Department of
Computer Science and Engineering

Title: Implement Breadth-First Search Traversal

Artificial Intelligence Lab
CSE 404

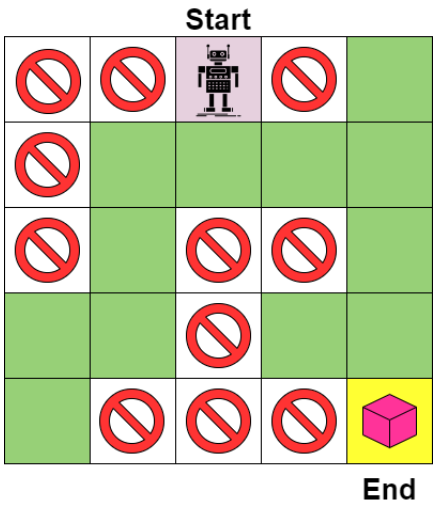
Green University of Bangladesh

1 Objective(s)

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

2 Problem analysis

Every graph is a set of points referred to as vertices or nodes connected using lines called edges. The vertices represent entities in a graph. Edges, on the other hand, express relationships between entities. Hence, while nodes model entities, edges model relationships in a network graph. A graph G with a set of V vertices and E edges is represented as G= (V, E). In the Cartesian two-dimensional plane, we can not represent graphs as nodes and edges. For example, in Fig. (1a), a robot is placed on a 2D plane and there are some obstacles on the plane. Also, the safe blocks are marked as green tiles. Now if the goal of the robot is to reach the end tile and pick the box placed on that tile, then it needs to traverse the 2D plane. We can represent the graph of the 2D plane as given in Fig. (1b), where obstacle tiles are marked as 0 cells and safe tiles are marked as 1 cells.



(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

In this scenario we will consider nodes are the states where the robot is currently standing. Now let's suppose the starting state of the robot is $(0, 2)$ cell, and it can move only up, down, right, and left. In Fig. (2) the possible moves are presented considering the start state. In the given scenario in Fig. (1a), the only valid move will be going to (1, 2) cell which is a new state and will be considered as a new node.

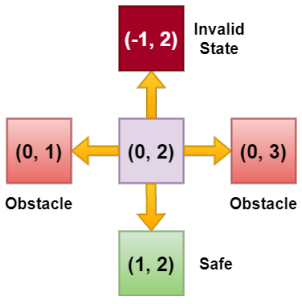


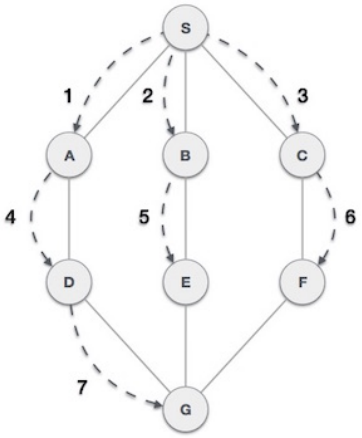
Figure 2: Possible move of the robot

3 Bread First Search

To solve the above problem where the robot can go to the Goal state from the starting state, we can apply the modified version of the Breadth First Search (BFS) algorithm. As BFS traverses a graph in a breadth ward motion and uses a queue to remember to get the next vertex to start search when a dead end occurs in any iteration. Here the idea will be traversing the safe states considering all possible moves of the robot and storing the newly possible move state as a node in the queue.

4 Algorithm of BFS

Let's first consider the conventional BFS algorithm and its traversal.



As in the figure given above, from the source, S BFS algorithm traverses its child A, B, and C first and will put in a queue. In the next phases child of A, B, and C nodes which are D, E, and F respectively will be visited and inserted in the queue. Finally, the G node can be visited from node D. The algorithm of conventional BFS is given in Algorithm 1.

Algorithm 1: Breadth-First Search

```
   Data: graph[N][N], visited[N], level[N], Queue Q
 1 for each vertex u ∈ V − {s} do
 2 |   visited[u] = WHITE
 3 |   level[u] = inf
 4 end
 5 visited[s] = GRAY
 6 level[s] = 0
 7 Q = empty
 8 ENQUEUE(Q, s)
 9 while Q not empty do
10 |   u = DEQUEUE(Q)
11 |   for each v ∈ adj[u] do
12 |   |   if visited[v] == WHITE then
13 |   |   |   visited[v] = GREY
14 |   |   |   level[v] = level[u] + 1
15 |   |   |   ENQUEUE(Q, v)
16 |   |   end
17 |   end
18 |   visited[u] = BLACK
19 end
```

Now the modified BFS algorithm for the given problem where robot is moving on a 2D plane is given in Algorithm 2.

Algorithm 2: Breadth-First Search on 2D Grid

Data: $graph[N][N]$, $level[N]$, $x\_move[1,-1,0,0]$, $y\_move[0,0,1,-1]$, $Queue\ Q$

```
1  Represent each Cartesian point as node
2  Initialize Start State S
3  Initialize Goal State G
4  level[S] = 0
5  Q = empty
6  ENQUEUE(Q, S)
7  while Q not empty do
8      Node u = DEQUEUE(Q)
9      for each child node v ∈ adj[u] do
10         if (v_x and v_y are valid considering x_move and y_move) & graph[v_x][v_y] == free
           then
11             if v == G then
12                 goal_flag = true;
13                 break;
14             end
15             graph[v_x][v_y] == visited
16             level[v] = level[u] + 1
17             ENQUEUE(Q, v)
18         end
19     end
20     if goal_flag == true then
21         Print(Goal found);
22     else
23         Print(Goal can not be reached);
24     end
25 end
```

5 Implementation in Java

```
1  package bfs_2d;
2
3  import java.util.LinkedList;
4  import java.util.Queue;
5
6  /**
7   *
8   * @author Jargis
9   */
10 class node {
11
12     int x, y;
13     int level;
14     node(int a, int b, int z) {
15         x = a;
16         y = b;
17         level = z;
18     }
19 }
20
21 class BFS {
22
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      BFS() {
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          StBFS(graph);
53          if (found) {
54              System.out.println("Goal found");
55              System.out.println("Number of moves required = " + goal_level);
56          } else {
57              System.out.println("Goal can not be reached from starting block");
58          }
59      }
60
61      void StBFS(int[][] graph) {
62
63          Queue<node> q = new LinkedList<node>();
64          q.add(source);
65
66          while (!q.isEmpty()) {
67              node u = q.poll();
68
69              for (int j = 0; j < directions; j++) {          //calculating up,
                    down, left and right directions
70                  int v_x = u.x + x_move[j];
71                  int v_y = u.y + y_move[j];                  //check the boundary
                        conditions
72                  if ((v_x < N && v_x >= 0) && (v_y < N && v_y >= 0) && graph[v_x
                        ][v_y] == 1) {
73                      int v_level = u.level + 1;
74                      if (v_x == goal.x && v_y == goal.y)   //goal check
75                      {
76                          found = true;
77                          goal_level = v_level;
78                          goal.level = v_level;
79                          break;
80                      }
```

```
                    graph[v_x][v_y] = 0;
                    node child = new node(v_x, v_y, v_level);
                    q.add(child);
                }
            }
            if (found == true) {
                break;
            }
        }
    }
}

public class BFS_2D {

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

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

Output:
Goal found
Number of moves required = 6

7 Implementation in Python

```
from collections import deque

class Node:
    def __init__(self, x, y, level):
        self.x = x  # x-coordinate of the node
        self.y = y  # y-coordinate of the node
        self.level = level  # level of the node in the BFS tree

class BFS:
    def __init__(self):
        self.directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]  # Possible
            movement directions
        self.found = False  # Flag to indicate whether goal is found
        self.goal_level = 0  # Level of the goal node
        self.N = 0  # Size of the grid
        self.source = None  # Source node
        self.goal = None  # Goal node

    def init(self):
        # Initialize the grid
        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)

        # Define source and goal nodes
        source_x, source_y = 0, 2
        goal_x, goal_y = 4, 4
        self.source = Node(source_x, source_y, 0)
        self.goal = Node(goal_x, goal_y, float('inf'))

        # Perform BFS
        self.st_bfs(graph)

        # Print result
        if self.found:
            print("Goal found")
            print("Number of moves required =", self.goal_level)
        else:
            print("Goal cannot be reached from starting block")

    def st_bfs(self, graph):
        queue = deque()
        queue.append(self.source)

        # Breadth-first search
        while queue:
            u = queue.popleft()  # Dequeue the front node

            # Explore neighbors
            for dx, dy in self.directions:
                v_x, v_y = u.x + dx, u.y + dy

                # Check if neighbor is within grid boundaries and is a valid
                    move
                if 0 <= v_x < self.N and 0 <= v_y < self.N and graph[v_x][v_y]
                    == 1:
                    v_level = u.level + 1  # Increment level
                    if v_x == self.goal.x and v_y == self.goal.y:  # Check if
                        neighbor is the goal
                        self.found = True
                        self.goal_level = v_level
                        self.goal.level = v_level
                        return
                    graph[v_x][v_y] = 0  # Mark neighbor as visited
                    child = Node(v_x, v_y, v_level)
                    queue.append(child)  # Enqueue the neighbor

if __name__ == "__main__":
    bfs = BFS()
    bfs.init()
```

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

Output:
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 BFS 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. Print the required moves throughout the traversal along with the tiles coordinate.
   For example -
   Moving Down -> (2, 3)
   Moving Right -> (2, 4)
   ...
   etc.

10 Lab Exercise (Submit as a report)

- Write a program where a robot traverses on a 2D plane using the BFS algorithm and goes from start to destination point. Now print the path it will traverse to reach the destination.
  (***Hint: Save parent state information and use that information to write a recursive function for printing the path.***)

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.