Laboratorio Nro. 3: Backtracking

Ricardo Rafael Azopardo Cadenas
Universidad Eafit
Medellín, Colombia
rrazopardc@eafit.edu.co

Jhon Jairo Chavarria Gaviria
Universidad Eafit
Medellín, Colombia
jjchavarrg@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1.6 Escriban una explicación entre 3 y 6 líneas de texto del código del ejercicio en línea del numeral 1.6 Digan cómo funciona, cómo está implementado y destaquen las estructuras de datos y algoritmos usados.

El código para encontrar el camino más corto entre dos vértices funciona de la siguiente manera: se llena un arreglo de distancias hasta el infinito y se marca el nodo inicial con 0, este arreglo representa las distancias de todos los nodos al nodo inicial, luego el algoritmo recorre desde el nodo inicial utilizando DFS y comparando las sumas de distancias, marcando en el arreglo de distancias, distancias mejoradas y comparando con las nuevas que vaya encontrando, sigue de tal forma hasta que llegue al nodo final y seleccione la suma de distancias menor.

3.1 Para resolver el problema del camino más corto en un grafo, fuera de fuerza bruta y backtracking, ¿qué otras técnicas computacionales existen?

Otra técnica computacional es el uso de algoritmos voraces, el cuál en cada nodo evalúa cuál es el mejor camino a tomar, esperando hallar una solución global para el camino más corto, no necesariamente espera hallar la solución más correcta del camino más corto, pero si una muy aproximada en un tiempo mucho más eficiente que fuerza bruta y backtracking.

3.2 Teniendo en cuenta lo anterior, tomen los tiempos de ejecución del programa realizado en el numeral 1.1 y en el laboratorio anterior con la solución de fuerza bruta de las n reinas. Completen la siguiente tabla.

Valor de N	Fuerza Bruta	Backtracking
4	1	0
8	49	1
16	Se demora más de 5 min.	13
32	Se demora más de 5 min.	37630
N	O(n!)	O(2^n)

3.3 En los ejercicios a lo largo del curso se han usado ambos para distintas aplicaciones, DFS se ha utilizado para encontrar caminos y ciclos entre vértices, como también se ha demostrado que tiene diversasaplicaciones con backtracking como encontrar el camino más corto entre dos vértices o encontrar la salida de un laberinto. Por otro lado, BFS puede ser seleccionado cuando se requiere encontrar soluciones que estén cerca de un nodo inicial, cuando en este caso el dfs puede irse a profundidades innecesarias, el BFS se ha utilizado para comprobar si un grafo es bipartito o no.

3.4 ¿Qué otros algoritmos de búsqueda existen para grafos? Basta con explicarlos, no hay que escribir los algoritmos ni programarlos.

Algoritmos voraces de búsqueda, los cuales bajo un criterio tienden a tomar la mejor decisión en cada paso para encontrar la mejor solución global, a diferencia del DFS y BFS al actuar bajo un criterio sirve como una herramienta importante para evitar búsquedas innecesarias y encontrar soluciones mucho más eficientes.

3.5 Expliquen con sus propias palabras la estructura de datos que utiliza para resolver el problema del numeral 2.1 y 2.2 [Ejercicio Opcional] y digan cómo funciona el programa.

El ejercicio en línea requiere encontrar el camino más corto desde el vértice 1 de un grafo no dirigido hasta un vértice N, la solución se desarrolló de la siguiente manera: se lee el número de vértices y aristas de un grafo, y luego las conexiones con pesos de los siguientes vértices, luego sobre este grafo construido se utiliza el algoritmo de caminoMasCorto desarrollado previamente en el laboratorio, el cuál encuentra una solución sin usar dijkstra y utiliza backtracking como se especificó en el ejercicio.

3.6 Calculen la complejidad de los ejercicios en línea del numeral 2.1 y 2.2 [Ejercicio Opcional] y agréguenla al informe PDF.

Al ser una solución utilizando DFS el algoritmo tiene una complejidad de O(V + E).

3.7 Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del numeral 3.6.

V simboliza el número de vértices o nodos que tiene un grafo, mientras que E simboliza el número de aristas o arcos.

4) Simulacro de Parcial

1.a) (n - a, a, b, c)
1.b) (res, solucionar(n - b, a, b, c) + 1);
1.c) (res, solucionar(n -c, a, b, c) + 1);
2. a) graph[path[pos]][path[0]]
2. b) (v,graph,path,pos)
2. c) (graph,path,pos+1)
3. a) 0 -> 3 -> 7 -> 4 -> 2 -> 1 -> 5 -> 6
1 -> 0 -> 3 -> 7 -> 4 -> 2 -> 6 -> 5
2 -> 1 -> 0 -> 3 -> 7 -> 4 -> 5 -> 6
3 -> 7
4 -> 2 -> 1 -> 0 -> 3 -> 7 -> 5-> 6
5
6 -> 2 -> 0 -> 1-> 3 -> 7 -> 4 -> 5
7
3. b) 0 -> 3 -> 4 -> 7 -> 2 -> 1 -> 6 -> 5
1 -> 5 -> 0 -> 3 -> 4 -> 7 -> 2 -> 6
2 -> 1 -> 6 -> 4 -> 0 -> 5 -> 3 -> 7
3 -> 7
4 -> 2 -> 1 -> 6 -> 0 -> 5 -> 3 -> 7
5
6 -> 2 -> 4 -> 1 -> 0 -> 5 -> 3-> 7
7

```
4.  public static boolean caminoMasCorto (Graph gd, int ini, int fin,
      LinkedList<Integer> camino, int [ ] distancias){
       if (ini == fin) return true;
      ArrayList<Integer> hijos = gd.getSuccessors(ini);
       if (hijos == null) return false;
      for (int i = 0; i < hijos.size(); i++) {
          if (gd.getWeight(ini, hijos.get(i))+ distancias[ini] < distancias[hijos.get(i)]){
             distancias[hijos.get(i)] = gd.getWeight(ini, hijos.get(i)) + distancias[ini];
```

```
            camino.get(hijos.get(i)) = ini;
            caminoMasCorto(gd, hijos.get(i),fin,camino,distancias);
        }
    }
    return false;
}

public static LinkedList<Integer> unCamino (Graph gd, int p, int q){
    LinkedList<Integer> camino = new LinkedList<Integer>(gd.size);
    int [] distancias = new int [gd.size()];
    for (int i = 0; i<distancias.length;i++){
        distancias[i] = Integer.MAX_VALUE;
    }
    distancias[ini] = 0;
    caminoMasCorto(gd, ini, fin, camino, distancias);
    LinkedList<Integer> visitados = new LinkedList<Integer>(gd.size());
    while (fin != ini){
        int num = camino.get(fin);
        visitados.add(fin);
        fin = num;
    }
    visitados.add(ini);
    Collections.reverse(visitados);
    return visitados;
}
```

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