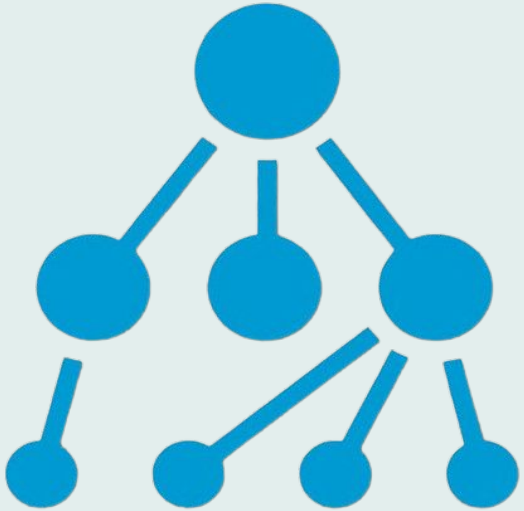
The Maze
Breadth-First and Depth-First Traversal Implementations

TABLE OF CONTENTS

01. INTRODUCTION

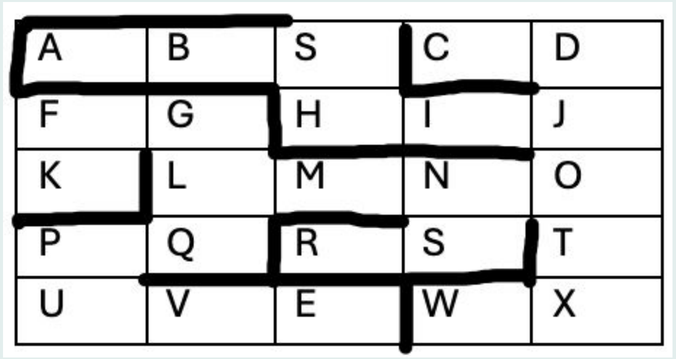
This project focuses on solving the maze traversal problem using both Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms. The maze represents a grid where certain paths are blocked, and the objective is to find a path from the start point to the destination point while navigating through the open paths.

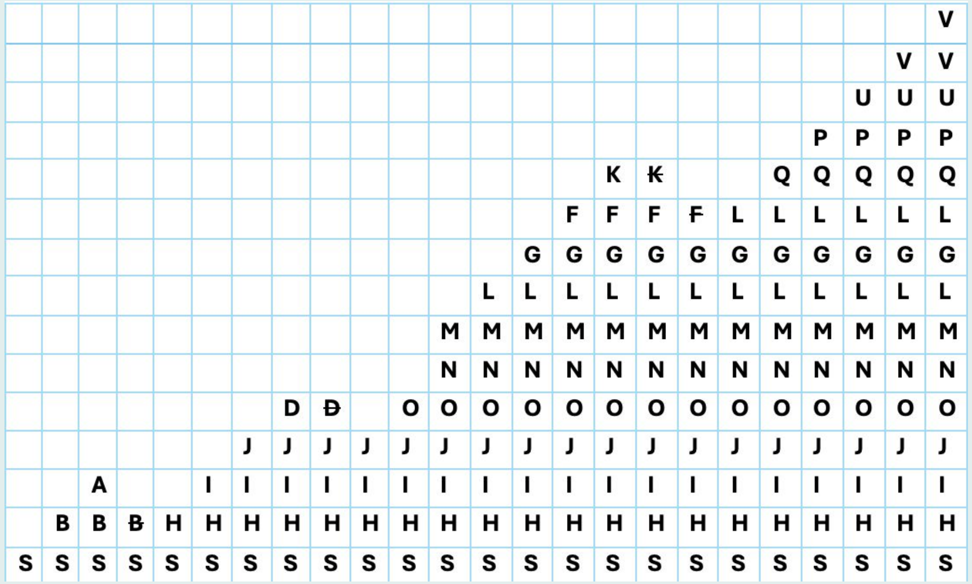


The process involves a manual demonstration of both Breadth-First and Depth-First Traversal concepts to solve the maze problem. Subsequently, Python solutions are implemented using both algorithms, and they are tested against various test cases provided by LeetCode for the "490. The Maze" problem.

02. Depth-First Traversal (without wheel - legged robot)

- Right, Left, Up, Bottom
- The ball can only move one cell at a time

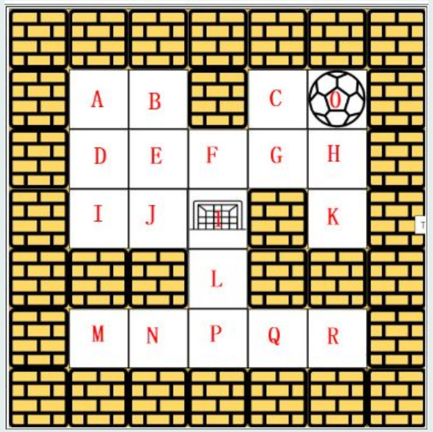


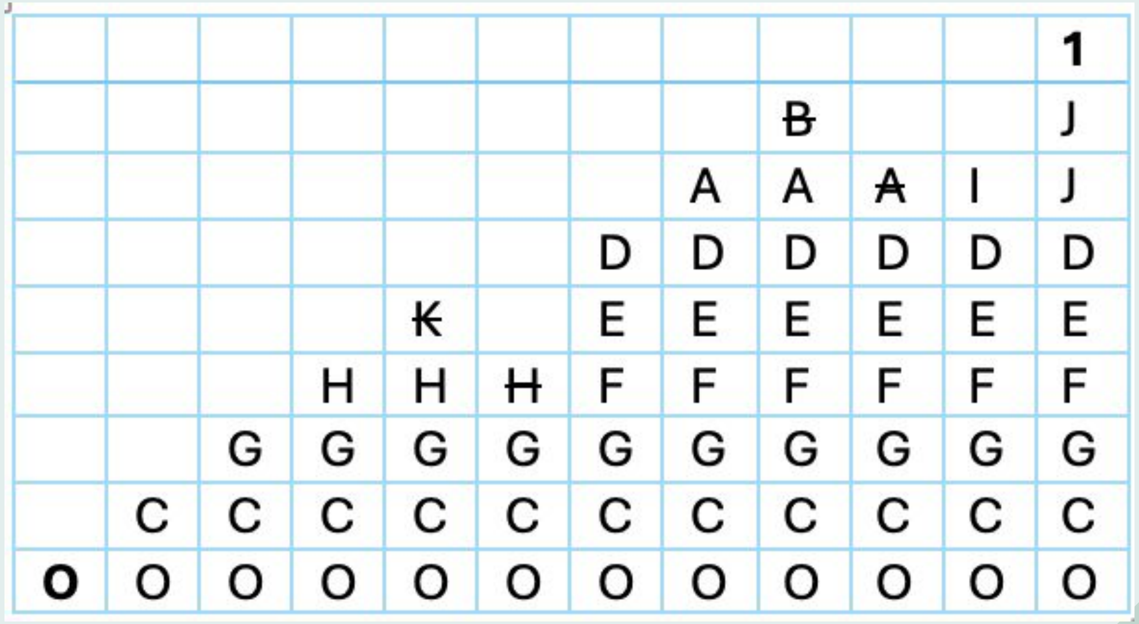


																							V
																						V	V
																					U	U	U
																				P	P	P	P
															K	~~K~~			Q	Q	Q	Q	Q
														F	F	F	~~F~~	L	L	L	L	L	L
													G	G	G	G	G	G	G	G	G	G	G
												L	L	L	L	L	L	L	L	L	L	L	L
											M	M	M	M	M	M	M	M	M	M	M	M	M
											N	N	N	N	N	N	N	N	N	N	N	N	N
							D	~~D~~		O	O	O	O	O	O	O	O	O	O	O	O	O	O
						J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J	J
		A			I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
	B	B	~~B~~	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S

02. Depth-First Traversal (with wheel - self-driving car)

- Right, Left, Up, Bottom
- The ball can go through the empty spaces by rolling R-L-U-D, but it won't stop rolling until hitting a wall

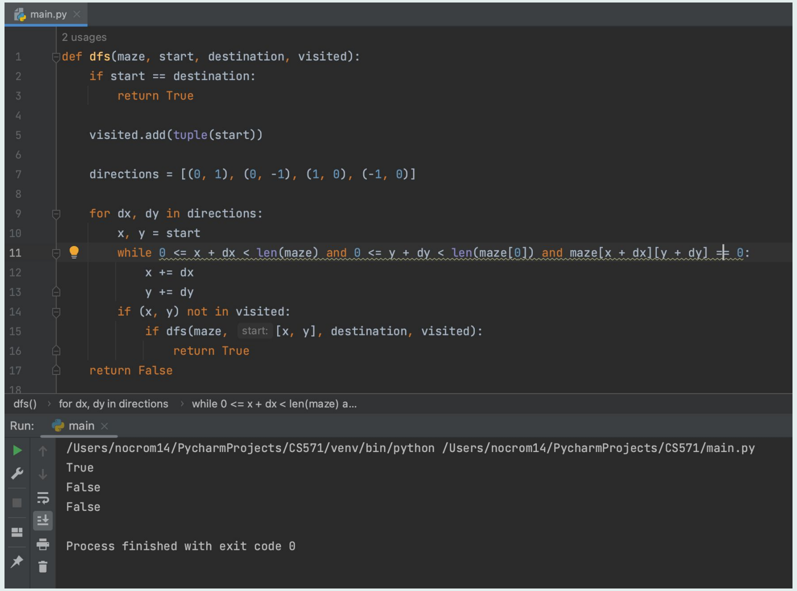




											1
								~~B~~			J
							A	A	~~A~~	I	J
						D	D	D	D	D	D
				~~K~~		E	E	E	E	E	E
			H	H	~~H~~	F	F	F	F	F	F
		G	G	G	G	G	G	G	G	G	G
	C	C	C	C	C	C	C	C	C	C	C
O	O	O	O	O	O	O	O	O	O	O	O

02. Depth-First Traversal (Implementation - Python Code)

```python
def dfs(maze, start, destination, visited):
    if start == destination:
        return True

    visited.add(tuple(start))

    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    for dx, dy in directions:
        x, y = start
        while 0 <= x + dx < len(maze) and 0 <= y + dy < len(maze[0]) and maze[x + dx][y + dy] == 0:
            x += dx
            y += dy
        if (x, y) not in visited:
            if dfs(maze, [x, y], destination, visited):
                return True
    return False


def hasPath(maze, start, destination):
    visited = set()
    return dfs(maze, start, destination, visited)


# Test cases
maze1 = [[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]]
start1 = [0, 4]
destination1 = [4, 4]
print(hasPath(maze1, start1, destination1))  # Output: True

maze2 = [[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]]
start2 = [0, 4]
destination2 = [3, 2]
print(hasPath(maze2, start2, destination2))  # Output: False

maze3 = [[0, 0, 0, 0, 0], [1, 1, 0, 0, 1], [0, 0, 0, 0, 0], [0, 1, 0, 0, 1], [0, 1, 0, 0, 0]]
start3 = [4, 3]
destination3 = [0, 1]
print(hasPath(maze3, start3, destination3))  # Output: False
```



main.py

2 usages

```python
def dfs(maze, start, destination, visited):
    if start == destination:
        return True

    visited.add(tuple(start))

    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    for dx, dy in directions:
        x, y = start
        while 0 <= x + dx < len(maze) and 0 <= y + dy < len(maze[0]) and maze[x + dx][y + dy] == 0:
            x += dx
            y += dy
        if (x, y) not in visited:
            if dfs(maze, start: [x, y], destination, visited):
                return True
    return False
```

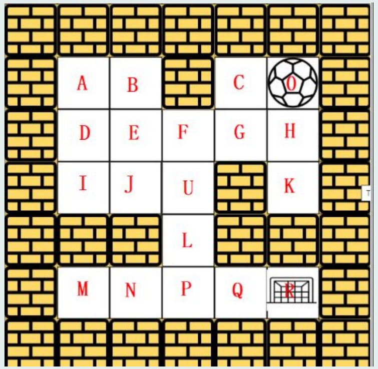
dfs() › for dx, dy in directions › while 0 <= x + dx < len(maze) a...

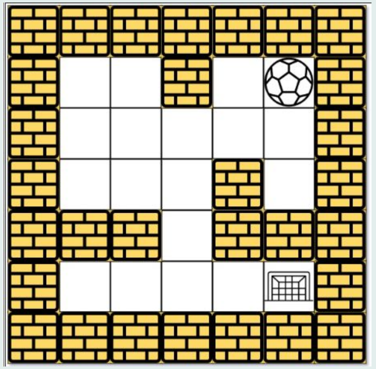
Run: main

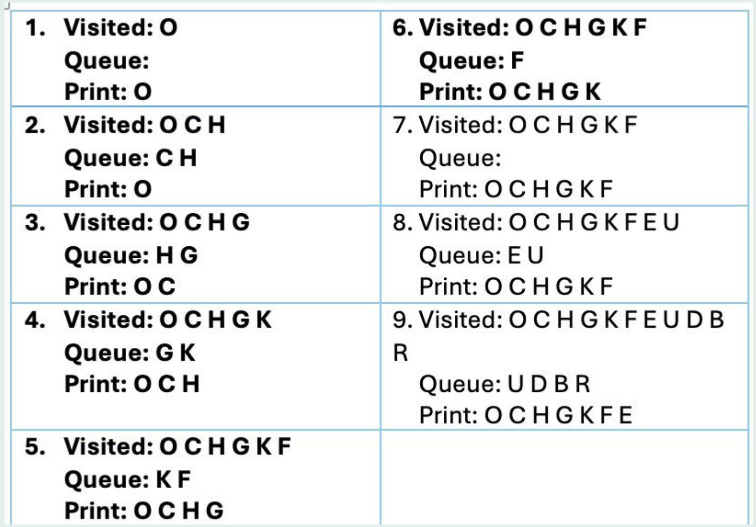
```
/Users/nocrom14/PycharmProjects/CS571/venv/bin/python /Users/nocrom14/PycharmProjects/CS571/main.py
True
False
False

Process finished with exit code 0
```

03. Breadth-First Traversal (without wheel - legged robot)

- Right, Left, Up, Bottom
- The ball can only move one cell at a time



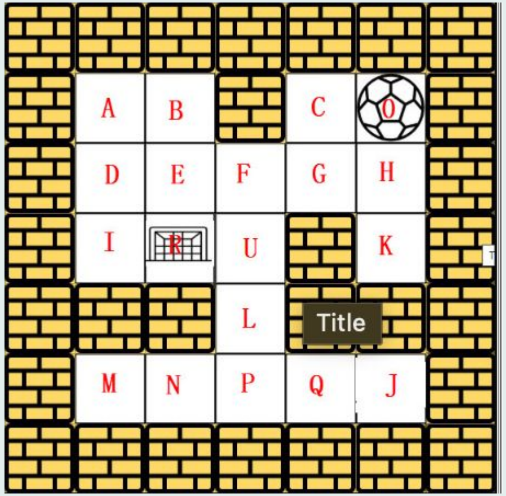


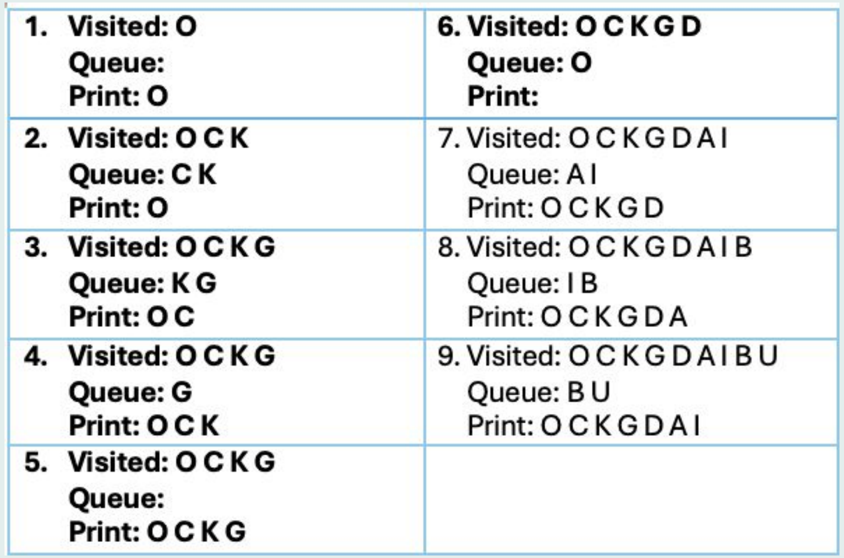


1. Visited: O **Queue:** **Print: O**	**6. Visited: O C H G K F** **Queue: F** **Print: O C H G K**
2. Visited: O C H **Queue: C H** **Print: O**	7. Visited: O C H G K F Queue: Print: O C H G K F
3. Visited: O C H G **Queue: H G** **Print: O C**	8. Visited: O C H G K F E U Queue: E U Print: O C H G K F
4. Visited: O C H G K **Queue: G K** **Print: O C H**	9. Visited: O C H G K F E U D B R Queue: U D B R Print: O C H G K F E
5. Visited: O C H G K F **Queue: K F** **Print: O C H G**	

03. Breadth-First Traversal (with wheel - self-driving car)

- Right, Left, Up, Bottom
- The ball can go through the empty spaces by rolling R-L-U-D, but it won't stop rolling until hitting a wall

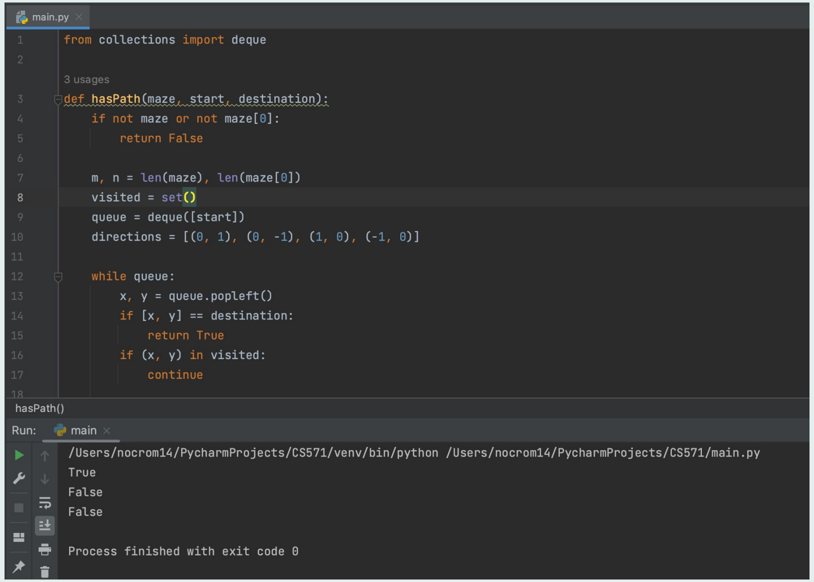




1. Visited: O **Queue:** **Print: O**	**6. Visited: O C K G D** **Queue: O** **Print:**
2. Visited: O C K **Queue: C K** **Print: O**	7. Visited: O C K G D A I Queue: A I Print: O C K G D
3. Visited: O C K G **Queue: K G** **Print: O C**	8. Visited: O C K G D A I B Queue: I B Print: O C K G D A
4. Visited: O C K G **Queue: G** **Print: O C K**	9. Visited: O C K G D A I B U Queue: B U Print: O C K G D A I
5. Visited: O C K G **Queue:** **Print: O C K G**	

03. Breadth-First Traversal (Implementation - Python Code)

```
from collections import deque

def hasPath(maze, start, destination):
    if not maze or not maze[0]:
        return False

    m, n = len(maze), len(maze[0])
    visited = set()
    queue = deque([start])
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    while queue:
        x, y = queue.popleft()
        if [x, y] == destination:
            return True
        if (x, y) in visited:
            continue

        visited.add((x, y))
        for dx, dy in directions:
            newX, newY = x, y
            while 0 <= newX + dx < m and 0 <= newY + dy < n and maze[newX + dx][newY + dy]
== 0:
                newX += dx
                newY += dy
            if (newX, newY) not in visited:
                queue.append((newX, newY))

    return False

# Test cases
maze1 = [[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]]
start1 = [0,4]
destination1 = [4,4]

maze2 = [[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]]
start2 = [0,4]
destination2 = [3,2]

maze3 = [[0,0,0,0,0], [1,1,0,0,1], [0,0,0,0,0], [0,1,0,0,1], [0,1,0,0,0]]
start3 = [4,3]
destination3 = [0,1]

print(hasPath(maze1, start1, destination1))  # Output: True
print(hasPath(maze2, start2, destination2))  # Output: False
print(hasPath(maze3, start3, destination3))  # Output: False
```



```
from collections import deque

def hasPath(maze, start, destination):
    if not maze or not maze[0]:
        return False

    m, n = len(maze), len(maze[0])
    visited = set()
    queue = deque([start])
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    while queue:
        x, y = queue.popleft()
        if [x, y] == destination:
            return True
        if (x, y) in visited:
            continue
```

```
/Users/nocrom14/PycharmProjects/CS571/venv/bin/python /Users/nocrom14/PycharmProjects/CS571/main.py
True
False
False

Process finished with exit code 0
```

04. Key Takeaways

- Breadth-First Traversal (BFS) ensures finding the shortest path in maze traversal problems.
- It systematically explores all possible paths in a level-by-level manner.
- This approach is efficient and optimal, particularly in scenarios where finding the shortest path is crucial.
- Unlike BFS, DFS explores as far as possible along each branch before backtracking.
- DFS may not guarantee finding the shortest path, but it can be more memory efficient compared to BFS.
- DFS is well-suited for scenarios where finding any valid path is sufficient, rather than necessarily the shortest one.
- Understanding the differences and trade-offs between DFS and BFS is essential for effectively solving maze problems and other traversal challenges.

Reference

GfG. (2019, May 21). *Difference between BFS and DFS*. GeeksforGeeks; GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-bfs-and-dfs/

Bari, A. (2018). 5.1 Graph Traversals - BFS & DFS -Breadth First Search and Depth First Search [YouTube Video]. In *YouTube*. https://www.youtube.com/watch?v=pcKY4hjDrxk&ab_channel=AbdulBari

GitHub Link

https://github.com/FanielS/Algorithms/Maze-Breadth-First-and-Depth-First-Traversal-Imple

mentations