Informe tarea 4

Hugo Maximiliano Castro Leppe
Algoritmos y Estructuras de Datos 2019-2
Universidad de Talca

29 de Marzo de 2020

1 Introducción

En este presente informe se detallará la resolución e implementación de la solución de la tarea 4, ¿Cuál es el mejor camino?. Se explicará la aproximación hacia el enunciado, el algoritmo de búsqueda utilizado, las clases y métodos más importantes y análisis de complejidad correspondientes.

2 Acercamiento al enunciado

El enunciado se puede traducir a un problemas de grafos y de encontrar el camino más corto, si es que existe, entre 2 pares de vértices. Para formar un grafo, pensé en formar una lista de adyacencia, que me permitiera realizar un traversal de manera simple. Para poder saber cual es la distancia más corta entre 2 pares de vértices, decidí usar **"Breadth First Search"** o **BFS** para abreviar, ya que me permite obtener la distancia entre un vértice inicial y todos los vértices presentes en el grafo, además que sirve para grafos sin peso en sus aristas ya que es arbitrario al ser 1 (**Unweighted**) y sin dirección (**Undirected**).

En sí, BFS no obtiene la distancia más corta para el vértice de destino, sino que obtiene obtiene la distancia más corta para todos los vértices del grafo, por lo que el tiempo de ejecución de BFS se realizará, en el mejor de los casos, en tiempo $O(1)$ y en el peor $O(V^2)$, con un tiempo de ejecución promedio de $O(E + V)$, con E como la cantidad de aristas y V como la cantidad de vértices.

Para usar BFS para obtener la distancia más corta, haremos uso del distTo[], que nos dirá la distancia mínima del vértice de inicio hasta el de destino. Para optimizar este algoritmo, el traversal del BFS se detendrá al momento de llegar al destino, ya que de otra forma, al ser un BFS, seguiría analizando el resto de vértices.

3 Implementación del algoritmo

3.1 Clase Nodo

Para poder implementar el algoritmo, primero fue necesario crear una clase Nodo, que pudiera representar los vértices del grafo. Los atributos más importantes de esta clase son los siguientes:

```java
/* Dado que se trabaja con listas, los valores de los vertices se corren -1 al guardarlos,
    por lo que en idNodo se guarda el valor real del nodo al agregarle un +1
*/
int idNodo;

// Atributo que permite saber si este nodo ya fue visitado
boolean marcado = false;

Nodo(int idNodo){
    this.idNodo = idNodo+1;
}
// Lista que contiene los nodos adyacentes a este nodo
ArrayList<Integer> nodosAdyacentes = new ArrayList<>();
```

3.2 Método que forma la lista de adyacencia

Ya que tenemos los vértices representados en la clase Nodo, necesitamos hacer una lista de adyacencia, que permita caracterizar el grafo con una lista, la que será recorrida posteriormente por el BFS. Este método realiza esta función:

```java
//  Se crea una lista de adyacencia de size N intersecciones
Nodo[] grafo = new Nodo[vertices];

/*  Se inicializa creando nodos con sus ID correspondientes. Por ejemplo, el vertice 1
    estaria ubicado en la posicion 0 de la lista, pero el objeto Nodo tendria como ID el 1.
*/
for (int i = 0; i < vertices; i++) {
    grafo[i] = new Nodo(i);
}

for (int[] interseccion : ciudad) {
    // Se leen ambas intersecciones
    int vA = interseccion[0];
    int vB = interseccion[1];

    // Se guardan las intersecciones en el nodo de la lista correspondiente
    grafo[vA-1].addNodoAdyacente(vB);
    grafo[vB-1].addNodoAdyacente(vA);
}
return grafo;
```

3.3 BFS para encontrar el camino más corto

Este es el método más importante, porque es el que realiza la tarea de hacer el traversal del grafo, que al final nos permitirá saber la distancia mínima entre el vértice de origen y el de destino.

Primero tenemos el inicio del método donde se crea e inicializa el distTo[] y el Queue para el BFS:

```
// Se crea un array de numeros que contendra la distancia del vertice inicial a todos los
    vertices del grafo
int[] distTo = new int[grafo.length];

// Queue que tendra los vertices pendientes de traversal
Queue<Integer> queue = new LinkedList<>();

// Suponemos que los vertices no tienen conexion con otros vertices, por lo que su distancia
     preliminar es -1
for (int i = 0; i < grafo.length; i++) {
    distTo[i] = -1;
}
```

Luego, el funcionamiento de BFS es el siguiente:

1. Se añade el vértice inicial a la cola
2. Se realiza un ciclo mientras la cola no esté vacía
    - Se remueve un vértice de la cola y se comprueba si sus vértices adyacentes no están marcados
    - Si un vértice adyacente está marcado, se ignora. Si no lo está, se marca, se calcula la distancia (distTo[ ]) y se agrega a la cola.
    - Si el nodo recién calculado es el nodo final, se termina el ciclo anticipadamente, terminando el traversal del BFS.
3. Una vez revisada la cola, o que se haya encontrado el destino, se imprime la distancia. En caso de que se haya revisado el grafo completo y el valor del vértice objetivo siga siendo -1, se imprime "SIN CAMINO"