Laboratorio Nro. 3: Vuelta atrás (Backtracking)

Alejandro Cano Munera
Universidad Eafit
Medellín, Colombia
acanom@eafit.edu.co

Sebastián Giraldo Gómez
Universidad Eafit
Medellín, Colombia
sgiraldog@eafit.edu.co

1.8. El algoritmo recibe un grafo lleno de nodos y arcos, un nodo de inicio y un nodo de fin; el algoritmo inicia con el nodo de inicio y busca el nodo con el cual consigue el camino más corto, y hace esta operación repetidas veces hasta llegar a el nodo de fin, mientras va buscando el camino el método va acumulando los valores en la variable way y va guardando los nodos visitados en un ArrayList, una vez acabe la ejecución imprime el peso del camino y los nodos que se deben visitar para llegar a el otro nodo.

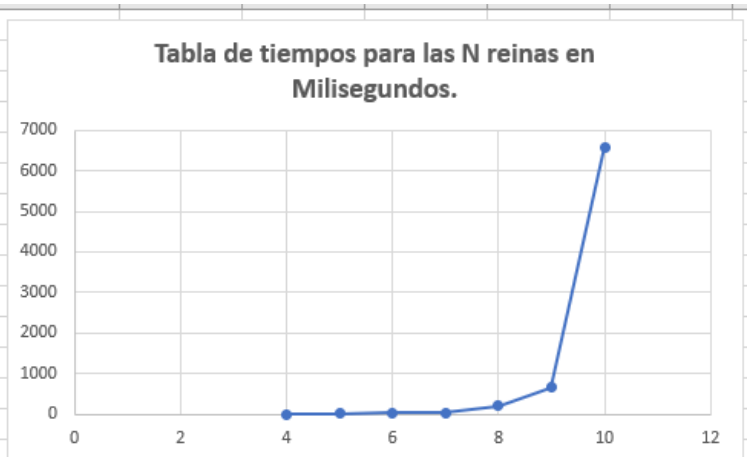
3) Simulacro de preguntas de sustentación de Proyectos:

3.1. Además de fuerza bruta y Backtracking para resolver el problema del camino más corto en un grafo también existe la técnica de los algoritmos voraces, estos consisten en tomar una decisión basado en heurística a la hora de elegir el nodo mas “prometedor” y seguir utilizando esta heurística hasta llegar al objetivo.

3.2.

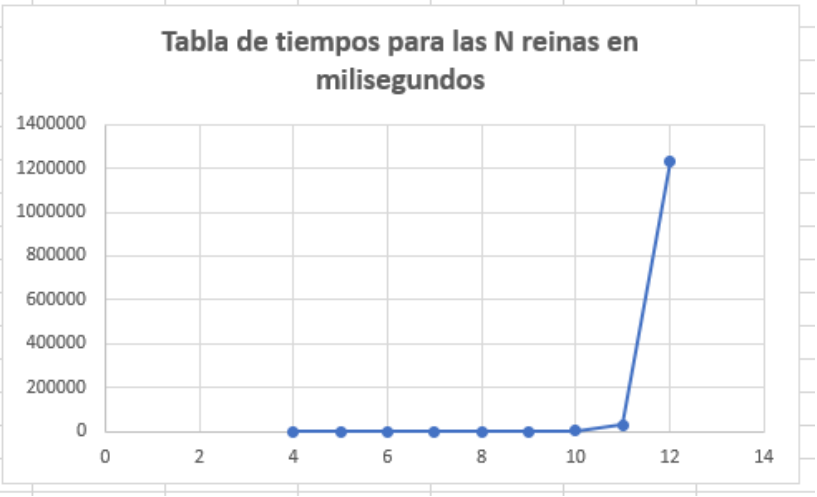
Con fuerza bruta:

N	Milisegundos
4	2
5	6
6	23
7	39
8	195
9	648
10	6581
>11	Más de 30 min



Con Backtracking:

N	Milisegundos
4	0
5	0
6	0
7	0
8	31
9	69
10	2305
11	30522
12	1234904
>13	Más de 20 min



3.3. Utilizar BFS y DFS dependen de los objetivos que queramos alcanzar o el problema que se desea solucionar, también del conocimiento que tengamos de la forma del grafo. Por ejemplo, si conocemos que la solución del problema se encuentra cerca del nodo inicial, es mas conveniente usar BFS, sin embargo, si conocemos que la solución se encuentra muy lejos del nodo inicial es más recomendable usar DFS.

3.4. Los principales algoritmos de búsqueda en grafos son DFS, BFS, Blockut y Dijkstra. DFS o búsqueda en profundidad consiste básicamente en visitar los nodos más profundos, es decir, se elige un nodo inicial, un vecino de este y se explora toda la rama del vecino antes de visitar otro vecino. BFS o búsqueda en amplitud, consiste en visitar todos los vecinos del nodo inicial, y realizar esta acción recursivamente con cada vecino hasta visitar todos los nodos. Blockut, es una adaptación del algoritmo DFS con una ligera variación, donde se puede partir el grafo en bloques de búsqueda, haciéndolo más eficiente. Por último, el algoritmo de Dijkstra es un algoritmo de búsqueda que soluciona el problema del camino mínimo entre dos nodos.

3.5. Estructura de datos: Para resolver este problema se utilizaron ArrayList, con el fin de ir agregando el nodo que se va a visitar teniendo en cuenta la condición de buscar el camino más cercano, es decir, con menos peso entre sus nodos. Al final una vez agregado al ArrayList todo el recorrido que se debe hacer entre los dos nodos para obtener el menor peso, se imprime-

3.6. O(n) = m*g + T(n-1)*h

3.7. Las variables **:**

M: Es el número de líneas que se leen por consola.
G: Es el número de caracteres que contiene la línea leída por consola.
N: Es el número de nodos visitados del grafo.
H: Es el número de sucesores de cada nodo visitado en el grafo.

4) Simulacro de Parcial:

1. **a)** solucionar(14, 3, 2, 7)
   b) res, solucionar(n-b, a,b,c)+1
   c) res, solucionar(n-c, a,b,c)+1

2. **a)** graph.length-1
   b) v, graph, path, pos
   c) graph, path, pos+1

3. **a) DFS:**
   0. 0 3 7 4 2 1 5 6
   1. 1 0 3 7 2 4 6 5
   2. 2 1 0 3 7 5 4 6
   3. 3 7
   4. 4 2 1 0 3 7 5 6
   5. 5
   6. 6 2 1 0 3 7 5 4
   7. 7

   b) BFS:
   0. 0 3 4 7 2 1 6 5
   1. 1 0 2 5 3 4 6 7
   2. 2 1 4 6 0 5 3 7
   3. 3 7
   4. 4 2 1 6 0 5 3 7
   5. 5
   6. 6 2 1 4 0 5 3 7
   7. 7

4. **Métodos:**

```
public static LinkedList<Integer> unCamino(Digraph g, int p, int q) {
```

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

```
        boolean[] visitados = new boolean[g.size()];
        LinkedList<Integer> camino = new LinkedList<>();
        camino.add(p);
        if (unCaminoAux(g, p, q, visitados, camino)) {
            return camino;
        }
        return null;

    }

    private static boolean unCaminoAux(Digraph g, int vert, int targ, boolean[]
visitados, LinkedList<Integer> camino) {
        ArrayList<Integer> sucesores = g.getSuccessors(vert);
        visitados[vert] = true;
        if (sucesores != null) {
            for (Integer sucesor : sucesores) {
                camino.addLast(sucesor);
                if (!visitados[sucesor] && (sucesor == targ || unCaminoAux(g,
sucesor, targ, visitados, camino))) {
                    return true;
                } else {
                    camino.removeLast();
                }
            }
        }
        return false;
    }
```

5. **a)** 1
b) ni, nj
c) T(n) = 2*T(n-1) + C

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co