ESTRUCTURA DE DATOS 2
Código ST0247

Laboratory practice No. 3: Backtracking to pathfinding

Juan David Echeverri Villada
Universidad Eafit
Medellín, Colombia
jdecheverv@eafit.edu.co

Juan Sebastián Guerra Hernández
Universidad Eafit
Medellín, Colombia
jsguerrah@eafit.edu.co

3) Practice for final project defense presentation

3.1 Besides BFS and DFS, there are a couple of different computational techniques that allows us to find the shortest path between two points. Some of them are the Bidirectional Search, the Dijkstra´s algorithm and the Bellman-Ford Algorithm. These algorithms have a hard time depending on the graph where they are executed. For example, graphs with negative weights could cause Dijkstra and bidirectional search to not end, so its important to effectively recognize the problem to solve and the structure of the graph to decide which algorithm use.

3.2 The number of possible paths in a complete graph is given by the equation $\sum_{i=1}^{i=n} \frac{n!}{(n-i)!}$, where n is the number of vertexes in the graph.

3.3 Time Tables (seconds)

N-Reinas	Backtracking	Brute Force
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0,001
7	0	0,00302
8	0,00199	0,01596
9	0,001	0,07979
10	0,003	0,36199
11	0,00199	2,096880
12	0,00499	12,69858
13	0,00199	79,75614
14	0,04189	570,29852
15	0,03491	4185,25927
16	0,28025	+50 minutos

N-Reinas	Backtracking	Brute Force
17	0,15957	+50 minutos
18	1,36634	+50 minutos
19	0,09275	+50 minutos
20	6,87488	+50 minutos
21	0,03812	+50 minutos
22	69,71462	+50 minutos
23	1,0871	+50 minutos
24	18,4144	+50 minutos
25	2,51536	+50 minutos
26	22,46614	+50 minutos
27	51,47666	+50 minutos
28	227,94687	+50 minutos
29	111,63037	+50 minutos
30	4502,21946	+50 minutos
31	+50 minutos	+50 minutos
32	+50 minutos	+50 minutos

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

3.4 To decide whether to use BFS or DFS you should look at the problem you want to solve. BFS is usually used for finding the shortest path between two nodes, testing if a graph is bipartite, finding all connected components in a graph, and others. DFS, in other hand, is usually used for topological sorting, solving problems that require graph backtracking, detecting cycles in a graph, finding paths between two nodes. Is necessary to say that BFS usually requires way more memory than DFS, while DFS is not as good as BFS finding the shortest path.

3.5 Since the problem does not requires a complete, the structure we used to solve it was an adjacency list graph, which is the best way to represent graphs with these conditions. Now, the way we find the best path is using backtracking, so we start in the source node, looking for its successors, and then we start tracking each possible route to achieve the destination node. We have a list where we save each visited node, thus avoiding the repetition of nodes and cycles. We also made sure of taking the best possible path between two nodes (when there is more than one edge between the same two nodes). Finally, when we complete the path (from source to destination) we sum the weights of the node in the path and see if we find the shortest one comparing it with the previous one. Finally, we return a pair of the best distance withs its respective path.

3.6 O(v + e) due that it works similar to the DFS algorithm.

3.7 in the previous exercise, ‘v’ means the number of vertexes in the graph, while ‘e’ means the number of edges.

3.8 Firstly, we read the txt to create the graph according to what is specified in it. Then, we start looking for the best path using DFS, but with the restriction that we cannot go through one vertex more than once. Finally, we sum all the weights of the edges in each path, comparing them and then returning the pair of the lowest distance and its respective path.

4) Practice for midterms

4.1.1 n-a, a, b, c
4.1.2 res, solucionar(n-b, a, b, c)+1
4.1.3 res, solucionar(n-c, a, b, c)+1
4.2.1 path.length
4.2.2 v, graph, path, pos
4.2.3 graph, path, pos+1
4.3.1

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 -- 1 -- 0 -- 3 -- 7 -- 4 -- 5
7

4.3.2

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473

0 -- 3 -- 4 -- 7 -- 2 -- 1 -- 6 -- 5
1 -- 0 -- 2 -- 5 -- 3 -- 4 -- 7 -- 6
2 -- 1 -- 4 -- 6 -- 0 -- 5 -- 3 -- 7
3 -- 7
4 -- 2 -- 1 -- 6 -- 0 -- 5 -- 3 -- 7
5
6 -- 2 -- 4 -- 1 -- 0 -- 5 -- 3 -- 7
7

4.4

```
def path(self, sourceNode, destinationNode, path = []):
    if sourceNode == destinationNode:
      return path.copy()

    successors = self.getSuccessors(sourceNode)

    for node in successors:
      if node in path:
        continue

      path.append(node)
      finalPath = self.path(node, destinationNode, path = path)
      path.remove(node)


    return finalPath
```

4.5.1 1
4.5.2 ni, nj
4.5.3 T(n) = 2T(n-1) + c

4.6.1 Opcion C
4.6.2 Opcion A

4.7.1 r == 0
4.7.2 i
4.7.3 r - 1

PhD. Mauricio Toro Bermúdez
Professor | School of Engineering | Informatics and Systems
Email: mtorobe@eafit.edu.co | Office: Building 19 – 627
Phone: (+57) (4) 261 95 00 Ext. 9473