Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2133 - Estructuras de Datos y Algoritmos

Informe Tarea 3

7 de julio de 2021

1º semestre 2021 - Profesores C. Gazali - Y. Eterovic

Vicente Espinosa - 17639247

Algoritmos

Algoritmo usado: (Kruskal)

Implementación:

```
kruskal(grafo, cantidad_clientes)
{
    crear lista de resultados;
    qsort(grafo);
    crear subsets;
    for(elementos in grafo)
    {
        if (elemento is cliente)
            parent = elemento;
        else // Sería C. de distribución
            parent = C+1;
    }
    while(caminos encontrados < V-1 && not termino grafo)
    {
        arista = siguiente arista en grafo;
        if (padre(inicio) != padre(final) && no son ambos un centro de distribución)
            Unir(inicio, final);
    }
    costo = suma soluciones;
}
```

Esto es de manera muy básica el código que se implementó para la solución, el único cambio relevante respecto al algoritmo original es el que los nodos que son centros de distribución se les asignó el mismo padre (para que eviten conectarse entre sí), y también se modificó el algoritmo de unión para que si un elemento tiene como padre a C+1 (C. de Dist.) se mantenga este padre para ambos nodos, independiente de cual tiene mayor rango.

Complejidad:

- Primero se cargan todos las aristas en el grafo (H)

- Se ordenan las aristas, con una función que es similar a quicksort, por lo que toma $n(logn)$, lo que sería $H(logH)$

- Se recorren todas las aristas del grafo (H)

- Se recorren los caminos o todo el grafo (H)

Teniendo todo lo anterior en cuenta, la complejidad de ejecución de este código sería algo en la función de $O(H(logH))$

Otra opción de algoritmo: (Prim)

Implementación:

La implementación de este algoritmo era más difícil que el anterior, ya que este algoritmo debe comenzar en un nodo indicado, y luego ir expandiéndose hacia sus vecinos para encontrar el camino más corto. El problema está en que este algoritmo genera un grafo conexo, y la solución a este problema es muy probablemente un grafo no conexo. Sería muy complejo el adaptar este algoritmo para que detecte cuando conviene hacer un nuevo grafo no conexo.

Complejidad:

Este algoritmo tiene como base la complejidad de O(n^2), lo que sería O(H^2) en este caso.

Probablemente la complejidad se mantendría o aumentaría, ya que se le están agregando más restricciones al final.

Por lo tanto, se concluye que el algoritmo de Kruskal es una opción mucho mejor para este caso, tanto por que es más facil adaptar el algoritmo original y agregarle restricciones nuevas, como por que la complejidad del codigo parece ser menor, y por lo tanto, más eficiente.