Mahmud Hamed Omer, 19660

Week9_hw1

34. Project : "490. The Maze" - LC - Depth-First Traversal
- 490. The Maze - (local copy) - Medium
  - Process
    - Step 1: Manual process to demonstrate concepts

| Robot | Clear Route (Street, Highway) | Unclear Route (Hotel, Hospital) |
|---|---|---|
| **Without Wheel (Legged Robot)** | Step 1.1: Tree<br>• Following the examples shown on Depth-First Traversal to manually solve the problem<br>▪ Maze example | |
| **With Wheel (Self-driving Car)** | | Step 1.2: Matrix<br>• Following the examples shown on Depth-First Traversal to manually solve the problem<br>▪ Maze example -- assuming the ball can go through the empty spaces by rolling. |

    - Step 2: Implement a Python solution using the algorithm Depth-First Traversal and test the Python code
      - To prove that you can convert a concept into a program (Sample code) and test the program based on all the test cases provided by LeetCode 490. The Maze - (local copy)
        - Please study the programs. Since the program is provided, there is not much you can do if you decide not to study the programs.
    - Step 3: Update your portfolio about the Maze project
      - Please use this structure to describe the project

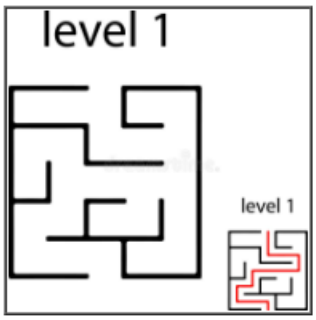
```
Algorithm
   Depthe First Search
      Maze
```

    - Step 4: Submit the URL of your GitHub webpage as the homework answer.
  - References
    - Subject: Depth-First Search - more similar questions
    - Maze
    - 490. The Maze, medium, BFT abnd DFT - LC
    - Leet Code 490. The Maze — Explained Python3 Solution
    - LeetCode 490. The Maze - Youtube
    - 490 The Maze - Java solution

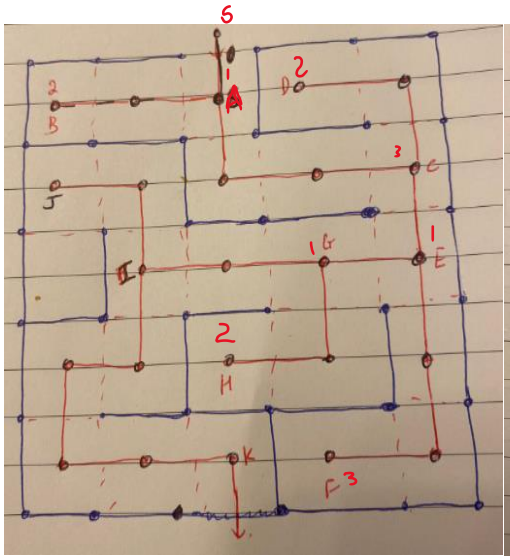
ANS:

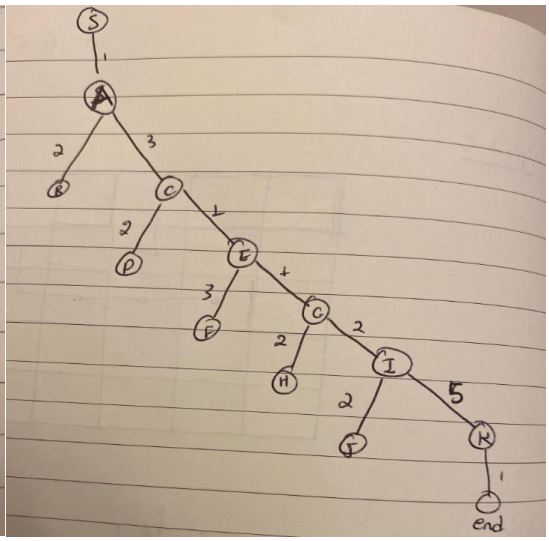
1. Step 1.1

35. Conduct Depth First Traversal (DFT) on a maze - Level 1 Maze



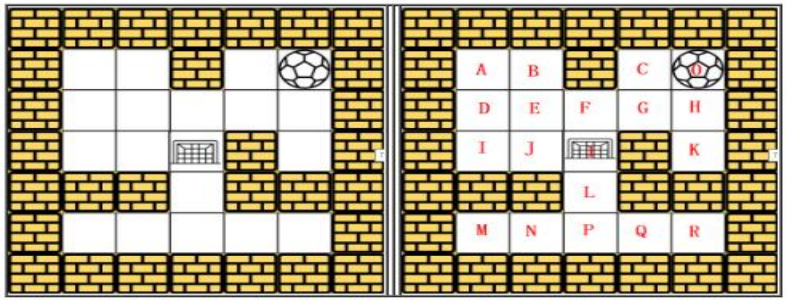

- References
  - Maze
  - Depth First Traversal (DFT)




step 1.2.

39. Depth-First Traversal for matrix maze
- Please refer the concepts shown on Maze to draw the detailed steps on using Depth-First Traversal to find the path.




- The search sequence is

```
Right ==> Left ==> Top ==> Bottom
```

- References
  - Depth-First Traversal
  - Maze

ANS:

**Process: right, left, up or down**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | 1 |
| | | | | | | | | | | J | J |
| | | | | | | | | | B | B | B |
| | | | | k | | | | A | A | A | A |
| | | | H | H | H | | D | D | D | D | D |
| | | G | G | G | G | G | G | G | G | G | G |
| | C | C | C | C | C | C | C | C | C | C | C |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Step 2:

```
class Solution:
    def hasPath(self, maze, start, destination):
        m, n, stopped = len(maze), len(maze[0]), set()

        def dfs(x, y):
            if (x, y) in stopped:
                return False
            stopped.add((x, y))
            if [x, y] == destination:
                return True
            for i, j in (-1, 0), (1, 0), (0, -1), (0, 1):
                newX, newY = x, y
                while 0 <= newX + i < m and 0 <= newY + j < n and maz
e[newX + i][newY + j] != 1:
                    newX += i
                    newY += j
                if dfs(newX, newY):
                    return True
            return False
        return dfs(*start)
maze = [[0, 0, 1, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 1, 0], [1, 1, 0,
1, 1], [0, 0, 0, 0, 0]]
start = [0, 4]
destination = [4, 4]
obj = Solution()
print(obj.hasPath(maze, start, destination))
```

Result: