Department of
Computer Science and Engineering

Title: Implement Depth-First Search

Artificial Intelligence Lab
CSE 404

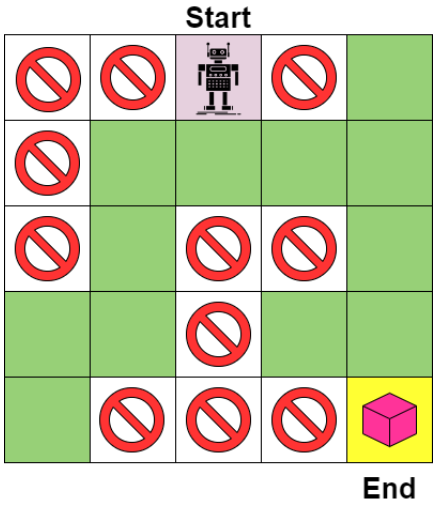
Green University of Bangladesh

1 Objective(s)

- To understand how to represent states and nodes in a graph.
- To understand how Depth-First Search (DFS) works on a two-dimensional (2D) plane.

2 Problem analysis

Two of the most popular tree traversal algorithms are breadth-first search (BFS) and depth-first search (DFS). Both methods visit all vertices and edges of a graph; however, they are different in the way in which they perform the traversal. This difference determines which of the two algorithms is better suited for a specific purpose. As represented in the BFS lab here we also consider a similar scenario where a robot is placed on a two-dimensional plane and there is a goal state that it needs to reach. Fig. (1) represents the 2D graph and the matrix formation of the graph containing obstacle tiles and safe tiles. In Fig. (2) the possible moves are represented considering the starting state is (0, 2) cell and four directional moves - up, down, left, right.



(a) Robot placed on a 2D plane containing obstacles and Goal

R \ C	0	1	2	3	4
0	0	0	S	0	1
1	0	1	1	1	1
2	0	1	0	0	1
3	1	1	0	1	1
4	1	0	0	0	G

(b) Matrix view of the 2D plane

Figure 1: A sample Two-dimensional plane

3 Depth First Search

Depth-first Search or Depth-first traversal is a recursive algorithm for searching all the vertices or nodes of a graph or tree data structure. Traversal means visiting all the nodes of a graph. In Fig. (3 shows a sample traditional undirected graph. If we apply the DFS algorithm on the graph, starting from the S node the traversal sequence will be - S to A to D to G to E to B first. Now from node B, there is an edge to S, but it is not possible to visit S as it was already visited. Now B will return to its parent E and E will return to G. From G

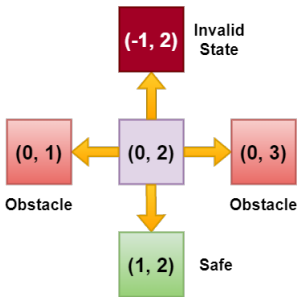


Figure 2: Possible move of the robot

it has another child F, so traversal happened on F, and after that F to C. As all the nodes are traversed the whole process is complete.

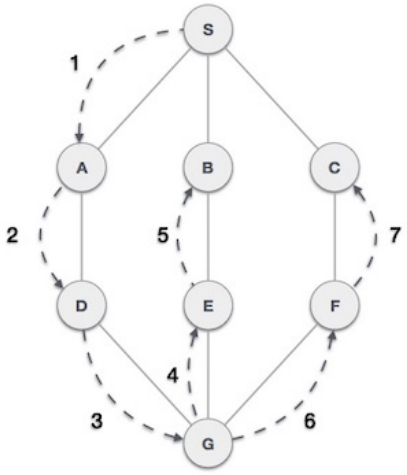


Figure 3: A simple graph

4 Algorithm (DFS)

The DFS algorithm for traditional graphs is given in the following Algorithm.

Algorithm 1: Depth-First Search

```
   Data: graph[N][N], color[N], prev[N], d[N], f[N], time
 1 Function DFS():
 2     for each vertex u ∈ V − {s} do
 3         color[u] = WHITE
 4         prev[u] = null
 5         f[u] = inf
 6         d[u] = inf
 7     end
 8     time = 0
 9     for each u ∈ V do
10         if color[u] == WHITE then
11             DFS_Visit(u)
12         end
13     end
14 return
15 Function DFS_ Visit(u):
16     color[u] = GRAY
17     time = time + 1
18     d[u] = time
19     for each v ∈ adj[u] do
20         if color[v] == WHITE then
21             prev[v] = u
22             DFS_Visit(v)
23         end
24     end
25     color[u] = BLACK
26     time = time + 1
27     f[u] = time
28 return
```

Similarly this idea can be applied to the 2D plane. Here the robot will be traversing the safe states considering

all possible moves. When the robot reaches a particular safe tile or state it will calculate the probable safe move and immediately go to that tile and recursively continue the similar process until it reaches the goal. The modified DFS algorithm for the traversal of the robot is given in the Algorithm 2.

Algorithm 2: Depth-First Search on 2D Grid

Data: graph[N][N], depth[N], x_move[1, -1, 0, 0], y_move[0, 0, 1, -1]

Function *DFS()*:
  Represent each Cartesian point as node
  Initialize Start State S
  Initialize Goal State G
  $depth[S] = 0$
  $DFS\_Visit(S)$
  if $goal\_flag == true$ **then**
    Print(Goal found);
  else
    Print(Goal can not be reached);
  end
return
Function *DFS_ Visit(node u)*:
  $graph[u\_x][v\_y] = visited$;
  for *each child node* $v \in adj[u]$ **do**
    if (v_x *and* v_y *are valid considering* x_move *and* y_move) & $graph[v\_x][v\_y] == free$ **then**
      if $v == G$ **then**
        $goal\_flag = true$;
        break;
      end
      $level[v] = level[u] + 1$
      $DFS\_Visit(v)$
    end
  end
return

5 Implementation in Java

```
package dfs_2d;
import java.util.LinkedList;
import java.util.Queue;

/**
 *
 * @author Jargis
 */
class node {

    int x, y;
    int depth;

    node(int a, int b, int z) {
        x = a;
        y = b;
        depth = z;
    }
}

class DFS {

```

```
23      int directions = 4;
24      int x_move[] = {1, -1, 0, 0};
25      int y_move[] = {0, 0, 1, -1};
26      int N;
27      boolean found = false;
28      int goal_level;
29      int state;
30      node source, goal;
31
32      DFS() {
33          Init();
34      }
35
36      void Init() {
37          int graph[][] = {
38              {0, 0, 1, 0, 1},
39              {0, 1, 1, 1, 1},
40              {0, 1, 0, 0, 1},
41              {1, 1, 0, 1, 1},
42              {1, 0, 0, 0, 1}
43          };
44          N = graph.length;
45
46          int source_x = 0;       //source state
47          int source_y = 2;
48          int goal_x = 4;         //goal state
49          int goal_y = 4;
50          source = new node(source_x, source_y, 0);         //init source
51          goal = new node(goal_x, goal_y, 999999);      //init goal
52          StDFS(graph, source);
53
54          if (found) {
55              System.out.println("Goal found");
56              System.out.println("Number of moves required = " + goal.depth);
57          } else {
58              System.out.println("Goal can not be reached from starting block");
59          }
60      }
61
62      void printDirection(int m, int x, int y) {
63          if (m == 0) {
64              System.out.println("Moving Down (" + x + ", " + y + ")");
65          } else if (m == 1) {
66              System.out.println("Moving Up (" + x + ", " + y + ")");
67          } else if (m == 2) {
68              System.out.println("Moving Right (" + x + ", " + y + ")");
69          } else {
70              System.out.println("Moving Left (" + x + ", " + y + ")");
71          }
72      }
73
74      void StDFS(int[][] graph, node u) {
75
76          graph[u.x][u.y] = 0;
77          for (int j = 0; j < directions; j++) {          //calculating up, down,
                left and right directions
78              int v_x = u.x + x_move[j];
```

```
            int v_y = u.y + y_move[j];                    //check the boundary
                conditions
            if ((v_x < N && v_x >= 0) && (v_y < N && v_y >= 0) && graph[v_x][v_y
                ] == 1) {
                int v_depth = u.depth + 1;
                printDirection(j, v_x, v_y);
                if (v_x == goal.x && v_y == goal.y) //goal check
                {
                    found = true;
                    goal.depth = v_depth;
                    return;
                }

                node child = new node(v_x, v_y, v_depth);
                StDFS(graph, child);
            }
            if (found) {
                return;
            }
        }
    }
}

public class DFS_2D {

    public static void main(String[] args) {
        // TODO code application logic here
        DFS d = new DFS();
    }
}
```

6 Sample Input/Output (Compilation, Debugging & Testing)

Output:
Moving Down (1, 2)
Moving Right (1, 3)
Moving Right (1, 4)
Moving Down (2, 4)
Moving Down (3, 4)
Moving Down (4, 4)
Goal found
Number of moves required = 6

7 Implementation in Python

```
class Node:
    def __init__(self, a, b, z):
        self.x = a
        self.y = b
        self.depth = z

class DFS:
    def __init__(self):
        self.directions = 4
        self.x_move = [1, -1, 0, 0]
```

```
        self.y_move = [0, 0, 1, -1]
        self.found = False
        self.N = 0
        self.source = None
        self.goal = None
        self.goal_level = 999999
        self.state = 0

    def init(self):
        graph = [
            [0, 0, 1, 0, 1],
            [0, 1, 1, 1, 1],
            [0, 1, 0, 0, 1],
            [1, 1, 0, 1, 1],
            [1, 0, 0, 0, 1]
        ]
        self.N = len(graph)

        source_x = 0
        source_y = 2
        goal_x = 4
        goal_y = 4
        self.source = Node(source_x, source_y, 0)
        self.goal = Node(goal_x, goal_y, self.goal_level)
        self.st_dfs(graph, self.source)

        if self.found:
            print("Goal found")
            print("Number of moves required =", self.goal.depth)
        else:
            print("Goal cannot be reached from the starting block")

    def print_direction(self, m, x, y):
        if m == 0:
            print("Moving Down ({}, {})".format(x, y))
        elif m == 1:
            print("Moving Up ({}, {})".format(x, y))
        elif m == 2:
            print("Moving Right ({}, {})".format(x, y))
        else:
            print("Moving Left ({}, {})".format(x, y))

    def st_dfs(self, graph, u):
        graph[u.x][u.y] = 0
        for j in range(self.directions):
            v_x = u.x + self.x_move[j]
            v_y = u.y + self.y_move[j]
            if (0 <= v_x < self.N) and (0 <= v_y < self.N) and graph[v_x][v_y]
               == 1:
                v_depth = u.depth + 1
                self.print_direction(j, v_x, v_y)
                if v_x == self.goal.x and v_y == self.goal.y:
                    self.found = True
                    self.goal.depth = v_depth
                    return

                child = Node(v_x, v_y, v_depth)
                self.st_dfs(graph, child)
```

```
68              if self.found:
69                  return
70
71  def main():
72      d = DFS()
73      d.init()
74
75  if __name__ == "__main__":
76      main()
```

8 Sample Input/Output (Compilation, Debugging & Testing)

Output:
Moving Down (1, 2)
Moving Right (1, 3)
Moving Right (1, 4)
Moving Down (2, 4)
Moving Down (3, 4)
Moving Down (4, 4)
Goal found
Number of moves required = 6

9 Lab Task (Please implement yourself and show the output to the instructor)

1. Write a program to perform DFS traversal on a 2D plane by taking the user input of grid size N. After that generate N×N matrix and place the obstacles randomly. Now print the matrix and take user input of starting and goal state.
2. Write a program to find the path from source to destination using DFS.

10 Lab Exercise (Submit as a report)

- Write a program to find the topological order of node traversal of the robot on the 2D graph plane.

11 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.