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

BY

JOSHUA T. AKOTO (19768)

Introduction

Given a maze with walls and empty spaces, a ball starting at cell '0', and a goal at cell 'J', determine if the ball can reach the goal by moving horizontally or vertically until hitting a wall.

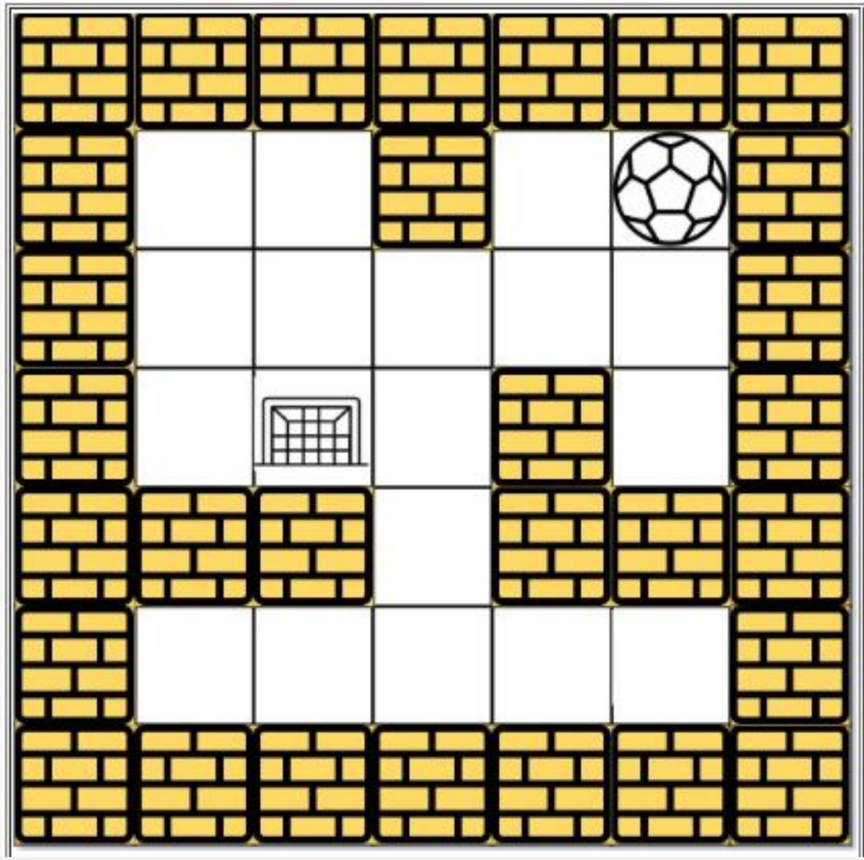
QUESTION

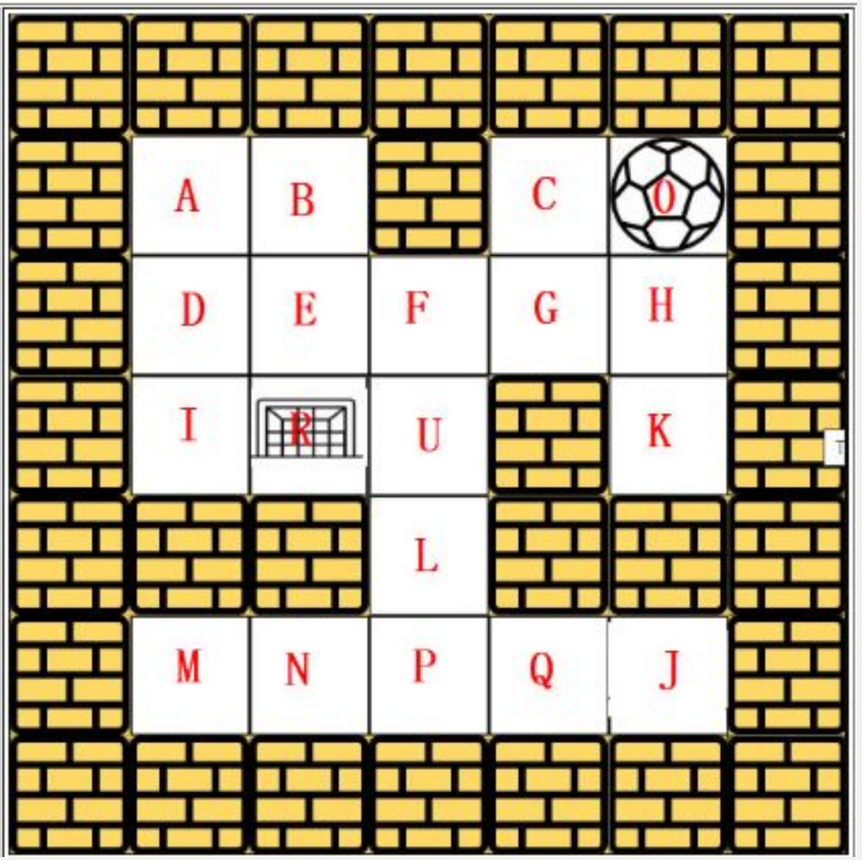
40. Project: "490. The Maze" - LC - Breadth-First Traversal .

- 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. Th
          - 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
```

THE MAZE

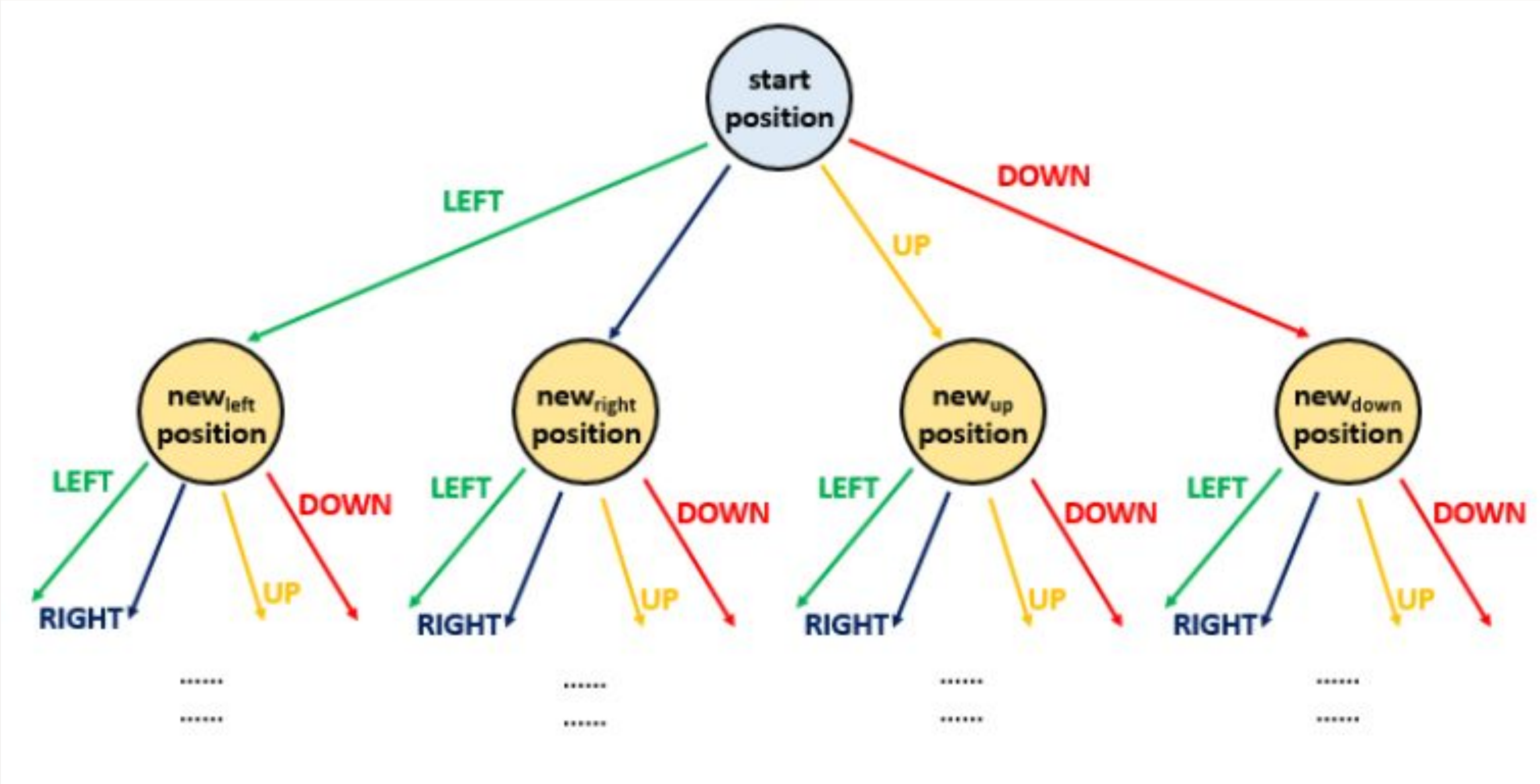




Approach:

- Breadth-First Traversal (BFS) and Depth-First Traversal (DFS):
- Utilizing BFS and DFS algorithms to explore the maze.
- BFS: Systematic exploration in breadth-first manner.
- DFS: Systematic exploration in depth-first manner.

APPROACH


start position
LEFT
DOWN
UP
new$_{left}$ position
new$_{right}$ position
new$_{up}$ position
new$_{down}$ position
LEFT
DOWN
UP
RIGHT
LEFT
DOWN
UP
RIGHT
LEFT
DOWN
UP
RIGHT
LEFT
DOWN
UP
RIGHT
....
....
....
....
....
....
....
....

BFS Manual Solution Process:

Step 1: Initialization

Start at cell '0'.

Initialize an empty queue for BFS traversal.

Mark starting cell as visited.

BFS Manual Solution Process:

Step 2: Exploration

Explore all accessible neighboring cells (left, right, up, down).

Enqueue neighboring cells that are not walls and have not been visited.

Step 3: Continuation

Dequeue the next cell from the queue.

Mark it as visited.

Repeat the exploration process from this new cell.

BFS Manual Solution Process:

Step 4: Reaching the Goal ('J')

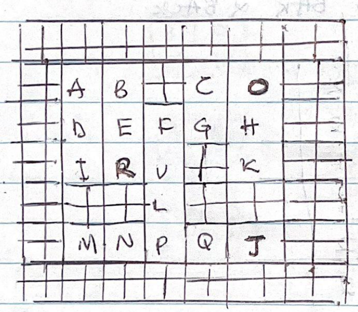
Continue exploring and dequeuing cells until reaching cell 'J' or the queue becomes empty.

If 'J' is reached during BFS traversal, return "true". Otherwise, return "false".

ANSWER

The Maze

Breadth-First Traversal



- Visited O
  Queue:
- visited O
  Queue: O
- visited O C K
  Queue: CK
- visited O C K
  Queue: K
- visited O C K G
  Queue: K G
- visited O C K G
  Queue: G
- visited O C K G
  Queue:
- visited O C K G D
  Queue: D
- visited O C K G D
  Queue:
- visited O C K G D A I
  Queue: A I
- visited O C K G D A I
  Queue: I
- visited O C K G D A I
  Queue: I B
- visited O C K G D A I B
  Queue: B
- visited O C K G D A I B U
  Queue: B U
- visited O C K G D A I B U
  Queue: U
- visited O C K G D A I B U
  Queue:
- visited: O C K G D A I B U P
  Queue: P
- visited: O C K G D A I B U P J
  Queue: J

DFS Manual Solution Process:

Step 1: Initialization

Start at cell '0'.

Initialize an empty stack for DFS traversal.

Mark starting cell as visited

Step 2: Exploration

Explore all accessible neighboring cells (left, right, up, down).

Push neighboring cells that are not walls and have not been visited onto the stack.

DFS Manual Solution Process:

Step 3: Continuation

Pop the next cell from the stack.

Mark it as visited.

Repeat the exploration process from this new cell.

Step 4: Reaching the Goal ('J')

Continue exploring and popping cells until reaching cell 'J' or the stack becomes empty.

If 'J' is reached during DFS traversal, return "true". Otherwise, return "false".

BFS Python Solution:

C: > Users > jayke > OneDrive > Desktop > algorithm > breath_first_traversal.py > ...

```python
from collections import deque

def hasPath(maze, start, destination):
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
    queue = deque([start])
    visited = set()

    while queue:
        x, y = queue.popleft()
        if (x, y) == tuple(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 < len(maze) and 0 <= newY + dy < len(maze[0]) and maze[newX + dx][newY + dy] != 1:
                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]
print(hasPath(maze1, start1, destination1))  # Output: True

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

BFS Python Solution:

TEST CASES

```
PS C:\Users\jayke\OneDrive\Desktop\algorithm> & C:/Users/jayke/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/jayke/OneDriv
True
False
False
PS C:\Users\jayke\OneDrive\Desktop\algorithm>
```

DFS Python Solution:

C: > Users > jayke > OneDrive > Desktop > algorithm > breath_first_traversal.py > ...

```python
def hasPath(maze, start, destination):
    def dfs(x, y):
        if x == destination[0] and y == destination[1]:
            return True
        if (x, y) in visited:
            return False

        visited.add((x, y))

        directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
        for dx, dy in directions:
            newX, newY = x, y
            while 0 <= newX + dx < len(maze) and 0 <= newY + dy < len(maze[0]) and maze[newX + dx][newY + dy] != 1:
                newX += dx
                newY += dy
            if dfs(newX, newY):
                return True

        return False

    visited = set()
    return dfs(start[0], start[1])

# 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 = [[0,0,0,0,0], [1,1,0,0,1], [0,0,0,0,0], [0,1,0,0,1], [0,1,0,0,0]]
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: True
```

DFS Python Solution:

TEST CASES

```
PS C:\Users\jayke\OneDrive\Desktop\algorithm> & C:/Users/jayke/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/jay
True
False
False
PS C:\Users\jayke\OneDrive\Desktop\algorithm>
```

Conclusion:

By utilizing both BFS and DFS approaches, we can efficiently determine whether the ball can reach the goal in the maze.

BFS ensures systematic exploration in a breadth-first manner, while DFS explores in a depth-first manner.