Victoria Klimkowski y Lorenzo Gandolfo

Resumen

Este informe describe la resolución de un problema de optimización de modems y cables entre oficinas donde se quiere encontrar la mínima cantidad de dinero que debe pagarse en cables para conectar todas las oficinas utilizando dos tipos de cables (UTP y fibra óptica). El algoritmo utilizado usa una implementación de Kruskal.

Introducción

En las nuevas *N* oficinas del edificio *0*-∞ se quiere encontrar la forma óptima de conectar *W* modems, con *W*<*N*, utilizando dos tipos de cables: UTP con costo *U* y fibra óptica con costo *V*. Los cables UTP tienen la condición particular de que solo pueden usarse entre dos oficinas si estas están a menos de *R* centímetros. Se quiere encontrar la mínima cantidad de dinero que debe pagarse en cables para conectar las oficinas, es decir, cuánto debe gastarse en UTP y cuánto en fibra óptica.

Para resolver el problema es posible utilizar un algoritmo basado en Kruskal, un famoso algoritmo sobre grafos pesados que sirve para encontrar árboles generadores mínimos, es decir, árboles generadores donde la sumatoria de los pesos de las aristas sea la mínima posible. Este algoritmo está basado en una estrategia greedy donde siempre se seleccionan las aristas de peso mínimo en orden creciente : primero se ordenan las aristas en forma creciente y se crea un árbol, luego se van agregando las aristas mínimas con la condición que la arista agregada no forme un ciclo con lo que se está construyendo. La estrategia de Kruskal se apoya fuertemente en la estructura de datos Disjoint Set Union (DSU), una estructura que permite guardar las componentes conexas de un grafo y permite realizar operaciones sobre ellas de forma eficiente, en específico verificar si una arista forma un ciclo o no.

Resolución

Como las oficinas están ubicadas en un lugar puntual del plano del edificio se debe medir la distancia euclídea entre todas ellas; la función *distancia* devuelve la distancia euclídea entre las oficinas cuyas coordenadas son (x_1,y_1) y (x_2,y_2). Al calcularlas, clasificamos el tipo de cables a utilizar entre esas oficinas: si la distancia es menor o igual a R el peso de esa arista va a ser *U * distancia entre las dos oficinas* (cable UTP,) en cambio, si es mayor a R el peso de la arista será *V*distancia.* Representamos a las aristas con tuplas de la forma $(peso, oficina_i, oficina_j, esUTP)$ donde *oficina* es un natural (un nodo) y *esUTP* es un bool que es verdadero si el cable es UTP y falso si es de fibra.

Para finalizar, corremos *Kruskal* adaptado a la estructura de aristas que creamos y detenemos el algoritmo cuando construye W árboles ya que el problema pide modelar las distancias para W modems (sabemos que siempre va eligiendo las aristas menos pesadas). Guardamos las aristas marcadas por Kruskal en la variable AGM y con la función *separarAristas* lo recorremos y separamos en 2 vectores distintos las aristas UTP y las de fibra. Finalmente, imprimimos la suma de los pesos de ambos vectores por separado.

Justificación del algoritmo

La resolución de este ejercicio se apoya fuertemente en el invariante de Kruskal, el cual garantiza que en cada iteración del algoritmo la solución parcial es correcta. Podemos definir al invariante de kruskal de la siguiente forma: *"Sea G un grafo conexo y T_k un bosque tal que es subgrafo de algún AGM de G, para todo 1≤ k < N. En la iteración k se tiene un bosque mínimo de n-k componentes conexas".*

Del invariante se sigue que no va a haber ciclos (es decir que no se van a malgastar cables conectando oficinas de manera no única) ya que nos asegura que se generará un bosque.

No queremos tener menos de W componentes conexas. Cada componente conexa está conectada a un modem (internet) y los cables son los que permiten que hayan conecciones en varias oficinas (en particular, las de esa componente conexa). Si hay menos de W componente conexas esto significa que o bien hay un modem que no se usó o bien que hay

alguna componente conexa que tiene más de un modem lo cual sería desperdiciar un modem (porque ya están comprados) pero igualmente gastamos dinero en conectar todas esas oficinas innecesariamente.

Tampoco queremos tener más de W componentes conexas porque como cada una está conectada a un modem no tendríamos la cantidad de modems necesaria para cubrir a todas. Esto pasa porque justamente la conectividad entre 2 componentes conexas debería hacerse a través de cables que no estamos poniendo: no tienen forma de tener conectividad.

De esto sale que necesitamos tener W componentes conexas. Por el invariante sabemos que en la iteración k tendremos n-k componentes conexas. Si n-k=W, sale que k=n-W. Es decir que lo conseguiremos en la iteración n-W. Esto vale porque como n>W>0 vale que 0<n-W<n y el invariante de Kruskal nos asegura que será un bosque mínimo de W componentes conexas. Luego no hay ninguna elección de aristas mejor (es decir que minimice el costo) si tenemos a disponibilidad W modems.

Complejidad

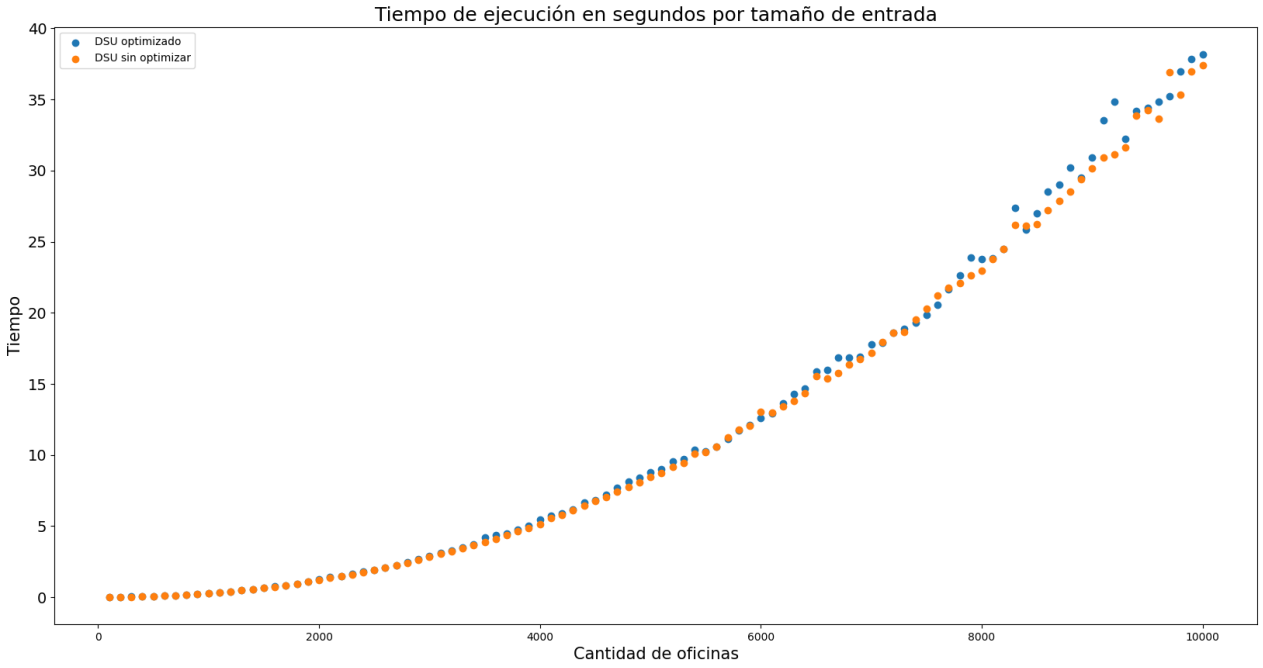
Armar el vector de n oficinas (V nodos) es O(V), calcular las distancias entre todas es O(V^2), Para el algoritmo de Kruskal, la complejidad es O(E log(E)). Esto es así porque primero se ordenan todas las E aristas según el peso y luego al hacer DSU hay una complejidad de O(E*α(V)) con α(V) la inversa de Ackermann. Separar las aristas UTP de las de fibra es O(E) y sumar sus pesos es O(E). Entonces es O(V+ E + V^2 + E*log(E)). Como E≈V^2 la complejidad es O(V^2 * log(V))

Distintas implementaciones de Kruskal

Como ya mencionamos, el algoritmo implementado de Kruskal se apoya fuertemente en la estructura de datos DSU. Ahora, en esta estructura se tienen las funciones *find* y *unite* las cuales permiten encontrar elementos y unir conjuntos distintos. Existe una implementación optimizada de Kruskal la cual modifica las funciones mencionadas a través de técnicas

denominadas *union by rank* y *path compression*. Estas técnicas permiten reducir balancear y reducir las alturas de los árboles que se van generando, garantizando que las operaciones sean menos costosas. Al utilizar las optimizaciones, las operaciones de DSU son casi O(1) amortizadas en lugar de logaritmica.

Realizamos un test con 100 inputs que van de 100 a 10000 oficinas para verificar si existe alguna diferencia entre el algoritmo con la implementación de Kruskal optimizado (union by rank y path compression) vs no optimizado. Para realizar el test utilizamos python en una notebook de jupyter con las bibliotecas numpy, subprocess, pandas y matplotlib.



Conclusiones

Teóricamente, el algoritmo con la implementación de Kruskal optimizado debería tener un tiempo de ejecución menor ya que el costo de las operaciones es logarítmico; sin embargo, el test realizado no mostró diferencias significativas al utilizar una implementación u otra, el tiempo de ejecución resultó ser prácticamente el mismo para cada test. Esto probablemente se deba a que en Kruskal también se realiza un *sort* que le suma una complejidad de a lo sumo O(E log E). Luego, el algoritmo tiene otras funciones tales como *separarAristas* y *distancia* cuyas complejidades no varían por utilizar una u otra implementación. La forma en que lo medimos fue haciendo casos de test con python, corriendo eso con nuestro código y

tomándole el tiempo entero a eso ya que es lo que nos permitía hacer la librería de python utilizada.