Project: "490. The Maze" -LC - Breadth-First Traversal

NAME: FEVEN BELAY ARAYA

ID: 20027

TABLE OF CONTENT

- 1. INTRODUCTION
- 2. DESIGN
- 3. IMPLEMENTATION
- 4. TEST
- 5. ENHANCEMENT IDEAS
- 6. CONCLUSION

1. INTRODUCTION

- •Breadth-First Search (BFS) is an algorithm for traversing or searching tree or graph data structures.
- •It starts at a selected node (often at the root in the case of trees, or some arbitrary node for graphs).
- •It explores all of the neighbor nodes at the present depth prior to moving on to nodes at the next depth level.

2. DESIGN

2.1. Without Wheel (Legged Robot)

X/A P	1	A			
A B		× ×			
1/ /	I IXA	*			
I R	UK	*			
XXX	LIVA	*			
MA	P 0 5				
KVVXX	MANAY				
Sept Vizile	1:0				
STATE OF THE OWNER, WHEN PERSON AND ADDRESS OF THE OWNER, WHEN PERSON					
	inst Traversal -	I leaged rol	orts haden Th	O Hotel -	man unhooled
Breadth - F	step 200	step 3	step 4	step 5	step 6
Breadth - F	Step 2	step 3	step 4	step 5	step 6
Breanth - F Steps visited 0	Vicited a OCH	Step 3 V + O CH G	V+ OCHG+	V+OCHGF	Stop 6
Breadth - F Steps Visted O Queue	Step 2 OCH OURUE -) CH	V+0 CH & Q+ H &	V+OCHGF Q+GK	V+0CHGF 0+KF	
Breadth - F Steps visited 0 Queue- Print - 0	Step 2 VICTIED + OCH OURUE + CH POINT + OCCH	V → 0 CH & D → 0 C P + 0 C	V+OCHGE Q+GK P+OCHGE	V+0CHGF 0+KF	
Breadth - F Steps Visted 0 Oueve- Print - 0 Step +	Step 2 VISTTECH OCH OWNED CH POINT DOON	V → 0 CH & Q → H & P → 0 C	V+OCHGF Q+GK P+OCHG	V+OCHGF O→KF P→OCHG	
Breadth - F Steps Visited O Oueve- Print - O Step + V+ O C. H. G. F. E	Step 2 VISTING TO CH OLIVET CH POINT TO CH STEP 8	V+OCHE Q+HE P+OC	V+ OCHGF Q + GK P + OCHG	V+OCHGF O+KF P+OCHG	
Breadth - F Steps Visited O Oueve- Print - O Step + V+ O C. H. G. F. E	Step 2 VISTTECH OCH OWNED CH POINT DOON	V+OCHE Q+HE P+OC	V+ OCHGF Q + GK P + OCHG	V+OCHGF O+KF P+OCHG	

2.2. With Wheel (Self-driving Car)

8FT-) wheeled								
1) VISTARD - O QUEUR - PATINT - O	2. VISTARD-OCK Overe - CK Print- O	3. Vinted-OCK & Queue - KG AnAI-DC	4. Visited - OCKG overe - G Pant - OCK	5. V + OCKG Q -> P + OCKG				
6. V + OC K & P Q + D P -	7) V+OCKGDAT Q+AI P+OCKGD	8)V+OCKGDAIB Ø+ JB P+OCKGDA	9) V+ OCKGDA. Q+ BU P+ OCKGD Add R+0+ T+ us alread	A I he overe ou				

3. IMPLEMENTATION

```
Users > fevenbelay > Desktop > FevDesktop > SFBU-Semester2 > Practical algorithm > 🏺 BFS-maze.py > ...
       from collections import deque
       class Solution:
           def hasPath(self, maze, start, destination):
               m, n = len(maze), len(maze[0])
               visited = set()
               queue = deque([(start[0], start[1])])
               while queue:
                   x, y = queue.popleft()
                   if (x, y) == (destination[0], destination[1]):
                       return True
                   if (x, y) in visited:
                       continue
                   visited.add((x, y))
                   for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
                       nx, ny = x + dx, y + dy
                       while 0 \le nx \le m and 0 \le ny \le n and maze [nx][ny] == 0:
                            nx += dx
                           ny += dy
                       nx -= dx
                       ny -= dy
                       if (nx, ny) not in visited:
                           queue.append((nx, ny))
               return False
```

4. TEST

```
# Test the function
       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]
       sol = Solution()
        print(sol.hasPath(maze, start, destination))
       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]
       sol = Solution()
       print(sol.hasPath(maze, start, destination))
  42
       maze = [[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]]
       start = [4,3]
       destination = [0,1]
       sol = Solution()
       print(sol.hasPath(maze, start, destination))
 PROBLEMS
             DEBUG CONSOLE
                             TERMINAL
 /usr/bin/python3 "/Users/fevenbelay/Desktop/FevDesktop/SFBU-Semester2/Practical algorithm/BFS-maze.py"
🏮 (base) fevenbelay@Fevens-MacBook-Air ~ % /usr/bin/python3 "/Users/fevenbelay/Desktop/FevDesktop/SFBU-Semester2/Practical algorithm/BFS-maze.py"
 True
 False
 False
o (base) fevenbelay@Fevens-MacBook-Air ~ %
```

5. ENHANCEMENT IDEAS

- 1. Space-Efficient Data Structures: Using more space-efficient data structures for the queue and visited set, such as compressed bit vectors, can significantly reduce the memory footprint of BFS.
- 2. Parallel BFS: BFS can be parallelized by processing multiple nodes at the same level concurrently, using parallel computing resources.
- 3. Sparse BFS: For sparse graphs, an adjacency list can be more efficient than an adjacency matrix, reducing the time complexity of checking for neighboring nodes from O(V) to O(deg(v)), where V is the number of vertices and deg(v) is the degree of a vertex.
- 4. Dynamic BFS: For graphs that change over time (dynamic graphs), maintaining BFS trees and updating them incrementally can be more efficient than re-computing the BFS from scratch after every change.
- 5. External BFS: For extremely large graphs that do not fit into memory, employing external memory algorithms can allow BFS to run on graphs stored on disk.

6. CONCLUSION

- This approach is ideal for mazes because it can find the shortest path without getting trapped in dead ends or visiting any part of the maze unnecessarily.
- Each position is visited at most once, ensuring the process is efficient and terminates once it either finds the destination or exhausts all possible paths.