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

By: Xinye L.

Table of Content

❑ Introduction

❑ Design

❑ Implementation

❑ Test

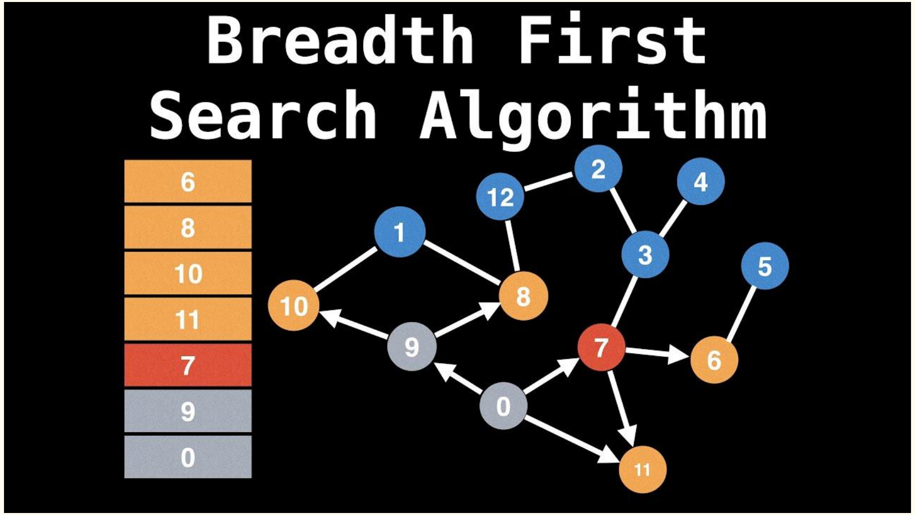
❑ Enhancement Ideas

❑ Conclusion

❑ References

Introduction

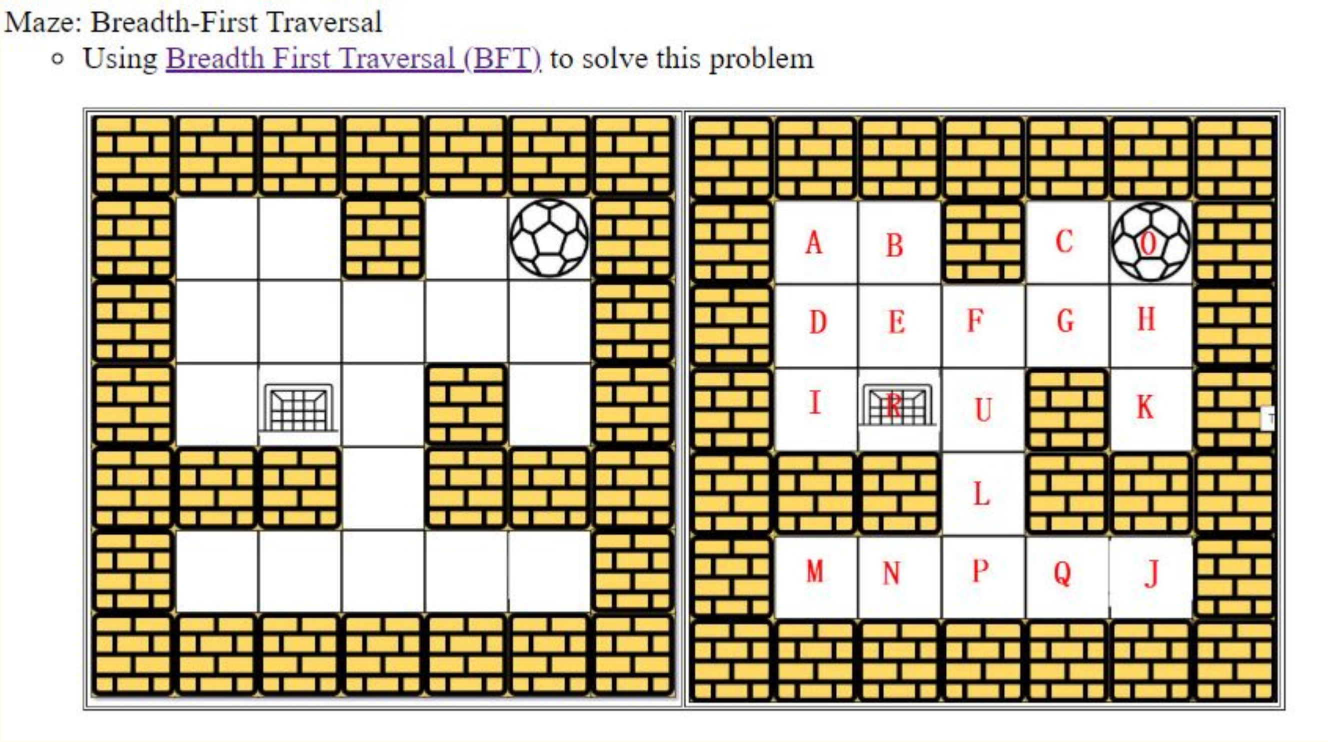
- ❏ Breadth-First Search algorithm
- ❏ Manually solve the maze
- ❏ Python implementation



Breadth-First Search Algorithm

- An algorithm for traversing or searching tree or graph data structures
- Start traversing from a selected node, and traverse the graph layerwise thus exploring the neighbour nodes
- You must then move towards the next-level neighbour nodes

Design - Mase Matrix

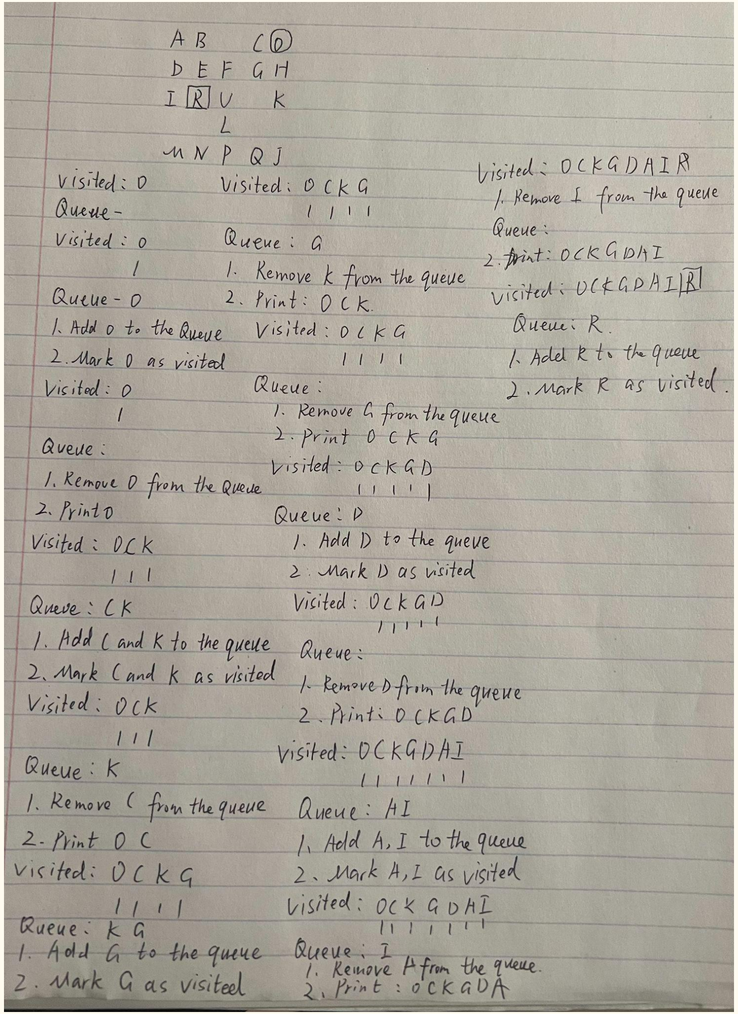


Maze: Breadth-First Traversal

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

❏ Wheeled robots moves in a hotel: BFS

Solution - Matrix Path



A	B		C	O
D	E	F	G	H
I	$\boxed{R}$	U		K
		L		
M	N	P	Q	J

visited: O

Queue -

visited: O
1

Queue - O

1. Add O to the Queue
2. Mark O as visited

Visited: O
1

Queue:

1. Remove O from the Queue
2. Print O

Visited: OCK
1 1 1

Queue: CK

1. Add C and K to the queue
2. Mark C and K as visited

Visited: OCK
1 1 1

Queue: K

1. Remove C from the queue
2. Print O C

visited: O C K G
1 1 1 1

Queue: K G

1. Add G to the queue
2. Mark G as visiteel

Visited: O C K G
1 1 1 1

Queue: G

1. Remove K from the queue
2. Print: O C K.

Visited: O C K G
1 1 1 1

Queue:

1. Remove G from the queue
2. Print O C K G

visited: O C K G D
1 1 1 1 1

Queue: D

1. Add D to the queue
2. Mark D as visited

Visited: OCKGD
1 1 1 1 1

Queue:

1. Remove D from the queue
2. Print: OCKGD

Visited: OCKGDAI
1 1 1 1 1 1 1

Queue: AI

1. Add A, I to the queue
2. Mark A, I as visited

Visited: OCK GDAI
1 1 1 1 1 1 1

Queue: I

1. Remove A from the queue.
2. Print: OCKGDA

Visited: OCKGDAIR

1. Remove I from the queue

Queue:

2. Print: OCKGDAI

visited: OCKGDAI$\boxed{R}$

Queue: R.

1. Add R to the queue
2. Mark R as visited.

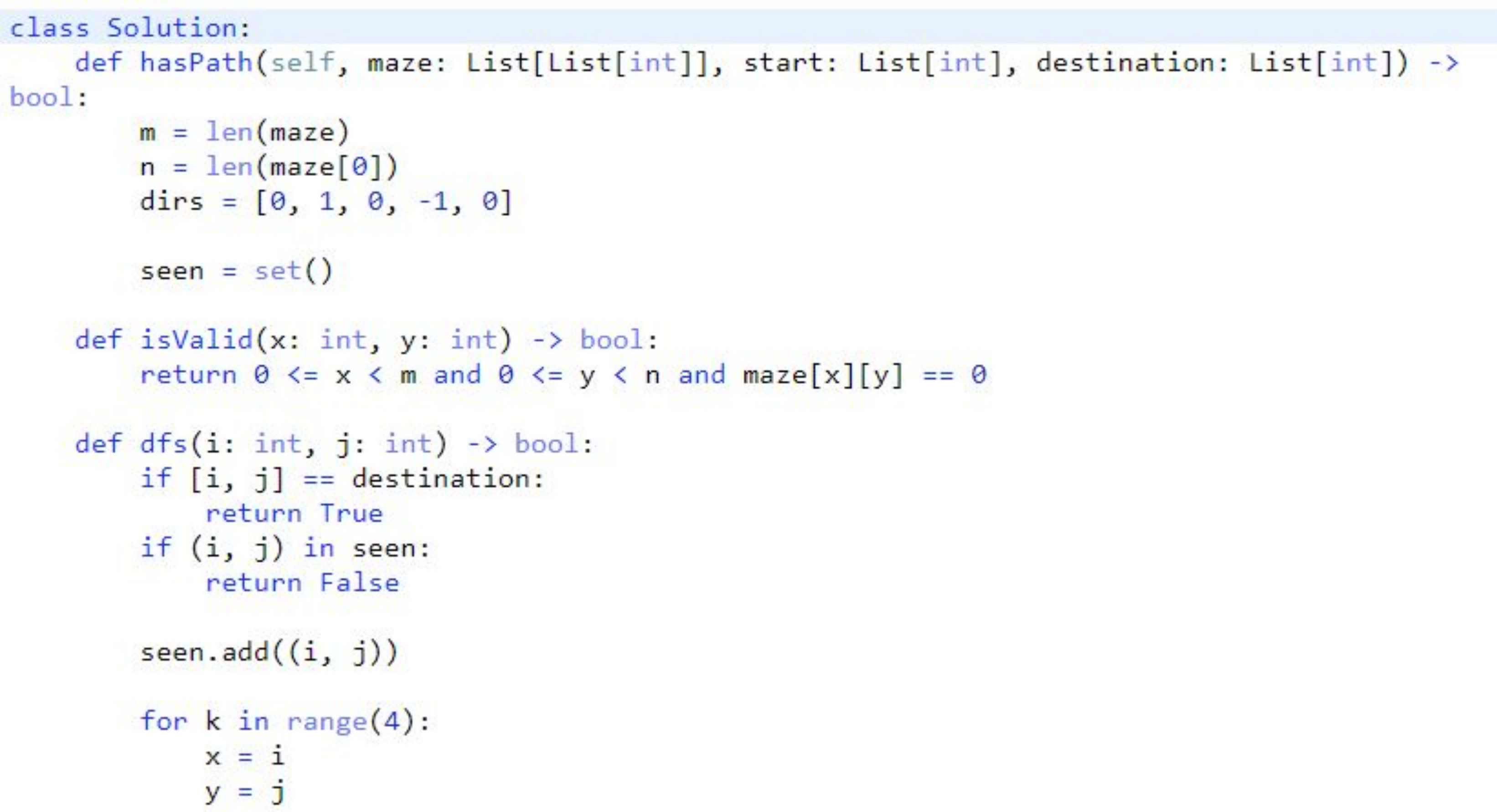
Implementation

❏ There is only one ball and one destination in the maze

❏ Both the ball and the destination exist on an empty space, and they will not be at the same position initially

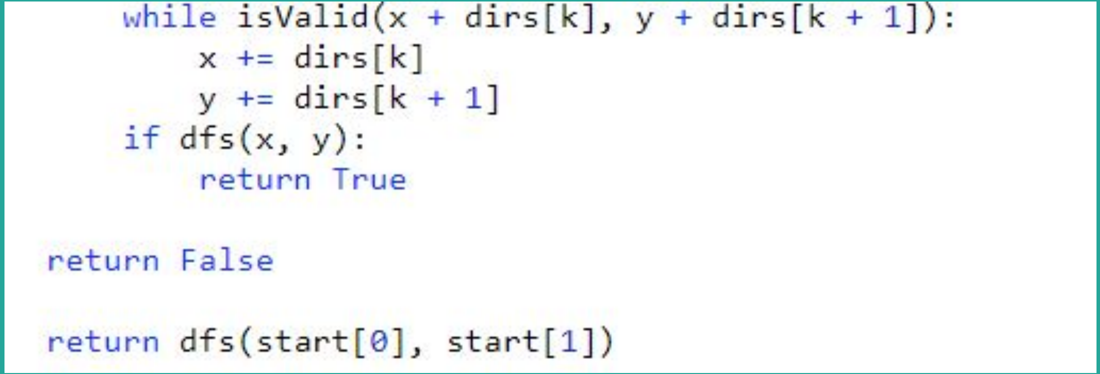
❏ The given maze does not contain border but the border of the maze are all walls

❏ The maze contains at least 2 empty spaces, and both the width and height of the maze won't exceed 100



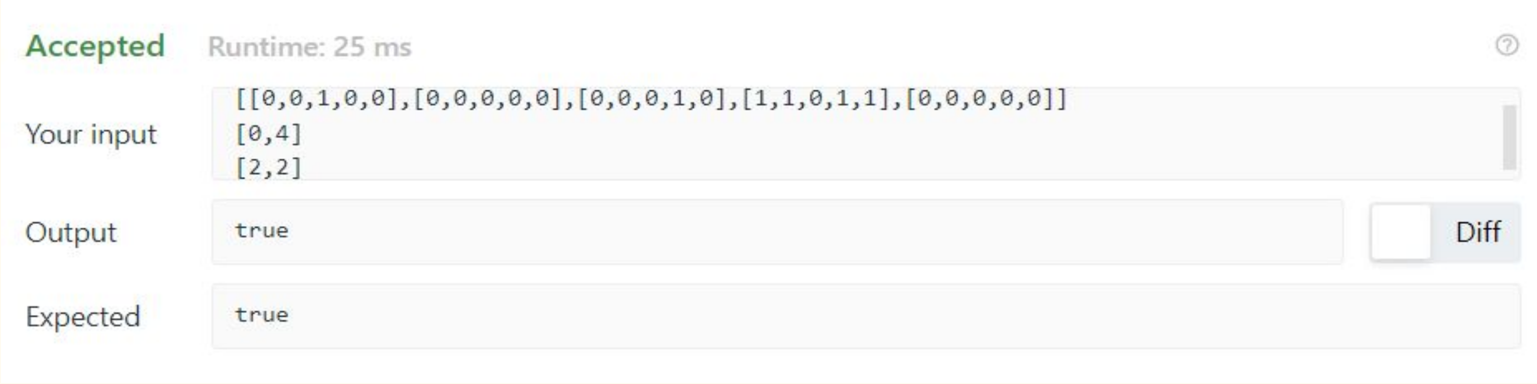
```
class Solution:
    def hasPath(self, maze: List[List[int]], start: List[int], destination: List[int]) ->
bool:
        m = len(maze)
        n = len(maze[0])
        dirs = [0, 1, 0, -1, 0]

        seen = set()

    def isValid(x: int, y: int) -> bool:
        return 0 <= x < m and 0 <= y < n and maze[x][y] == 0

    def dfs(i: int, j: int) -> bool:
        if [i, j] == destination:
            return True
        if (i, j) in seen:
            return False

        seen.add((i, j))

        for k in range(4):
            x = i
            y = j
```

Python implementation continued…



```python
    while isValid(x + dirs[k], y + dirs[k + 1]):
        x += dirs[k]
        y += dirs[k + 1]
    if dfs(x, y):
        return True

return False

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

Test cases



Accepted Runtime: 25 ms

Your input

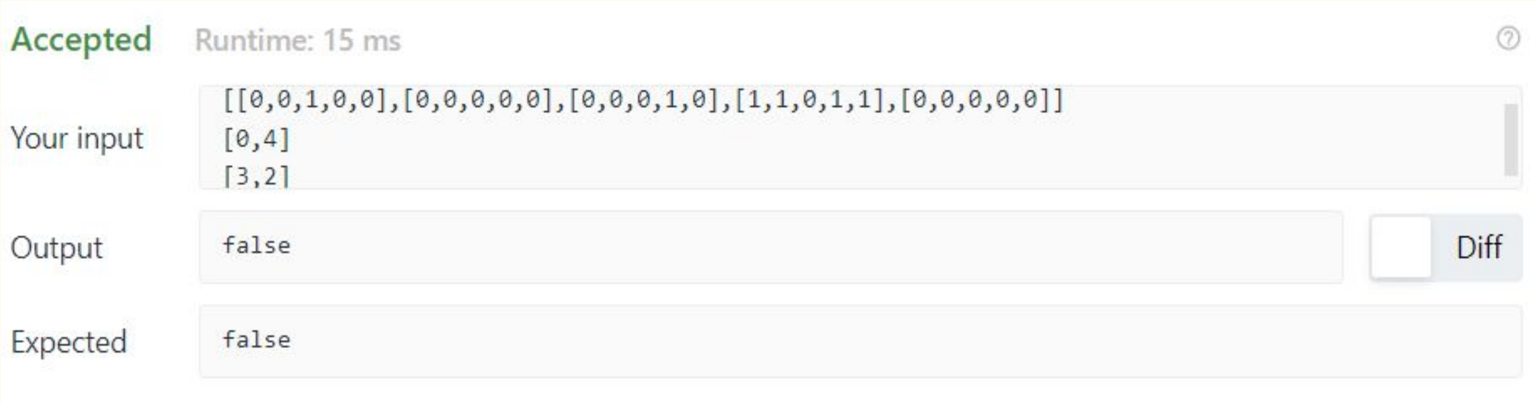
```
[[0,0,1,0,0],[0,0,0,0,0],[0,0,0,1,0],[1,1,0,1,1],[0,0,0,0,0]]
[0,4]
[2,2]
```

Output

```
true
```

Expected

```
true
```



Accepted Runtime: 15 ms

Your input

```
[[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]]
[0,4]
[3,2]
```

Output

```
false
```

Expected

```
false
```

Enhancement ideas

❑ Time: O(mn)

❑ Space: O(mn)

Conclusion

❑ We introduced the breadth-first search algorithm

❑ BFT problems can be solved manually

❑ We implemented and tested its python implementation

thank you

References

❏ Garg, P. (2016). *Breadth First Search Tutorials & Notes | Algorithms | HackerEarth*. HackerEarth.

https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

❏ Jeffrey. (2020, March 22). *leetcode 490. The Maze (Python)*. (Jeffrey's Blog).

https://zhenyu0519.github.io/2020/03/22/lc490/