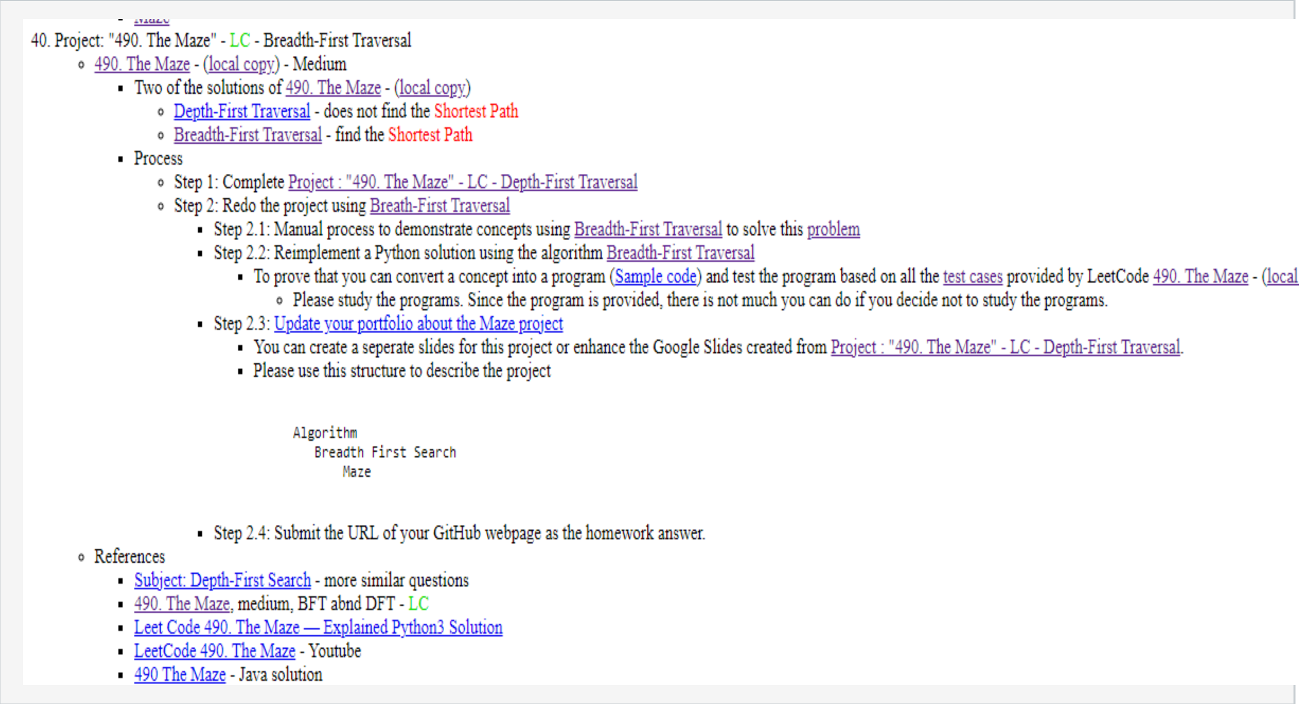
Mahmud Hamed Omer, 19660

Week9_hw2



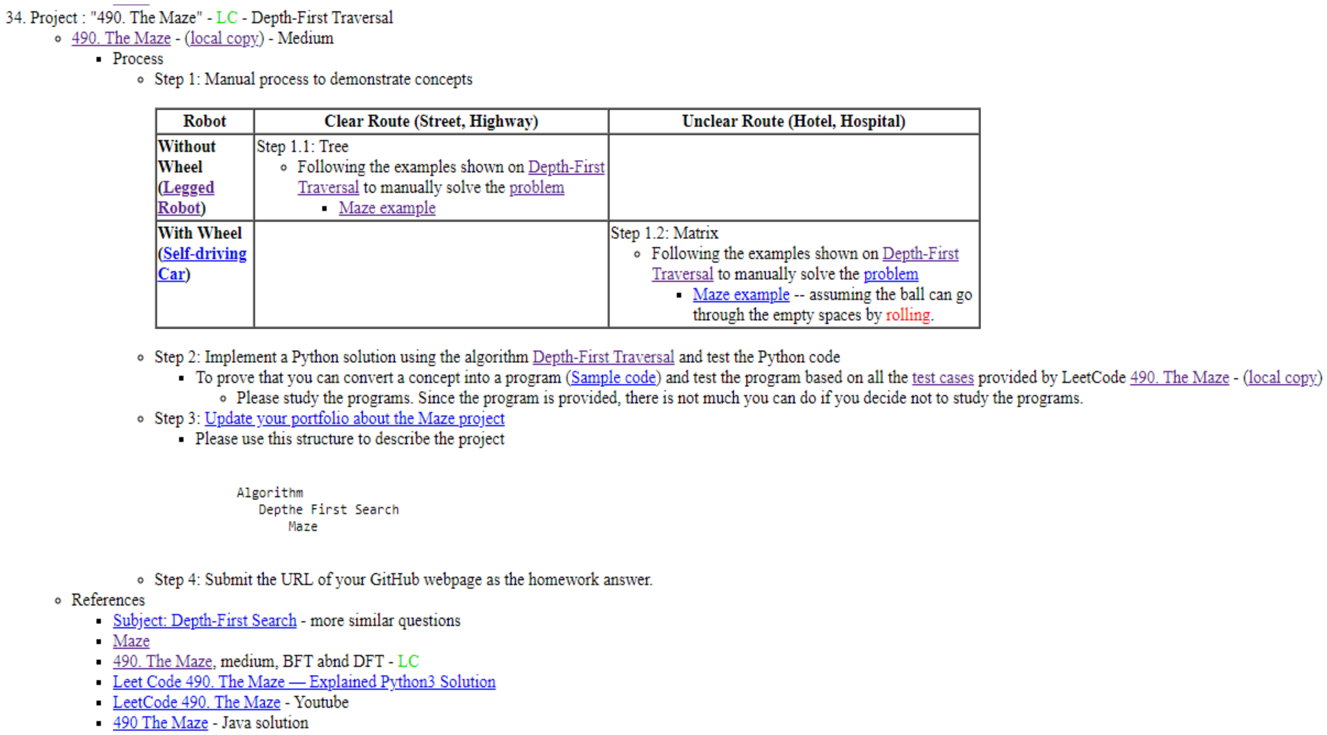
- Maze

40. Project: "490. The Maze" - LC - Breadth-First Traversal
  - 490. The Maze - (local copy) - Medium
    - Two of the solutions of 490. The Maze - (local copy)
      - Depth-First Traversal - does not find the Shortest Path
      - Breadth-First Traversal - find the Shortest Path
    - Process
      - Step 1: Complete Project : "490. The Maze" - LC - Depth-First Traversal
      - Step 2: Redo the project using Breath-First Traversal
        - Step 2.1: Manual process to demonstrate concepts using Breadth-First Traversal to solve this problem
        - Step 2.2: Reimplement a Python solution using the algorithm Breadth-First Traversal
          - 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 2.3: Update your portfolio about the Maze project
          - You can create a seperate slides for this project or enhance the Google Slides created from Project : "490. The Maze" - LC - Depth-First Traversal.
          - Please use this structure to describe the project

```
Algorithm
   Breadth First Search
       Maze
```

        - Step 2.4: Submit the URL of your GitHub webpage as the homework answer.
  - References
    - Subject: Depth-First Search - more similar questions
    - 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

Step 1:



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

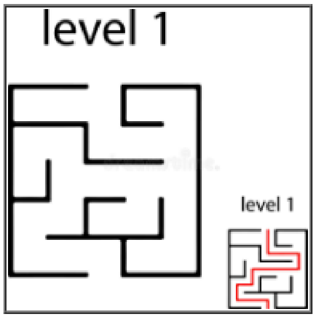
```
Algorithm
   Depthe First Search
       Maze
```

      - 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

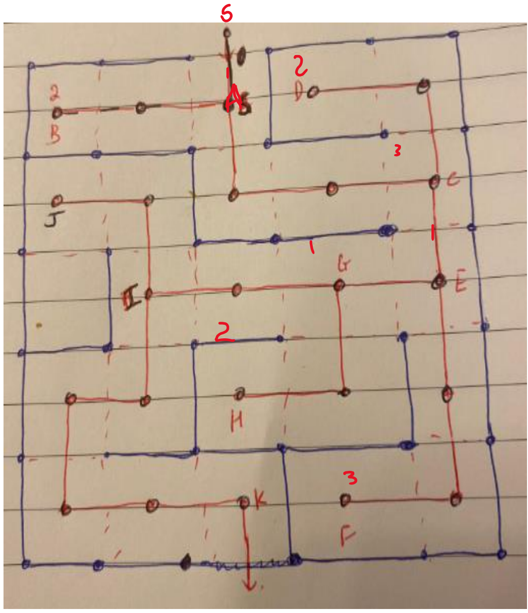
ANS:

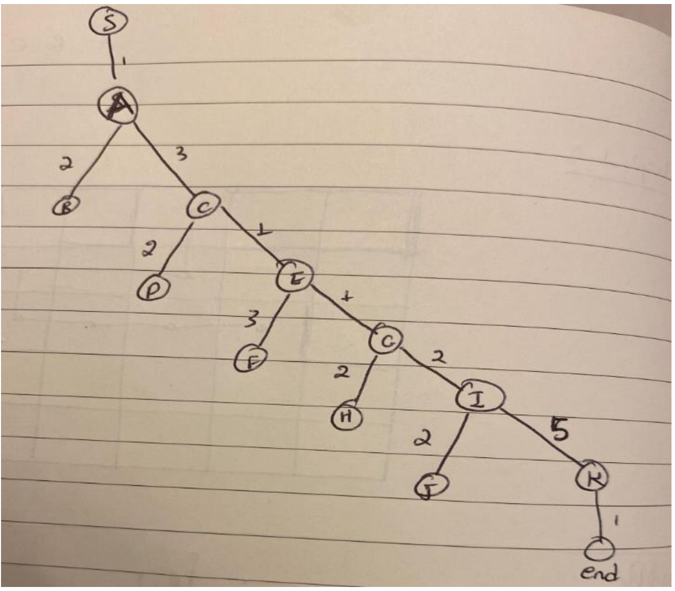
1. Part 1

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



- References
  - Maze
  - Depth First Traversal (DFT)

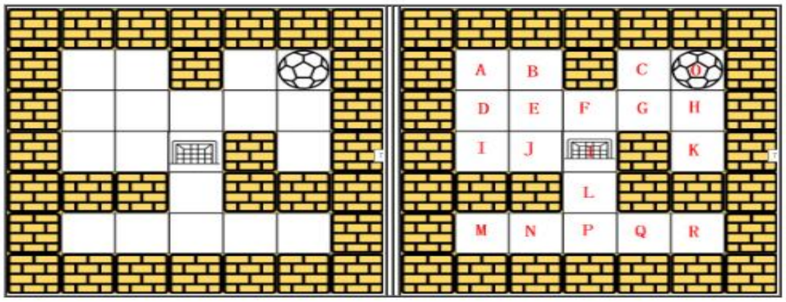




Part II. .

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
```

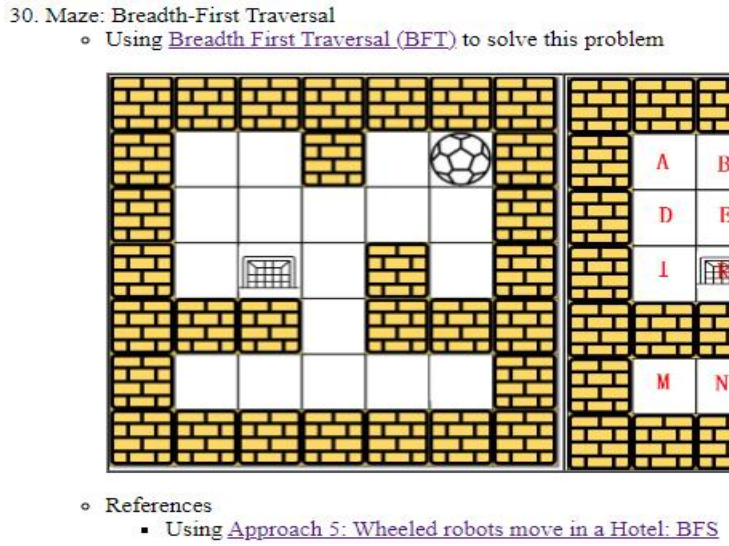
- References
  - Depth-First Traversal
  - Maze

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 2.1: Manual process to demonstrate concepts using Breadth-First Traversal to solve this problem  given below



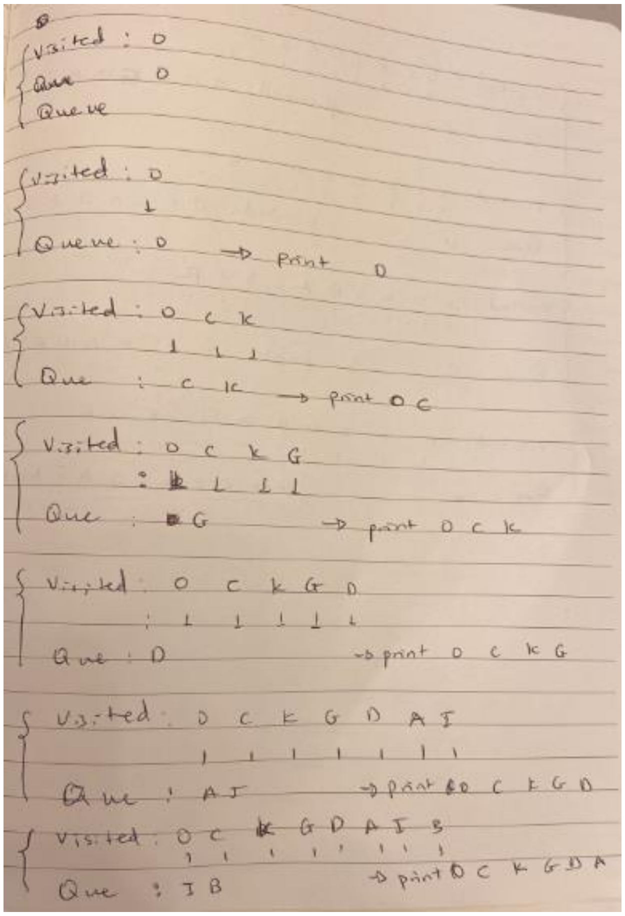
30. Maze: Breadth-First Traversal

- Using Breadth First Traversal (BFT) to solve this problem

- References
  - Using Approach 5: Wheeled robots move in a Hotel: BFS

ANS:

The problem is solved using the wheeled robot's approach using BFS, where the ball can go through the empty spaces by rolling right, left, up, down, but it won't stop rolling until hitting a wall. When the ball stops, it can choose the next direction as shown in the movement steps below. The result is highlighted in red



Visited : O
Que O
Queue

Visited : O
1
Queue : O → Print O

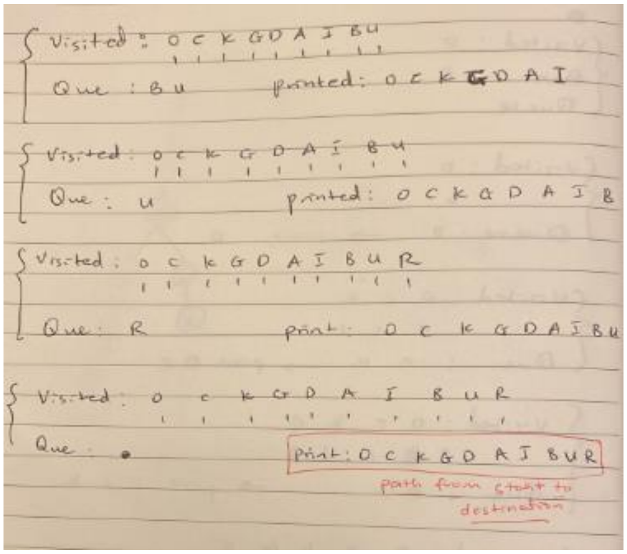
Visited : O C K
1 1 1
Que : C K → print O C

Visited : O C K G
: 1 1 1 1
Que : G → print O C K

Visited O C K G D
: 1 1 1 1 1
Que : D → print O C K G

Visited O C K G D A I
1 1 1 1 1 1 1
Que : A J → print O C K G D

Visited : O C K G D A I B
1 1 1 1 1 1 1 1
Que : I B → print O C K G D A



Visited : O C K G D A I B U
1 1 1 1 1 1 1 1 1
Que : B U printed: O C K G D A I

Visited : O C K G D A I B U
1 1 1 1 1 1 1 1 1
Que : U printed: O C K G D A I B

Visited : O C K G D A I B U R
1 1 1 1 1 1 1 1 1 1
Que : R print : O C K G D A I B U

Visited : O C K G D A I B U R
1 1 1 1 1 1 1 1 1 1
Que : •

Print: O C K G D A I B U R

path from start to destination

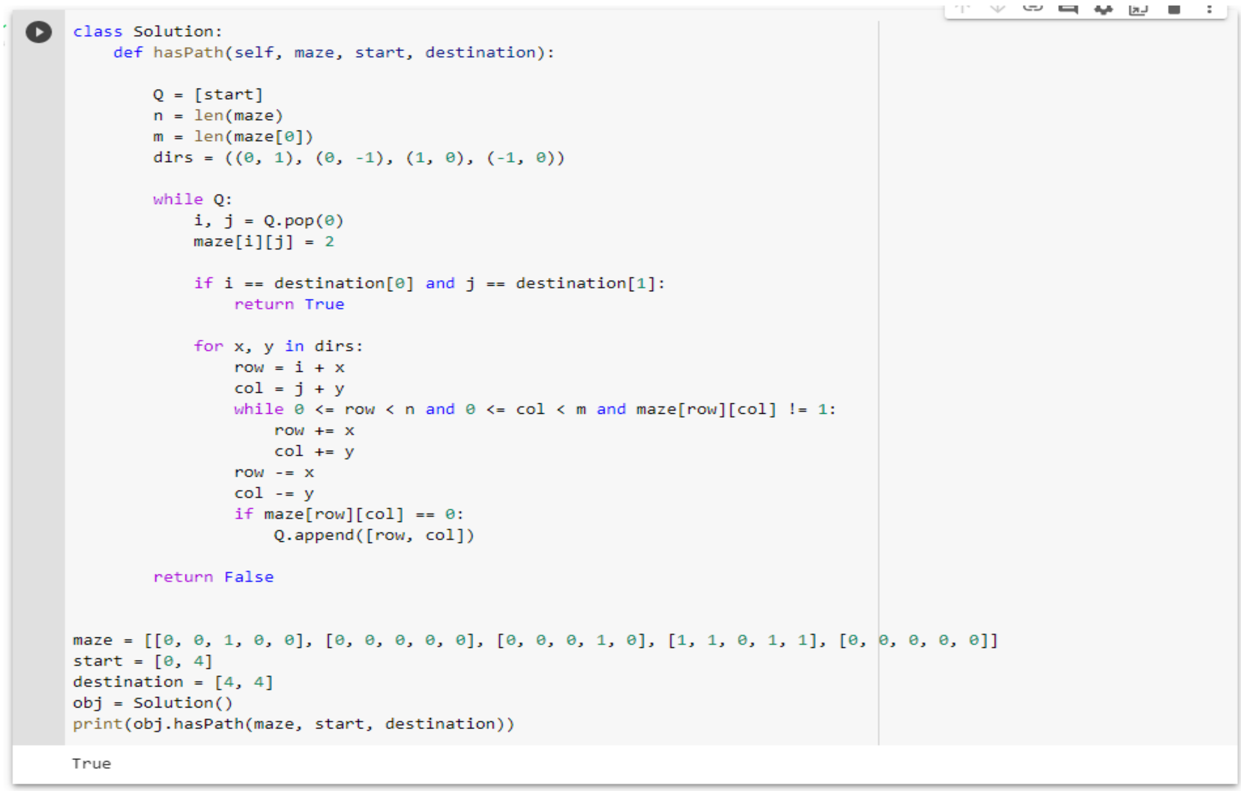
## 2.2 Implement python

```python
class Solution:
    def hasPath(self, maze, start, destination):
        Q = [start]
        n = len(maze)
        m = len(maze[0])
        dirs = ((0, 1), (0, -1), (1, 0), (-1, 0))
        while Q:
            i, j = Q.pop(0)
            maze[i][j] = 2

            if i == destination[0] and j == destination[1]:
                return True

            for x, y in dirs:
                row = i + x
                col = j + y
                while 0 <= row < n and 0 <= col < m and maze[row][col] != 1:
                    row += x
                    col += y
                row -= x
                col -= y
                if maze[row][col] == 0:
                    Q.append([row, col])
        return False
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))
```

The output checks if there is a path from the start to the destination and it return True if yes, and False if there is no pas as follows



```
class Solution:
    def hasPath(self, maze, start, destination):

        Q = [start]
        n = len(maze)
        m = len(maze[0])
        dirs = ((0, 1), (0, -1), (1, 0), (-1, 0))

        while Q:
            i, j = Q.pop(0)
            maze[i][j] = 2

            if i == destination[0] and j == destination[1]:
                return True

            for x, y in dirs:
                row = i + x
                col = j + y
                while 0 <= row < n and 0 <= col < m and maze[row][col] != 1:
                    row += x
                    col += y
                row -= x
                col -= y
                if maze[row][col] == 0:
                    Q.append([row, col])

        return False


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))
```

```
True
```