

Step 1: Study 490. The Maze - (local copy)

```

i Python3 | • Auto
1 from collections import deque
2
3 class Solution:
4     def hasPath(self, maze: List[List[int]], start: List[int], destination: List[int]) -> bool:
5         if not maze or not start or not destination:
6             # bad input
7             return False
8         if start == destination:
9             # input start and destination were the same
10            return True
11
12        q = deque([start])
13        # using a deque is important when used as a queue
14        # stack here would be DFS
15        visited = set()
16        # a set will provide constant time access, we will never have duplicates
17        directions_to_go = [(0, 1), (0, -1), (1, 0), (-1, 0)]
18        # we can always roll north, south, east, or west
19
20        while q:
21            current = q.popleft()
22            # first in first out (swap to pop here with stack for DFS)
23            if current[0] == destination[0] and current[1] == destination[1]:
24                return True
25            for direction in directions_to_go:
26                # move in a direction
27                x = current[0] + direction[0]
28                y = current[1] + direction[1]
29                while 0 <= x < len(maze) and 0 <= y < len(maze[0]) and maze[x][y] == 0:
30                    # keep moving until ONE PAST where you can't move (roll) anymore
31                    x += direction[0]
32                    y += direction[1]
33                # roll back one so that you're actually where you should be
34                rolled_to_x = x - direction[0]
35                rolled_to_y = y - direction[1]
36                if (rolled_to_x, rolled_to_y) not in visited:
37                    visited.add((rolled_to_x, rolled_to_y))
38                    # add this position to be searched from as well
39                    q.append((rolled_to_x, rolled_to_y))
40            # if you're here no solution was found and everything has been visited
41            return False
42

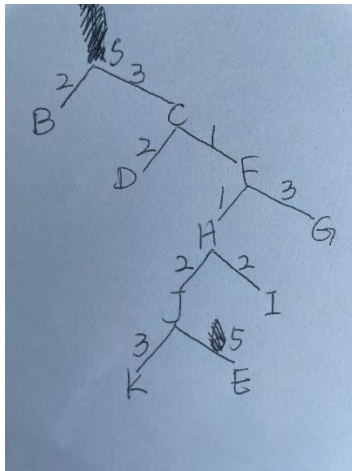
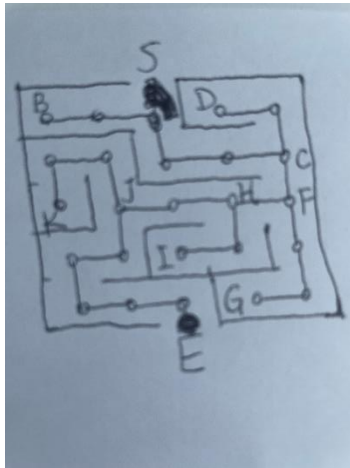
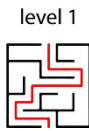
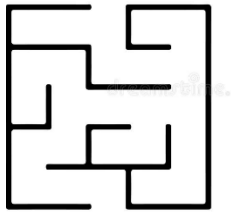
```

Step 2: Manual process to demonstrate concepts.

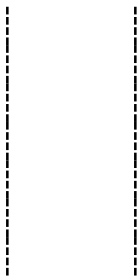
Robot	Clear Route (Street, Highway)	Unclear Route (Hotel, Hospital)
Without Wheel (Legged Robot)	Step 1.1: Tree <ul style="list-style-type: none"> Following the examples shown on Depth-First Traversal to manually solve the problem <ul style="list-style-type: none"> Maze example 	
With Wheel (Self-driving Car)		Step 1.2: Matrix <ul style="list-style-type: none"> Following the examples shown on Depth-First Traversal to manually solve the problem <ul style="list-style-type: none"> Maze example -- assuming the ball can go through the empty spaces by rolling.

Step 1.1: Tree

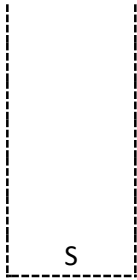
level 1



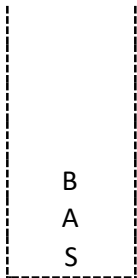
1. initialize the stack



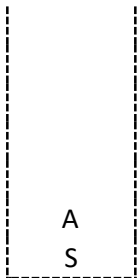
Stack
2. visited S



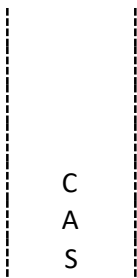
Stack
3. visited B



Stack
4. pop B



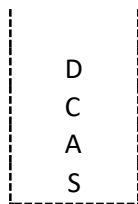
Stack
5. visited C



Stack

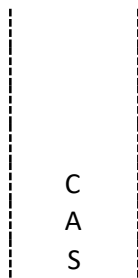
6. visited D





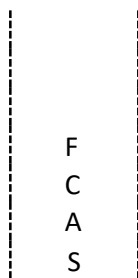
Stack

7. pop D



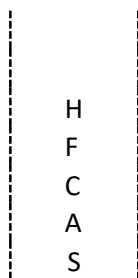
Stack

8. visited F



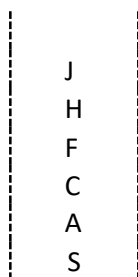
Stack

9. visited H



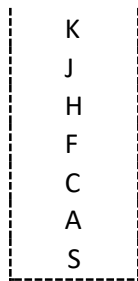
Stack

10. visited J



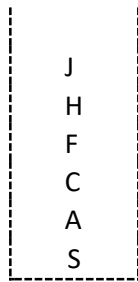
Stack

11. visited K



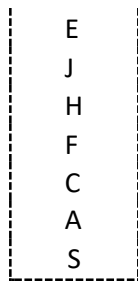
Stack

12. pop K



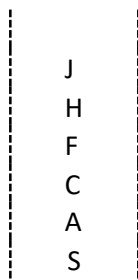
Stack

13. visited E



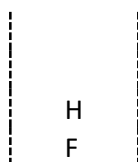
Stack

14. pop E



Stack

15. pop J



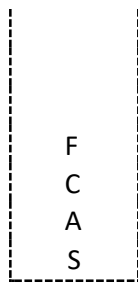
C
A
S
Stack
16. visited I

I
H
F
C
A
S
Stack
17. pop I

H
F
C
A
S
Stack
18. pop H

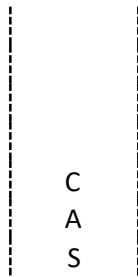
F
C
A
S
Stack
19. visited G

G
F
C
A
S
Stack
20. pop G



Stack

21. pop F



Stack

22. pop C



Stack

23. pop S; the stack is empty, stop.



Stack


Step 1.2: Matrix

To solve the LeetCode problem "490. The Maze" using a Depth-First Traversal approach in Python, you can follow these steps:

1. Implement a Depth-First Search (DFS) function that explores the maze from a given starting position. It will recursively move in different directions until it reaches a wall or an already visited cell.
2. Use a helper function to perform DFS on all possible directions (up, down, left, right) from a given cell.
3. Mark visited cells and keep track of visited positions to avoid infinite loops.
4. Check if the destination cell is reached.

Here's a possible implementation:

python

 Copy code

```
def hasPath(maze, start, destination):
    rows, cols = len(maze), len(maze[0])
    visited = [[False] * cols for _ in range(rows)]

    def dfs(row, col):
        if visited[row][col]:
            return False
        visited[row][col] = True

        if [row, col] == destination:
            return True
```

```

# Try moving in all possible directions
directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
for dr, dc in directions:
    r, c = row, col
    while 0 <= r + dr < rows and 0 <= c + dc < cols and maze[r + dr]
        r += dr
        c += dc
        if dfs(r, c):
            return True
    return False

return dfs(start[0], start[1])

# Test data
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]]

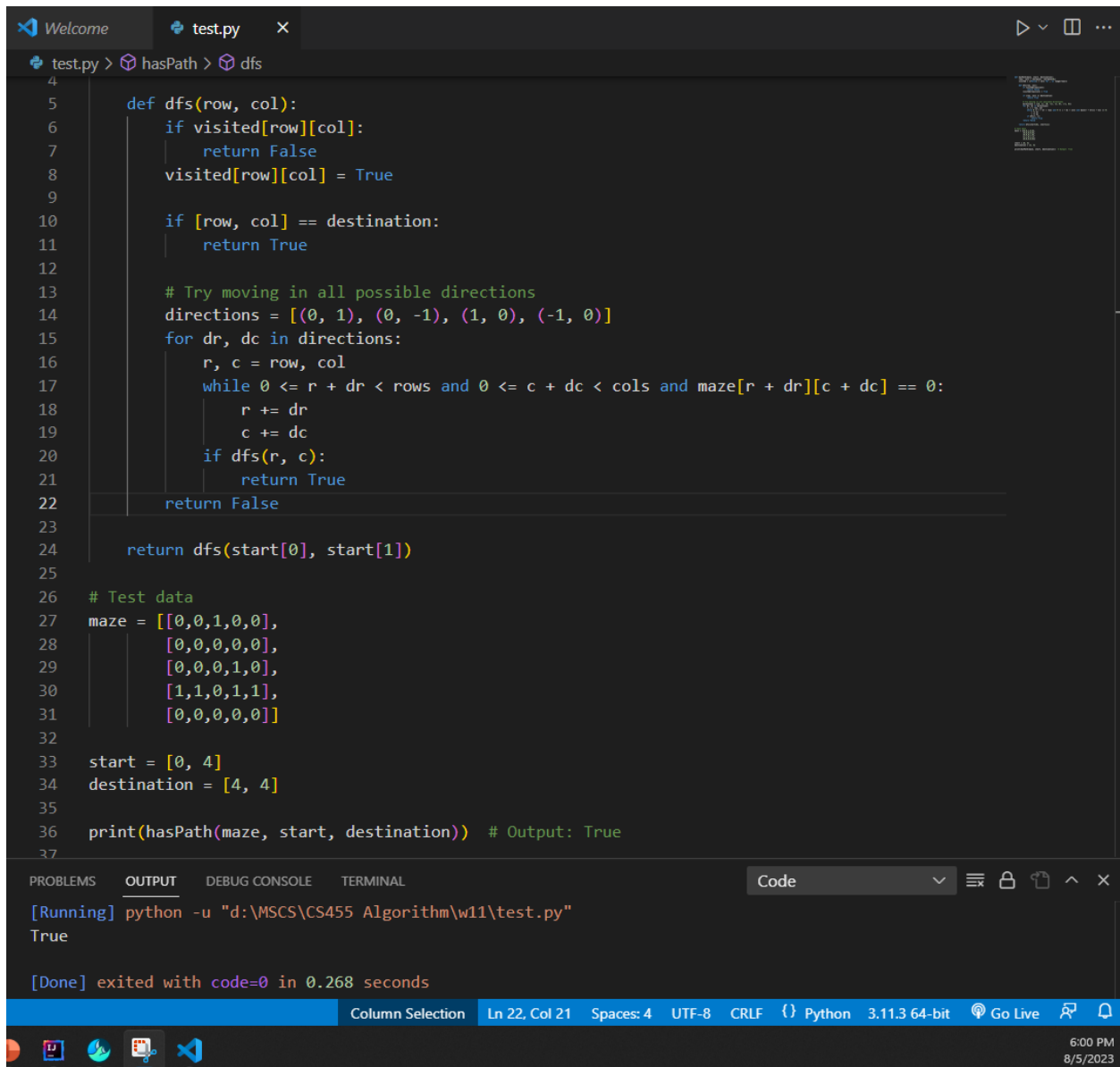
start = [0, 4]
destination = [4, 4]

```

```
print(hasPath(maze, start, destination)) # Output: True
```

This implementation uses Depth-First Traversal to explore the maze and check if a path exists from the start to the destination. The `dfs()` function recursively explores each direction until the destination is reached or all paths are exhausted. The `visited` matrix is used to avoid revisiting cells.

Step 4: Implement the Python code created by ChatGPT



```
test.py > hasPath > dfs
4
5     def dfs(row, col):
6         if visited[row][col]:
7             return False
8         visited[row][col] = True
9
10        if [row, col] == destination:
11            return True
12
13        # Try moving in all possible directions
14        directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
15        for dr, dc in directions:
16            r, c = row, col
17            while 0 <= r + dr < rows and 0 <= c + dc < cols and maze[r + dr][c + dc] == 0:
18                r += dr
19                c += dc
20                if dfs(r, c):
21                    return True
22        return False
23
24    return dfs(start[0], start[1])
25
26 # Test data
27 maze = [[0,0,1,0,0],
28         [0,0,0,0,0],
29         [0,0,0,1,0],
30         [1,1,0,1,1],
31         [0,0,0,0,0]]
32
33 start = [0, 4]
34 destination = [4, 4]
35
36 print(hasPath(maze, start, destination)) # Output: True
37
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Code

[Running] python -u "d:\MSCS\CS455 Algorithm\w11\test.py"

True

[Done] exited with code=0 in 0.268 seconds

Column Selection Ln 22, Col 21 Spaces: 4 UTF-8 CRLF Python 3.11.3 64-bit Go Live

6:00 PM 8/5/2023

Step 5: Test the Python code with all the test cases provided by 490. The Maze - (local copy)

test.py > hasPath > dfs

```
4
5     def dfs(row, col):
6         if visited[row][col]:
7             return False
8         visited[row][col] = True
9
10        if [row, col] == destination:
11            return True
12
13        # Try moving in all possible directions
14        directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
15        for dr, dc in directions:
16            r, c = row, col
17            while 0 <= r + dr < rows and 0 <= c + dc < cols and maze[r + dr][c + dc] == 0:
18                r += dr
19                c += dc
20                if dfs(r, c):
21                    return True
22        return False
23
24    return dfs(start[0], start[1])
25
26    # Test data
27    maze = [[0,0,1,0,0],
28            [0,0,0,0,0],
29            [0,0,0,1,0],
30            [1,1,0,1,1],
31            [0,0,0,0,0]]
32
33    start = [0, 4]
34    destination = [4, 4]
35
36    print(hasPath(maze, start, destination)) # Output: True
37
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Code

[Running] python -u "d:\MSCS\CS455 Algorithm\w11\test.py"

True

[Done] exited with code=0 in 0.268 seconds

Column Selection Ln 22, Col 21 Spaces: 4 UTF-8 CRLF Python 3.11.3 64-bit Go Live 6:00 PM 8/5/2023

Welcome

test.py

test.py > ...

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

```
def dfs(row, col):
    if visited[row][col]:
        return False
    visited[row][col] = True

    if [row, col] == destination:
        return True

    # Try moving in all possible directions
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
    for dr, dc in directions:
        r, c = row, col
        while 0 <= r + dr < rows and 0 <= c + dc < cols and maze[r + dr][c + dc] == 0:
            r += dr
            c += dc
            if dfs(r, c):
                return True
    return False

return dfs(start[0], start[1])

# Test data
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 = [3, 2]

print(hasPath(maze, start, destination)) # Output: True
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

[Running]

python -u "d:\MSCS\CS455 Algorithm\w11\test.py"

False

[Done]

exited with code=0 in 0.241 seconds

Column Selection

Ln 34, Col 20

Spaces: 4

UTF-8

CRLF

() Python

3.11.3 64-bit

Go Live

6:02 PM

8/5/2023

```
test.py > ...
1 def hasPath(maze, start, destination):
2     rows, cols = len(maze), len(maze[0])
3     visited = [[False] * cols for _ in range(rows)]
4
5     def dfs(row, col):
6         if visited[row][col]:
7             return False
8         visited[row][col] = True
9
10        if [row, col] == destination:
11            return True
12
13        # Try moving in all possible directions
14        directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
15        for dr, dc in directions:
16            r, c = row, col
17            while 0 <= r + dr < rows and 0 <= c + dc < cols and maze[r + dr][c + dc] == 0:
18                r += dr
19                c += dc
20                if dfs(r, c):
21                    return True
22        return False
23
24    return dfs(start[0], start[1])
25
26 # Test data
27 maze = [[0,0,0,0,0],
28         [1,1,0,0,1],
29         [0,0,0,0,0],
30         [0,1,0,0,1],
31         [0,1,0,0,0]]
32
33 start = [4, 3]
34 destination = [0, 1]
35
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

[Running] python -u "d:\MSCS\CS455 Algorithm\w11\test.py"

False

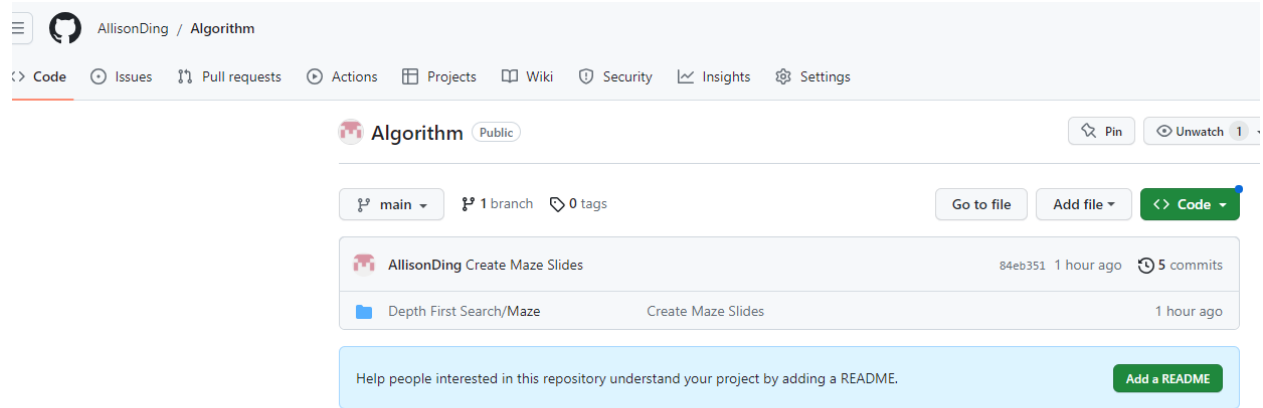
Column Selection Ln 34, Col 20 Spaces: 4 UTF-8 CRLF Python 3.11.3 64-bit Go Live

6:04 PM 8/5/2023

Step 6: Update your portfolio about the Maze project

- Please use this structure to describe the project

```
Algorithm
  Depthe First Search
    Maze
```



Step 7: Submit the URL of your GitHub webpage as the homework answer.

<https://github.com/AllisonDing/Algorithm/tree/main/Depth%20First%20Search/Maze>