Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 – Estructuras de datos y algoritmos

Informe 4

Estudiante: Felipe Baitelman

1. Representación del gráfico

Para representar el grafo se utilizaron listas de adyacencia. Esta representación consiste en un arreglo *Adj* de tamaño igual a la cantidad de vértices del grafo, en donde cada entrada del arreglo contiene una lista de nodos. De esta forma, para un nodo u, todos los nodos en la lista *Adj[u]* son nodos adyacentes a u. Cada nodo del arreglo tiene como atributos el número del vértice, su capacidad y una referencia al siguiente nodo de la lista.

En general las dos formas tradicionales de representar grafos es con listas de adyacencia o con una matriz de adyacencia y la elección de una u otra técnica depende de las características del problema. En este caso en particular, los grafos serán poco densos ya que la cantidad de aristas es considerablemente menor que la cantidad de vértices al cuadrado. Por lo tanto, conviene utilizar listas de adyacencia.

2. Análisis del algoritmo seleccionado

El algoritmo seleccionado para resolver el problema fue una variación del Algoritmo de Prim. Básicamente, el algoritmo trabaja de la misma forma que el Algoritmo de Prim pero en vez de buscar el árbol de cobertura de costo mínimo busca el árbol de cobertura de flujo minimizando el costo. Para esto, en vez de utilizar un *Min-Heap* para guardar los vértices aún no seleccionados en el árbol de cobertura, se utiliza un *Max-Heap* tal que se agrega al árbol el nodo con el mayor flujo. Si hay más de una opción con flujos iguales, se selecciona la con menor costo.

Al igual que el Algoritmo de Prim, el algoritmo utilizado es codicioso. Esto se debe a que en cada iteración se agrega al árbol de cobertura una arista liviana que conecta al árbol con un nodo que no estaba no

conectado anteriormente. La arista seleccionada es segura para el árbol, y garantiza que siempre se agrega al árbol la arista que maximiza el flujo total. La restricción de minimizar el costo es secundaria, por lo tanto no afecta el comportamiento mismo del algoritmo.

3. Complejidad del algoritmo seleccionado

La complejidad del algoritmo depende de la forma en como se implementa la cola de prioridades. Ya que la cola se implementa con un *Max-Heap*, la inserción de todos los vértices V en ella toma un tiempo de $\mathcal{O}(V)$.

Por otro lado, el *while* principal del algoritmo que verifica que la cola de prioridades no este vacía se ejecuta $|V|$ veces y cada operación de extracción de la cola toma $\mathcal{O}(\log V)$, por lo que que el tiempo total en llamadas a extraer el mínimo tendrá una complejidad de $\mathcal{O}(V \log V)$.

Más aún, el *for* que recorre las listas de adyacencia para buscar las aristas livianas se ejecuta en total $2|E|$ veces, por lo que su complejidad será $\mathcal{O}(E)$. Ya que al seleccionar la arista liviana se deben actualizar las claves de los elementos en el *Max-Heap*, operación que toma $\mathcal{O}(\log V)$, el tiempo total que toma el *for* tendrá una complejidad de $\mathcal{O}(E \log V)$.

Tomando todo lo anterior en consideración, el tiempo total que toma el algoritmo será $\mathcal{O}(V \log V + E \log V)$ que es equivalente a $\mathcal{O}(E \log V)$.

4. Bibliografía

Informe:

- Cormen, Thomas H., Charles Eric. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. Cambridge, MA: MIT, 1990. Print.

Código:

- Para implementar el algoritmo de Prim y listas de adyacencia se utilizó como código base el código proveniente de la siguiente página: http://www.geeksforgeeks.org/greedy-algorithms-set-5-prims-mst-for-adjacency-list-representation/.
- Para implementar el *Max-Heap* se utilizo como código base el de la Ayudantía 2.