Maze

Norina Akhtar
19643

Abstract

The project is based on maze problem . Maze has starting point and ending point and composed of walls. 0 indicates there is path to move ,while 1 means there is a wall. With the help of DFS we can retrieve the first available path without any concern of shortest route.

Objectives

There is a ball in maze with empty spaces represented as 0 and wall(as 1). Our goal is to find the path for ball through empty spaces by rolling **up**,

down, **left**, or right, but it won't stop the rolling until hitting a wall . When the ball stops , it could choose the next direction.

Method Used

Depth First Approach has been used to get the possible path for ball. Depth-first search is a graph/tree traversal algorithm that follows a path as far as it can until it either, reaches the goal or has nowhere else to go. It’s almost like running as far as you can in one direction until you hit a wall .Hence, DFS does not concerned with finding the shortest path.

Scope of work

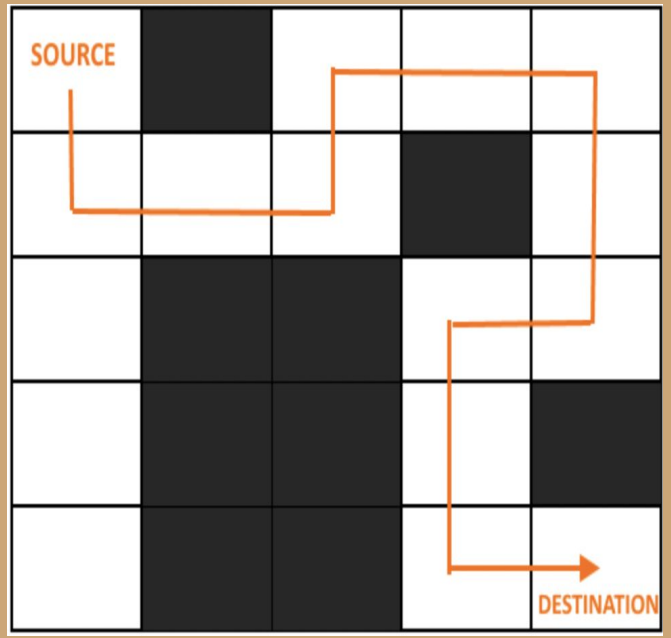
The purpose of scope of work is the implementation of project assigned by Dr. Chang, Henry which is maze problem .

The problem can be solved by using using different approaches i-e Dijkstra's algorithm, breadth first approach and Depth first approach etc.

The selection is algorithm is based on the constraints which is to find a path typically from a source to a destination. Hence, shortest path is not required so we have followed the DFS approach.

What has been Achieved

The path has been found from source to destination.



Acknowledgment

I would like to express my deepest gratitude to my Professor, Dr. Chang, Henry whose patience, guidance and questioning was critical to the completion of this study. I have not only acquired broad knowledge in my field from taking his courses but getting prepare for the professional career also.

I would also like to express my sincere thanks to my TA , Yixin Cao for making my doubts clear and providing constructive suggestions throughout the course.

Table of contents

Problem Description

Robot - Clear Route (Street, Highway)

The planner's problem is then to find a sequence of stances in L which take the robot from the start to the goal, such that the stances and the surfaces. It requires a clear path from start and can go as far as it can , and backtracks until it finds an unexplored path and then explore it step by step.

Self Driving Car- Unclear Route (Hotel, Hospital)

While the self driving car detects curbs and other vehicles when parking. Its like while rolling in an empty space until it detects any hurdle or curb.

Self driving cars are looking for available space move in that direction until it achieve its desired path.

Introduction

A maze is a 2D matrix in which some cells are blocked. One of the cells is the source cell, from where we have to start. And another one of them is the destination, where we have to reach. We have to find a path from the source to the destination without moving into any of the blocked cells.

How to design

Identify and Understand the problem

1:Walkable Path

0: Non-walkable Path

S:Starting point (start will be (0,0))

D:Destination

When you see problem first thing should come to your mind is what if starting point is not walkable(0) if it is not walkable then you should immediately return False or Not possible.

If the ball doesn't stop at destination because the ball is rolling through all cell, It means there is no possible way for ball.

Investigation

We'll solve our generated maze with two different methods:

1. Depth first search
2. Breadth first search

If you're not concerned about memory, you can pick either. If you're fairly comfortable with recursion, DFS should be easier to implement

They have similar running time, but either may greatly outperform the other on any given problem simply due to the order in which the cells are visited.

Using depth first search will generally be quicker because it is stack based algorithm ,easier to implement (leveraging the call stack of the recursive algorithm) rather than explicit (requiring a "wrapper" function that creates a queue and a "helper" function that receives that queue as a by-reference parameter.

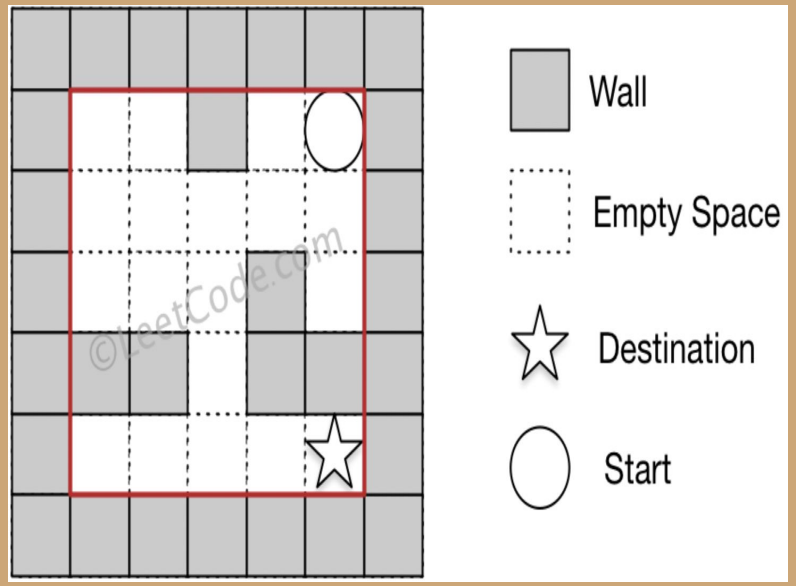
Solutions

For mazes specifically ,if we define a maze as there being only one way to reach a cell from the starting point without backtracking, meaning it's essentially a tree , BFS will generally use more memory, as we'll need to keep multiple paths in memory at the same time, where DFS only needs to keep track of a single path at any given time .

This will find a solution, but it won't necessarily find the shortest solution. It focuses on you, is fast for all types of Mazes, and uses stack space up to the size of the Maze. If you're at a wall (or an area you've already plotted), return failure, else if you're at the finish, return success, else recursively try moving in the four directions. Plot a line when you try a new direction, and erase a line when you return failure, and a single solution will be marked out when you hit success.

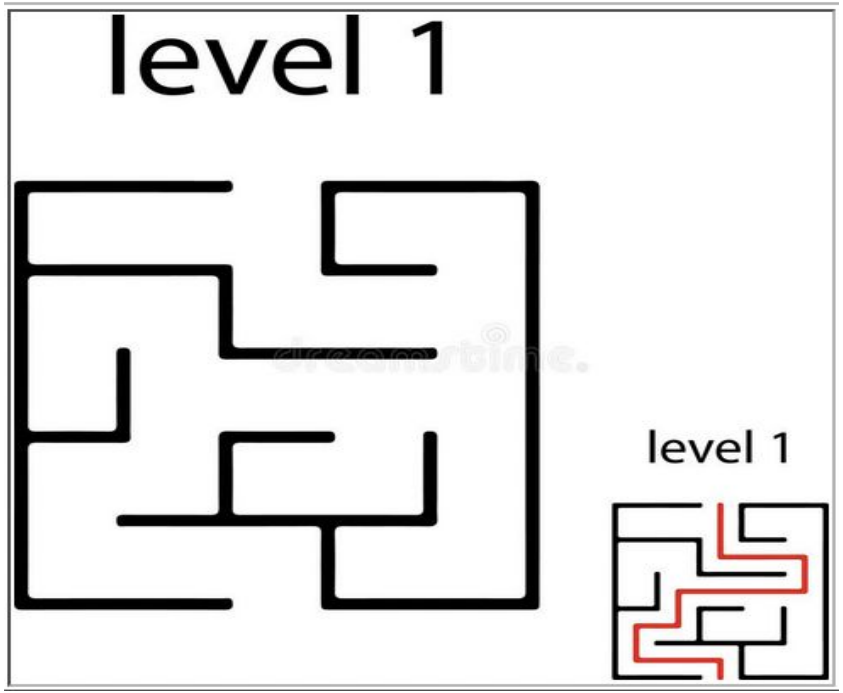
Objective

Our objective is to find the possible way from m , the balls start position to n the destination , where start =[start(row),start(col)] and destination = [destination(row), destination(col)] ,return true if the ball can stop at the destination, otherwise return false.



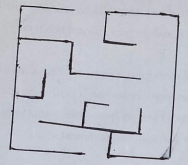
Tree : Depth first traversal on a maze

A maze can be viewed as a graph, if we consider each juncture (intersection) in the maze to be a vertex, and we add edges to the graph between adjacent junctures that are not blocked by a wall

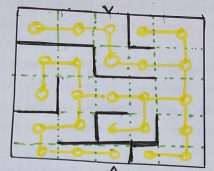


Implementation

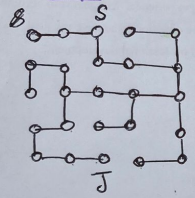
Step 1



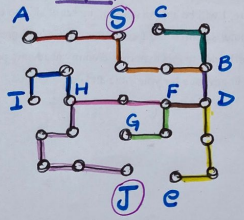
Step 2



Step 3



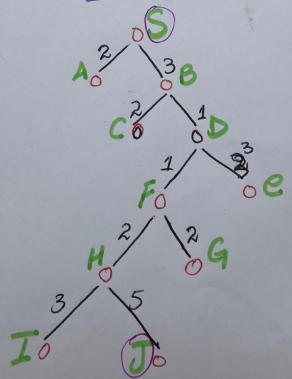
Step 4



Step 5

S = Source

J = Destination



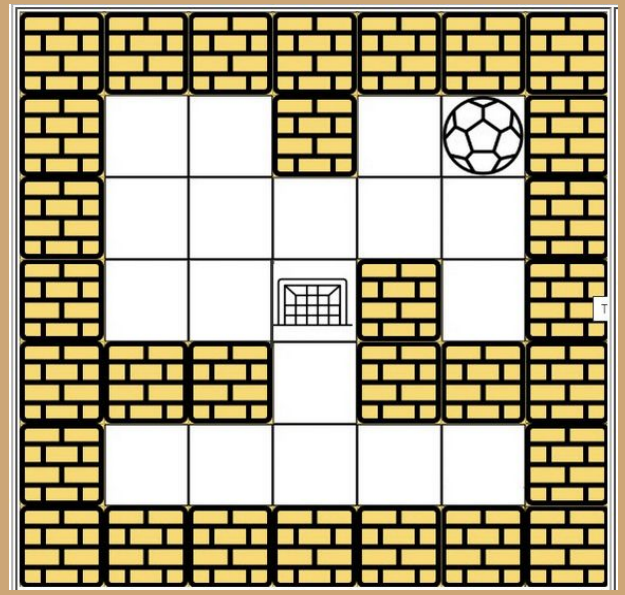
Example 2: Approach 2

Depth First Traversal

The ball can move only one cell at a time

The search sequence is

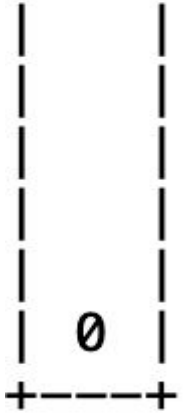
Right ==› Left ==› Top ==› Bottom

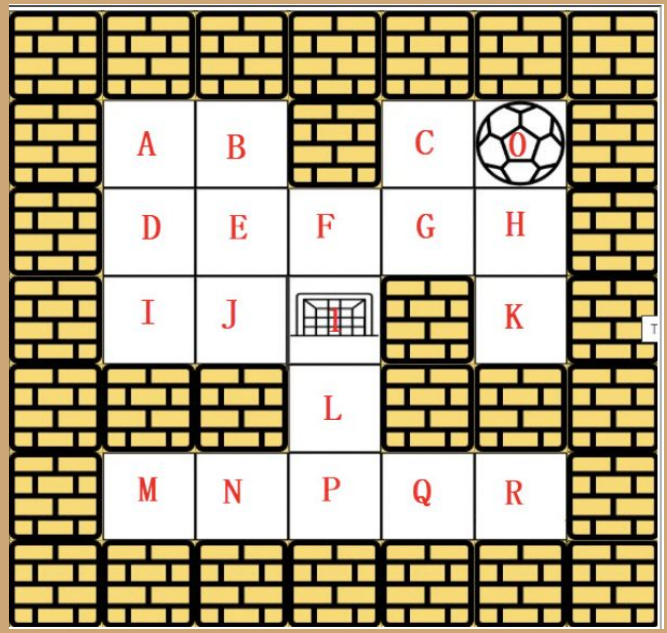


Example 2: Approach 2

Step1 :

From starting point 0:

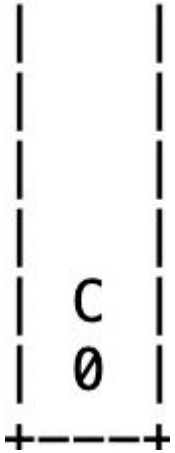


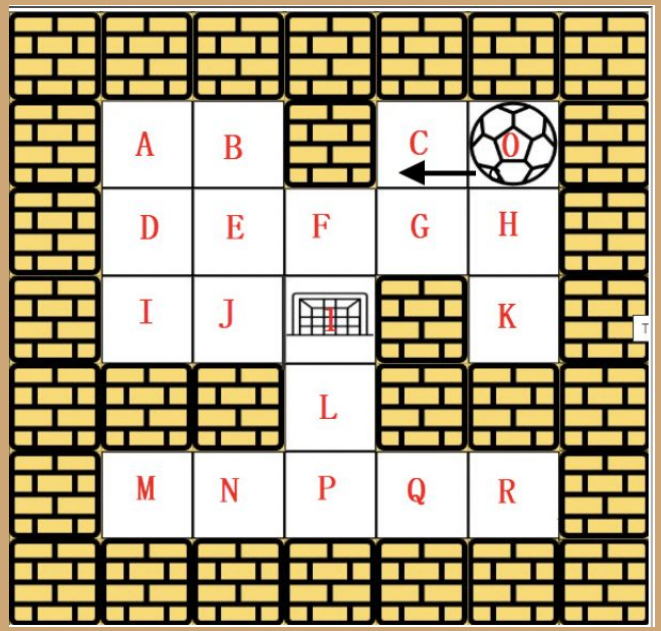


Example 2: Approach 2

Step 2: 0 - C

Right , left ,up , Down

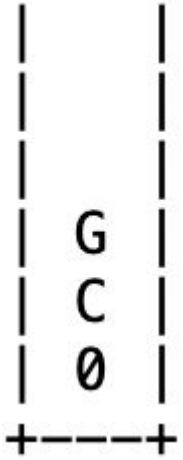


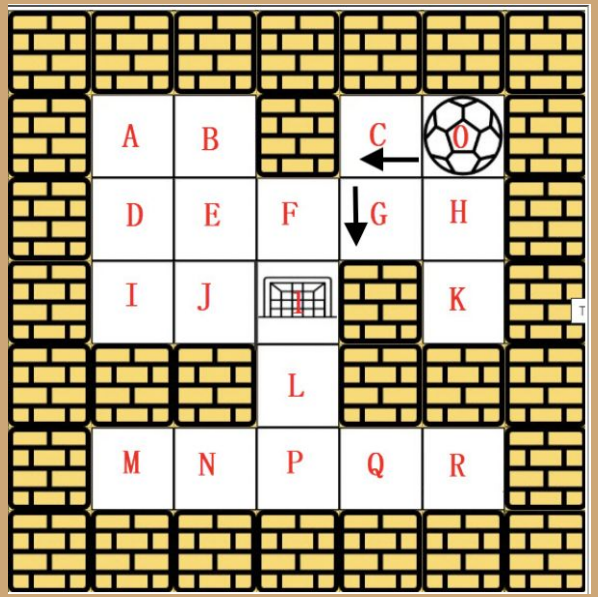


Example 2: Approach 2

Step 3: 0 - C - G

Right , left ,up , Down





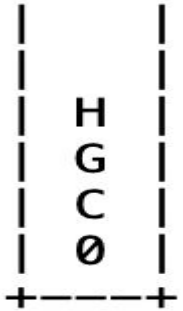
Example 2

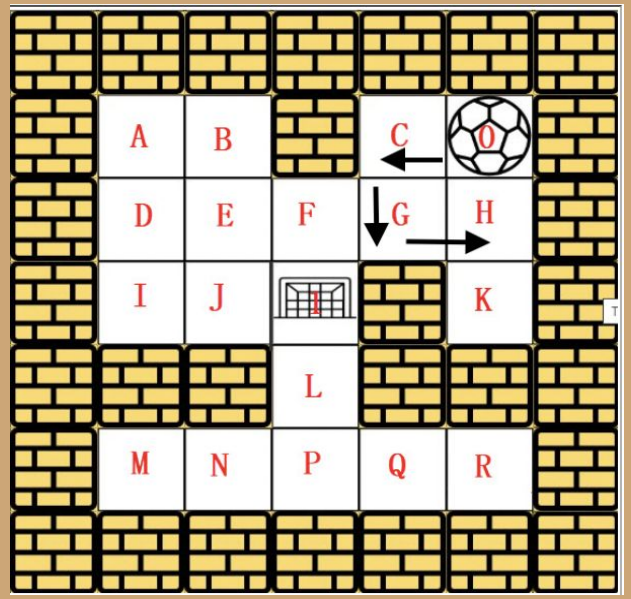
Step 4: 0 - C - G - H

Right , left ,up , Down

From G ball has four options i-e C ,F and h,

C is already visited , so it can't go back, it will choose H because priority of H is higher according to the rule

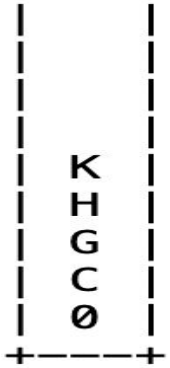


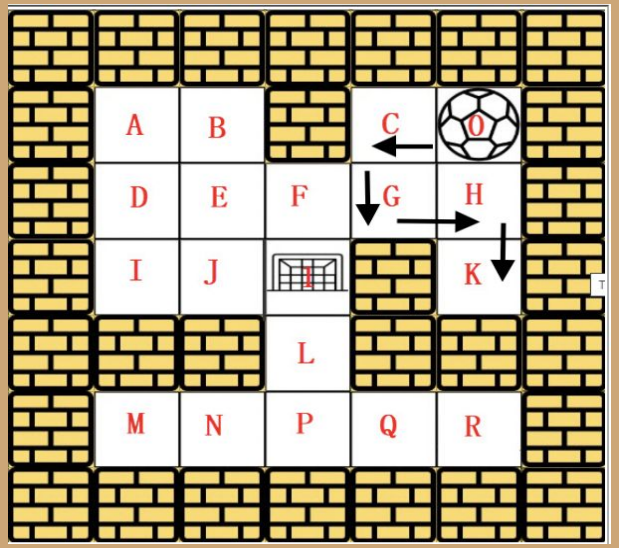


Example 2

Step 5: **0 - C - G - H - k**

Right , left ,up , **Down**

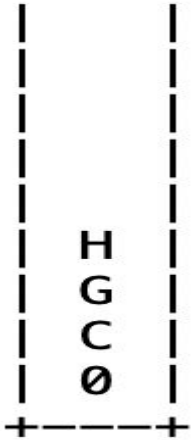


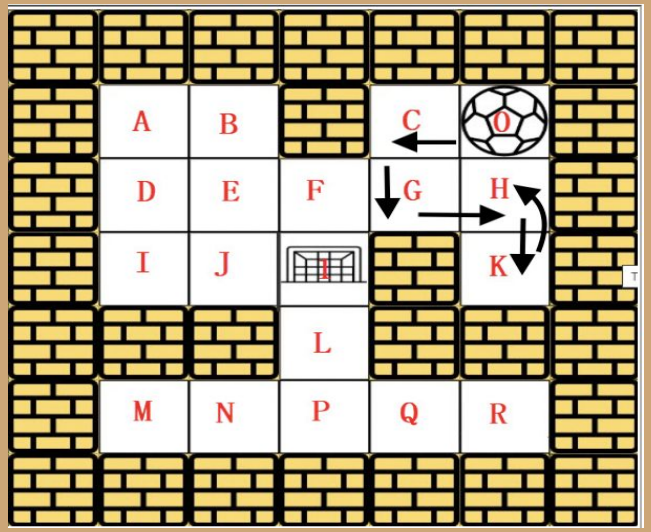


Example 2

Step 6: **0 - C - G - H**

Backtracking because it's not the destination and pop k node from stack

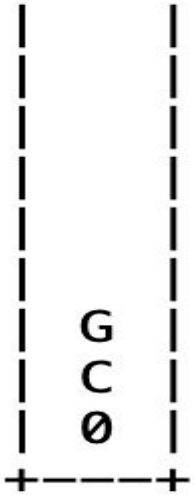


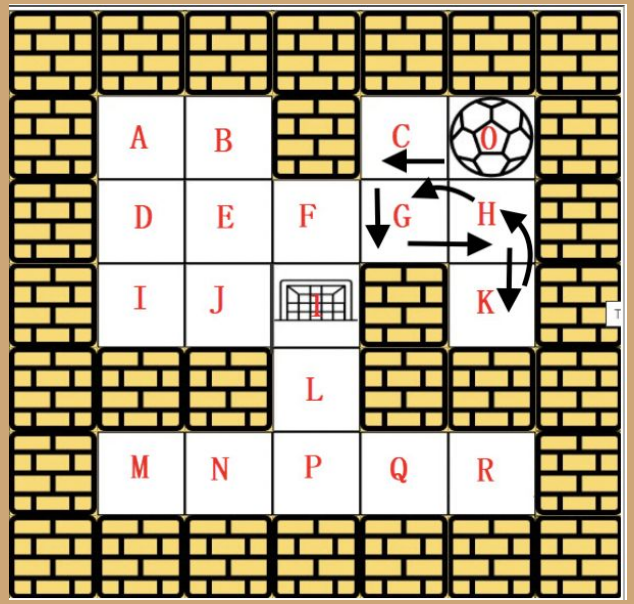


Example 2

Step 7: **O - C - G - H**

Pop h node from stack

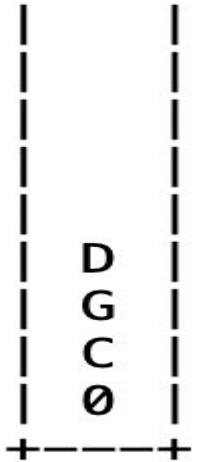


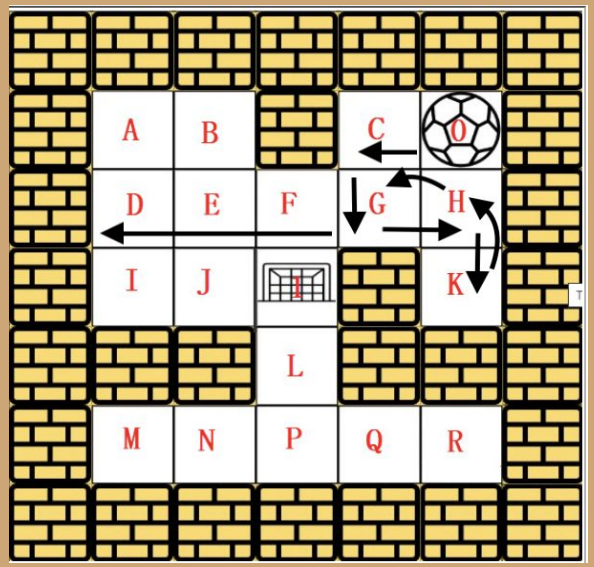


Example 2

Step 8: **0 - C - G - H -D**

Right , **left** ,up , Down

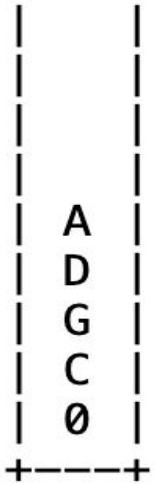


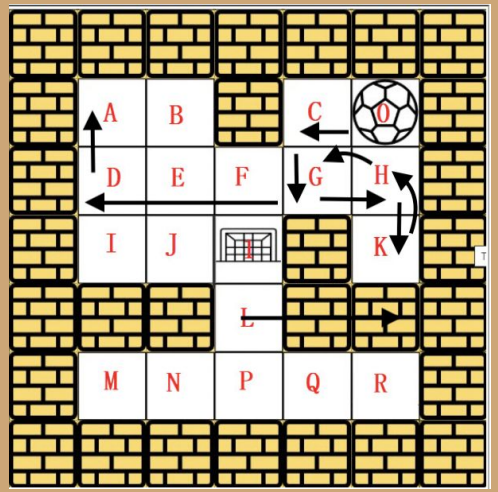


Example 2

Step 9: 0 - C - G - H - D - A

Right , left ,**up** , Down



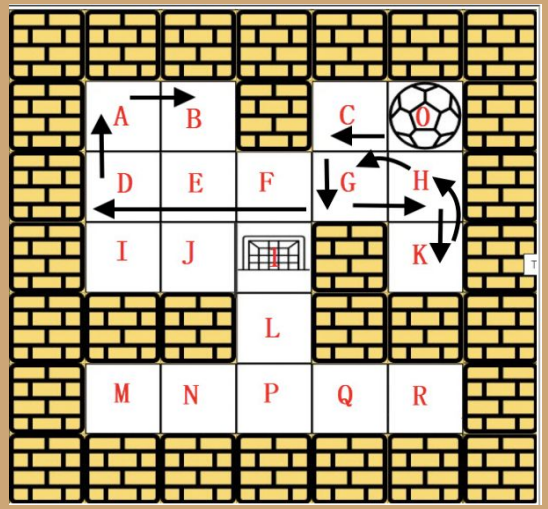


Example 2

Step 10: 0 - C - G - H - D - A - B

Right , left ,up , Down

```
|     |
|     |
|     |
|     |
| B   |
| A   |
| D   |
| G   |
| C   |
| 0   |
+---+
```

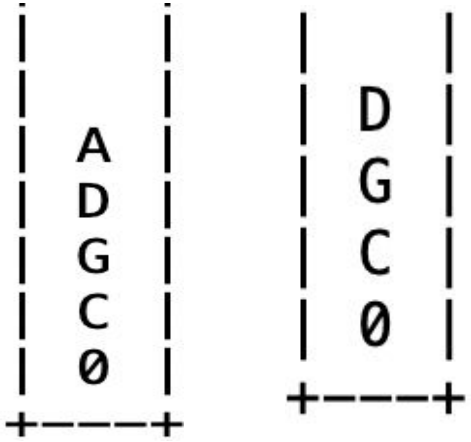


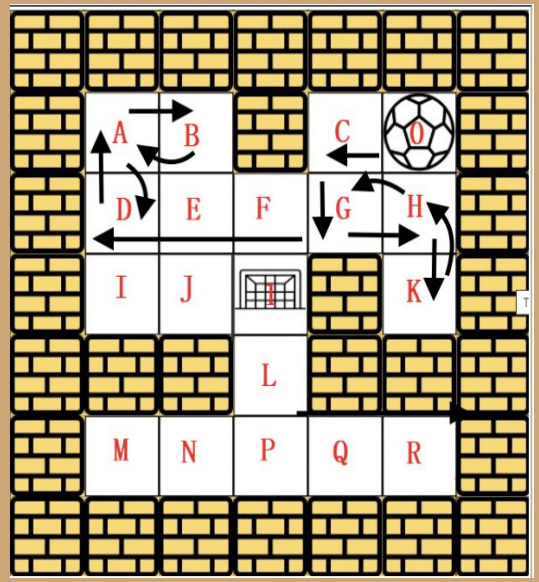
Example 2

Step 11 : **0 - C - G - H - D - A , 0 - C - G - H - D**

Right , left ,up , Down

Pop B and A from stack



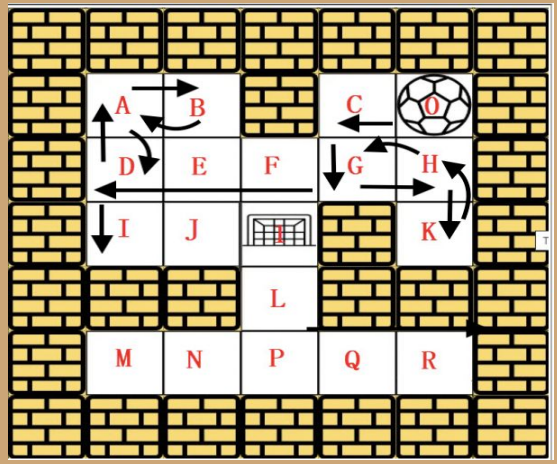


Example 2

Step 12 : **0 - C - G - H - D - I**

Right , left ,up , **Down**

```
+---+
|   |
| I |
| D |
| G |
| C |
| 0 |
+---+
```



Example 2

Step 13 : 0 - C - G - H - D - I - 1

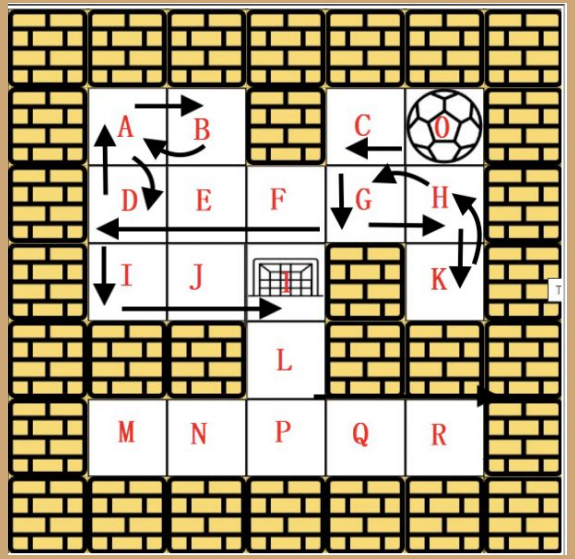
Right , **left** ,up , Down

One possible way is:

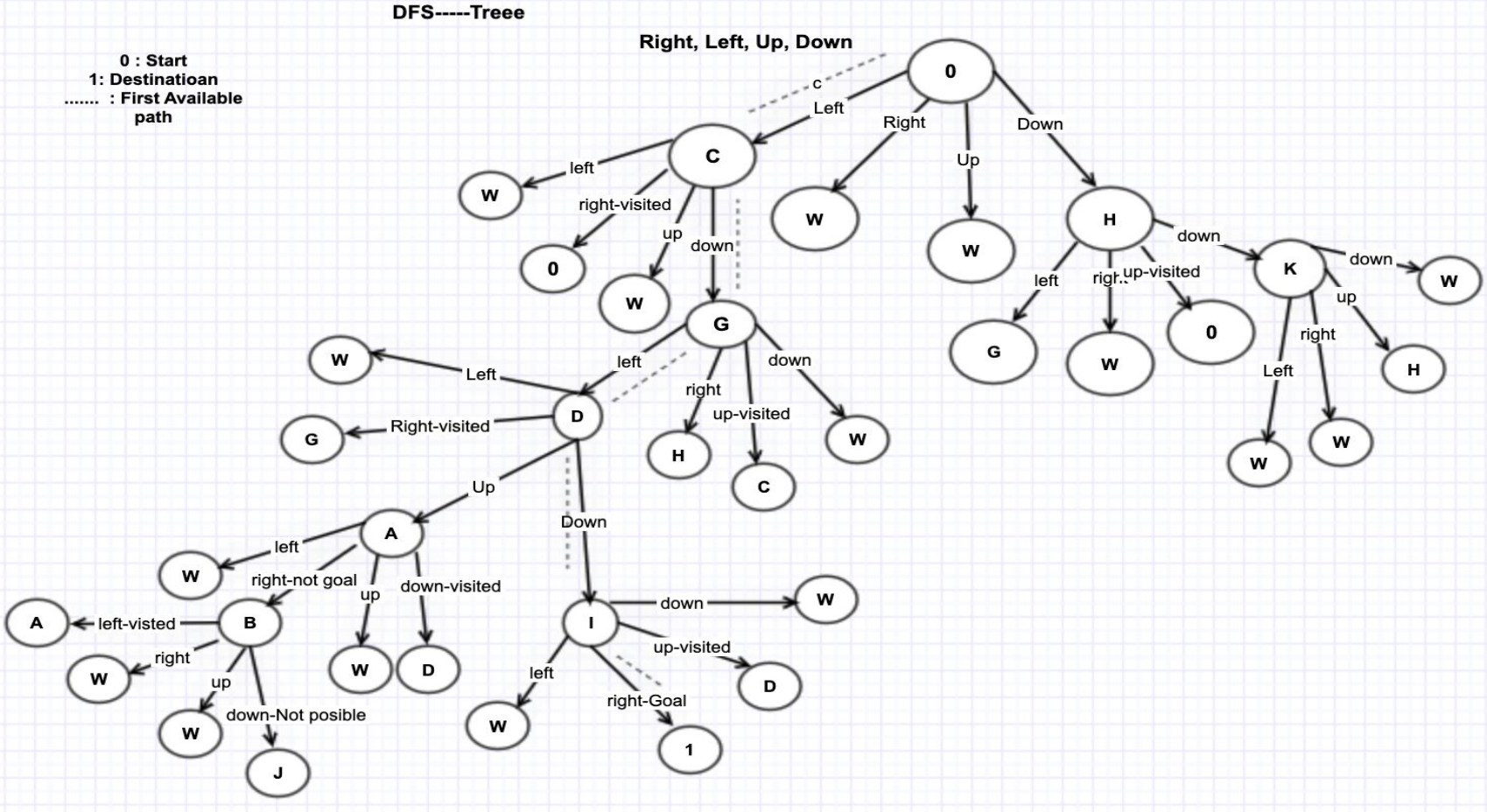
Left - down - left - down - right

The ball will stop looking for other Node because it reached at its destination.

```
+---+
|   |
| 1 |
| I |
| D |
| G |
| C |
| 0 |
+---+
```



Approach 1 Depth First Search : TREE

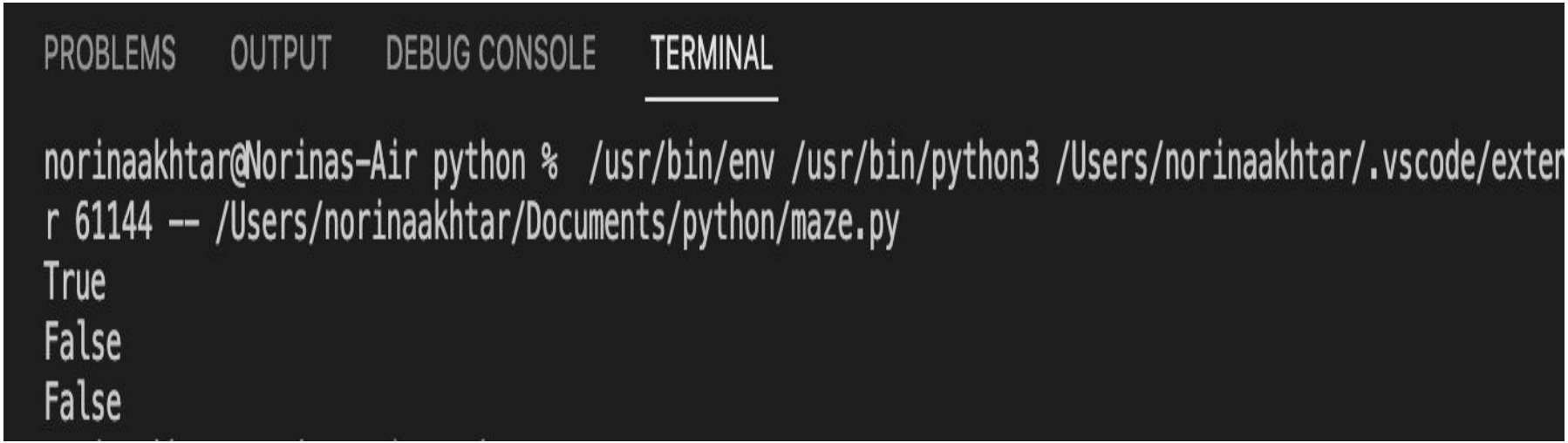

DFS-----Treee
Right, Left, Up, Down
0 : Start
1: Destinatioan
..... : First Available path
0
c
Left
Right
Up
Down
C
left
W
right-visited
0
up
W
down
G
W
Left
left
D
G
Right-visited
right
up-visited
H
C
W
Up
Down
A
left
W
right-not goal
up
down-visited
A
left-visted
B
W
right
up
W
down-Not posible
J
W
D
I
down
W
up-visited
D
left
W
right-Goal
1
W
Left
H
W
left
right
up-visited
G
W
0
H
down
K
down
W
up
right
Left
W
W

Python - code

maze.py > ...

```python
from typing import List
class Solution:
    def hasPath(self, maze: List[List[int]], start: List[int], destination: List[int]) -> bool:
        directions= [(1,0),(-1,0),(0,-1),(0,1)]
        m = len(maze)
        n = len(maze[0])

        stack =[]
        seen = set()
        stack.append((start[0],start[1]))
        seen.add((start[0],start[1]))
        while stack:
            cur_i,cur_j =stack.pop()
            for d in directions:
                ni = cur_i
                nj = cur_j
                while 0 <= ni < m and 0 <= nj < n and maze[ni][nj] == 0:
                    ni += d[0]
                    nj += d[1]

                ni -= d[0]
                nj -= d[1]

                if ni == destination[0] and nj == destination[1]:
                    return True

                if(ni,nj) not in seen:
                    stack.append((ni,nj))
                    seen.add((ni,nj))
        return False

```

Test


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
norinaakhtar@Norinas-Air python % /usr/bin/env /usr/bin/python3 /Users/norinaakhtar/.vscode/exter
r 61144 -- /Users/norinaakhtar/Documents/python/maze.py
True
False
False

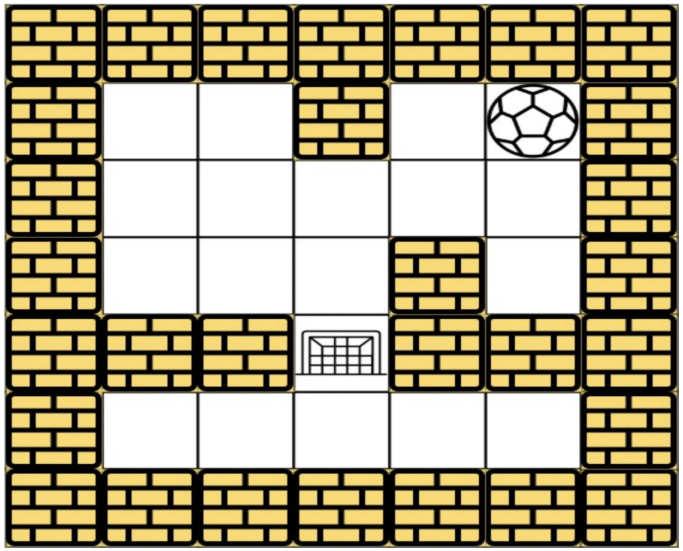
Enhancement Ideas

The problem can be solved by using using different approaches i-e Dijkstra's algorithm, Bellman Ford’s algorithm, breadth first approach and Depth first approach etc.

However, DFS is the easiest approach to implement and shortest path is not required thats why DFS has been considered for this problem.

Drawback

The drawback to the Depth first search is that it can get stuck going down the wrong path and never recover, with respect to time, from an unlucky choice at one of the nodes near the top of the tree. The Depth first search will continuing going downward even though a solution may be very close to the top of the tree.



Conclusion

The time complexity for a Depth first search is $O(b^m)$, where b is the branching factor and m is the maximum depth.

Depth first search has moderate memory requirements. For a Depth first search in memory, it only needs to store a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on the path. With a branching factor of b and a maximum depth m, the Depth first search requires storage of only bm nodes

References

- Depth-First Search (DFS)
- Graphs and Maze
- Problem F: The Maze Makers
- Mazes -
- 490. The Maze,
- LeetCode 490. The Maze -