerlo, lo cual termina en $O(n)$. Agregamos al AGM la arista de menor costo entre las más cercanas de todos los representantes que no genere un ciclo con los otros. Nótese que para mantener esta estructura la matriz de adyacencias resulta más que útil para mantener el vector en cuestión, ya que accedemos a los costos de cada arista particular en $O(1)$.

3.2. Kruskal para grafos ralos

En segundo lugar, tenemos la implementación de Kruskal para grafos ralos. Esta utiliza una lista de adyacencias en lugar de una matriz para representar los grafos. Esto facilita la obtención de la arista de menor costo en cada iteración, ya que nos permite simplemente ordenar las aristas de menor a mayor e ir tomándolas en orden.

Para agregar una arista al AGM que estamos construyendo, esta debe cumplir dos condiciones; debe ser la de menor costo y no debe generar ciclos con las aristas ya colocadas. Para esto se utiliza la estructura **Disjoint Set Union** o **DSU** lo cual nos ayuda a la hora de verificar si la arista que estamos viendo generará un ciclo o no en el AGM. Esto se logra manteniendo una estructura d padres a medida que vamos conectando los nodos; inicialmente cada nodo es su propio padre, pues pertenece a una componente conexa con él cómo último elemento, pero a medida que los vamos conectando se empiezan a compartir padres, o nodos representantes.

3.2.1. Kruskal para grafos ralos: DSU optimizado

Un DSU con optimización *Union by rank* mantiene dos vectores en cada iteración del algoritmo de Kruskal; uno con los padres o representantes de cada componente conexa que tengamos en esa iteración, y otro con el rango de cada nodo, que es una estimación de la altura del árbol al que pertenece el nodo en la estructura de conjuntos disjuntos (componentes conexas). En este enfoque, al realizar una operación de unión de dos conjuntos, se toma en cuenta el rango de cada uno. El que tenga menor rango se fusiona con el conjunto de mayor rango. Si ambos conjuntos tienen el mismo rango, se elige uno de ellos como el conjunto raíz y se incrementa su rango en uno.

La idea detrás del *Union by rank* es mantener los árboles lo más equilibrados posible para evitar que se vuelvan demasiado altos para poder realizar operaciones de unión y búsqueda de manera más rápida, mejorando la eficiencia del algoritmo, ya que la altura de los árboles influye directamente en la complejidad temporal de estas operaciones. Esto reduce significativamente el tiempo de ejecución del algoritmo de Kruskal, especialmente en grafos grandes. El uso de conjuntos disjuntos permite una fácil verificación de la conectividad entre nodos, lo que es útil para determinar si una arista forma un ciclo o no.

3.2.2. Kruskal para grafos ralos: DSU no optimizado

Una implementación de Kruskal con DSU no optimizado, únicamente mantiene la estructura de padres o representantes y no la de rangos. Mantener una estructura de rangos puede resultar

insignificante si trabajamos con grafos no densos. Sin embargo la principal desventaja de hacerlo es que los conjuntos disjuntos se unen de manera desbalanceada, lo cual genera árboles muy altos que resulta costoso recorrer en cada iteración de Kruskal, particularmente en grafos densos, como es el caso del ejercicio que motiva el presente informe. No utilizar la optimización *Union by rank* tendrá un gran impacto en la complejidad de nuestro programa.

3.3. Experimentacion

El objetivo del experimento es observar la relación entre la cantidad de nodos en cada caso de test y el tiempo de resolución (en milisegundos) del ejercicio con las distintas implementaciones del algoritmo de Kruskal para AGMs, y determinar cuál de estas implementaciones es más eficiente en esta situación.
Para este experimento se generaron cien casos de test, cada uno con una cantidad N de oficinas (es decir, N nodos), con $100 \leq N \leq 1000$.

Asimismo, para cada caso de test se generaron las variables R, W, U y V, respetando: $1 \leq R \leq 10000$, $1 \leq U \leq V \leq 10$ y $1 \leq W < N$. También, para la cantidad N de oficinas en cada caso de test se generaron las variables x e y que corresponden a la posición de cada nodo en un eje cartesiano, con valores entre $-10000 \leq x, y \leq 10000$.
La generación de variables para los cien casos de test se realizó con una distribución uniforme de números random para cada variable y se utilizaron los mismos casos de test para las tres distintas implementaciones del algoritmo de Kruskal.
En el siguiente gráfico se pueden observar tres conjuntos de puntos distintos, donde cada punto corresponde a un caso de test: Los puntos de color azul corresponden a la resolución del ejercicio con una implementación del algoritmo de Kruskal sin optimización de *Union by rank*, los puntos de color rosa corresponden a la resolución con una implementación de Kruskal con optimización de *Union by rank* y los puntos de color rojo corresponden a la resolución con una implementación del algoritmo de Kruskal para *grafos densos*. Las líneas sobre cada conjunto de puntos corresponden a la línea de regresión que ayuda a representar mejor la dispersión de los puntos que pertenecen a cada conjunto.

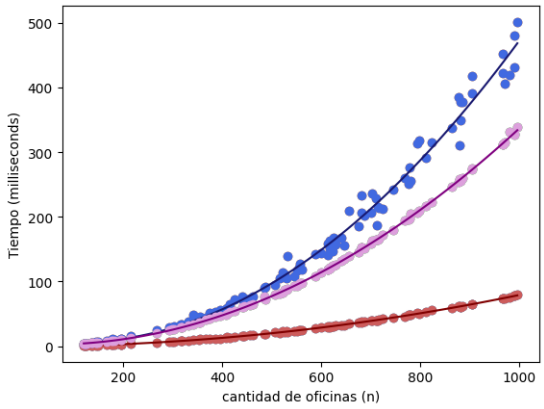

Figura 1: Relación entre la cantidad n de oficinas y el tiempo de resolución del algoritmo de Kruskal con distintas implementaciones

A partir del gráfico se puede observar una gran diferencia entre el tiempo de ejecución del algoritmo de Kruskal para grafos densos y el tiempo de ejecución de los algoritmos de Kruskal implementados con DSU (con y sin optimizar) cuando se trata de casos de test con una gran cantidad de nodos a analizar. Sin embargo, los tiempos de ejecución de todos los algoritmos cuando se trata de casos en los que la cantidad N de nodos no es muy alta ($N \lessapprox 250$) no resultan con diferencias muy relevantes, ya que todos tienen tiempos de ejecución muy parecidos, y los tiempos

solo comienzan a divergir cuando N $\gtrapprox$ 250.
Además, las diferencias entre los tiempos de ejecución del algoritmo implementado con DSU *optimizado* y DSU *sin optimizar* presentan una leve mejora del tiempo cuando el DSU se optimiza con *Union by rank* en los casos con N muy grande.
Consideramos que para este caso de uso la implementación del algoritmo de Kruskal para grafos densos es mejor, ya que es mucho más eficiente en términos del tiempo de ejecución para los casos con mayor cantidad de oficinas en comparación a las otras implementaciones y considerando la cantidad total de oficinas posibles a analizar en el ejercicio ($1 \leq N \leq 1000$).

4. Justificación de correctitud

Como mencionamos en el apartado de la resolución del problema, el modelo elegido para determinar la solución es tomar a cada oficina como un nodo y la distancia entre ellas como el costo de las artistas, así generando un grafo completo.

Por otro lado, el algoritmo de kruskal nos devuelve un Árbol Generador Mínimo del grafo G que le pasemos como parámetro, es decir, un subgrafo de G que conecta todos los nodos de manera acíclica utilizando las aristas de menor peso, o sea las distancias entre las oficinas. El invariante de este algoritmo es el siguiente:

Invariante: Tenemos un bosque generador de i aristas que es subgrafo de algún AGM.

Es decir en la actual iteración el algoritmo ya ha colocado i aristas cuyo costo es lo mínimo que pueden estar dentro de los datos de entrada, formando un grafo disconexo que es solución parcial de nuestro problema. Luego procede a elegir la próxima arista de menor peso que no genere un ciclo en nuestra solución parcial y la agrega. Es decir el cable de menor longitud que conecte dos componentes conexas de oficinas. En este momento es cuando hacemos el presupuesto de los tipos de cables; como sabemos que U es siempre menor o igual a V utilizamos cables de tipo UTP cada vez que podamos, que en las primeras iteraciones será siempre, pues sabemos que las aristas están ordenadas, o en su defecto, sabemos que elegiremos siempre la próxima más barata. Esto nos asegura que utilizamos todas las instancias posibles para abaratar costos. Luego, una vez que alguna sobrepase la cota, se colocaran cables de fibra, ya que no hay otra opción. Como en todo momento se cumple el invariante mencionado previamente, podemos asegurarnos de que el costo de fibra y de UTP será el mínimo que podemos obtener con un solo modem, pues se corresponde con el mínimo costo del AGM.
Una vez terminado este proceso, se hace uso de W, la cantidad de modems. Agregar más decrementará aún más el precio a pagar por los cables (que ya era el mínimo). Al poner un modem se remueve del AGM la arista más pesada y lo más probable es que sea una de tipo fibra, pues las distancias más grandes se agregaron al final para aprovechar la cota de distancias de UTP. Por lo tanto, de los cables de fibra que hayamos puesto o bien de los más largos de UTP se eliminarán W reduciendo aún más los costos. Finalmente, podemos afirmar que el resultado encontrado es el optimo para este problema.