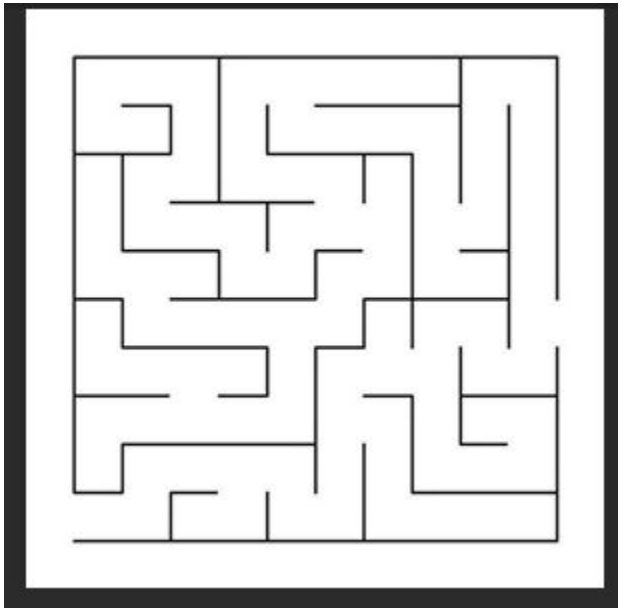
MAZE

----The Shortest Path---

Fatema Nagori 19635

Table of Content:

Introduction



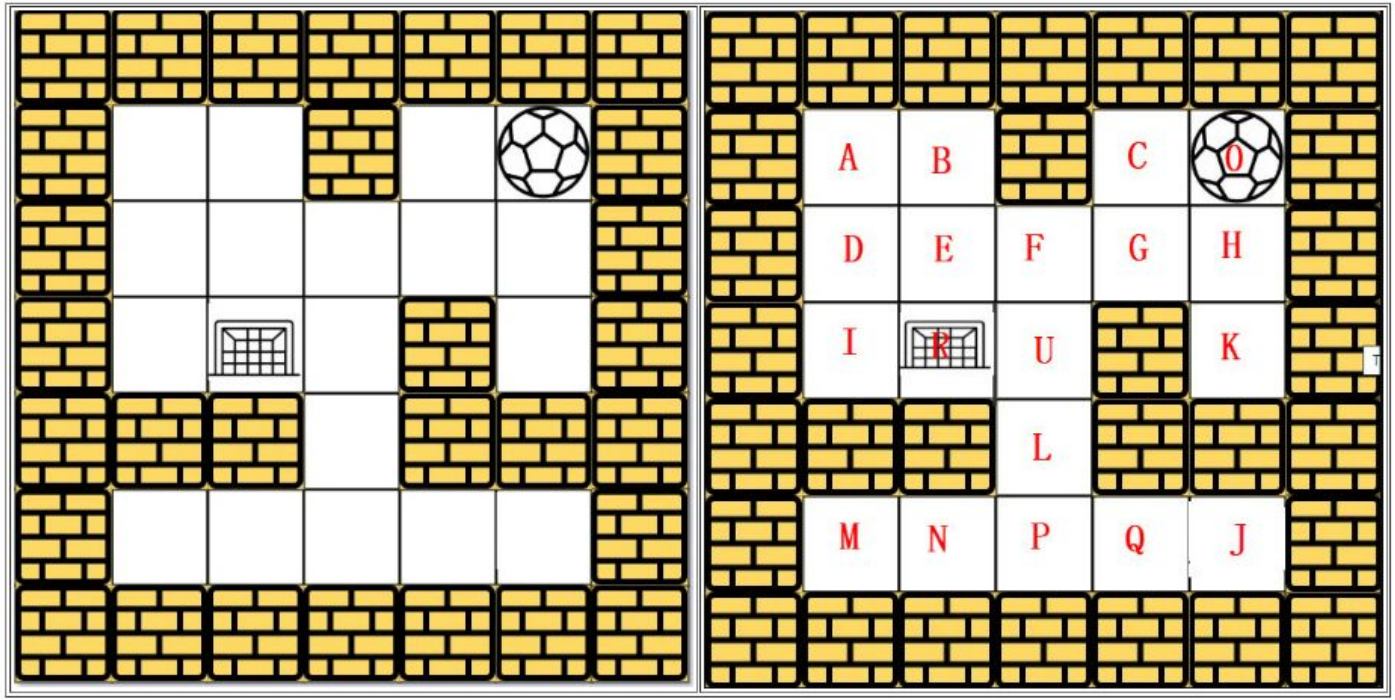
In this Project we solve maze using Breadth First Search(BFS) approach, we also demonstrate manual solution to find the shortest path of MAZE with unclear route(like WHEELED ROBOT MOVE IN HOTEL) using BFS , This methods is used to analyse the cost, time and business it can provide in real time.

Design Approaches:

1) Breadth-First Search (BFS)

2) Depth-First Search (DFS)

STEP 2.1 Manual Process Breadth-First Search



STEP 2.1 Breadth-First Search Implementation

```
Visited: 0
         0
Queue:
```

```
Visited: 0
         1
Queue: 0
1. Add 0 to the queue
2. Mark 0 as visited
```

```
Visited: 0
         1
Queue:
1. Remove 0 from the
queue
2. Print 0
```

```
Visited: 0 C K
         1 1 1
Queue: C K
1. Add C and K to the
queue
2. Mark C and K as
visited
```

```
Visited: 0 C K
         1 1 1
Queue: K
1. Remove C from the
queue
2. Print 0 C
```

```
Visited: 0 C K G
         1 1 1 1
Queue: K G
1. Add G to the queue
2. Mark G as visited
```

```
Visited: 0 C K G
         1 1 1 1
Queue: G
1. Remove K from the
queue
2. Print: 0 C K
```

```
Visited: 0 C K G
         1 1 1 1
Queue:
1. Remove G from the
queue
2. Print 0 C K G
```

```
Visited: 0 C K G D
         1 1 1 1 1
Queue: D
1. Add D to the queue
2. Mark D as visited
```

```
Visited: 0 C K G D
         1 1 1 1 1
Queue:
1. Remove D from the
queue
2. Print: 0 C K G D
```

```
Visited: 0 C K G D A I
         1 1 1 1 1 1 1
Queue: A I
1. Add A, I to the
queue
2. Mark A, I as visited
Visited: 0 C K G D A I
         1 1 1 1 1 1 1
Queue: I
1. Remove A from the
queue
2. Print: 0 C K G D A
```

```
Visited: 0 C K G D A I
B
         1 1 1 1 1 1 1
1
Queue: I B
1. Add B to the queue
2. Mark B as visited
```

```
Visited: 0 C K G D A I
B
         1 1 1 1 1 1 1
1
Queue: B
1. Remove I from the
queue
2. Print: 0 C K G D A I
```

```
Visited: 0 C K G D A I
B R
         1 1 1 1 1 1 1
1 1
Queue: B R
1. Add R to the queue
2. Mark R as visited
```

```
Visited: 0 C K G D A I
B R
         1 1 1 1 1 1 1
1 1
Queue: R
1. Remove B from the
queue
2. Print 0 C K G D A I
B
```

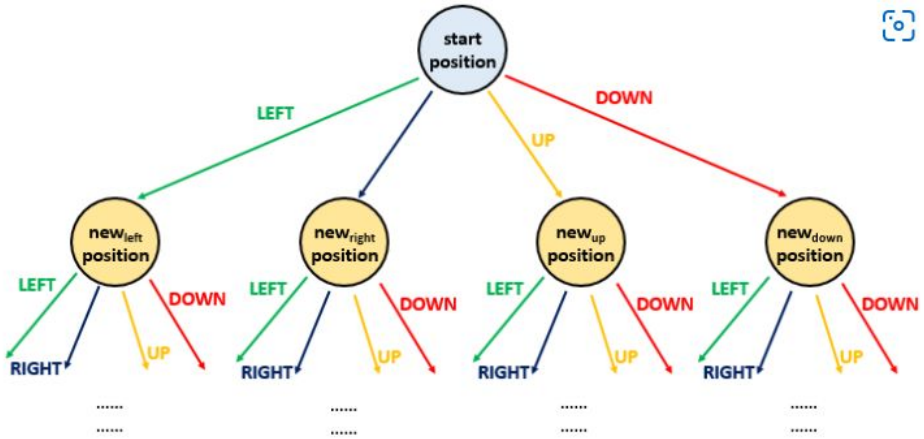
```
Visited: 0 C K G D A I
B R
         1 1 1 1 1 1 1
1 1
Queue:
1. Remove R from the
queue
2. Print 0 C K G D A I
B R
```

Approach : Breadth First Search

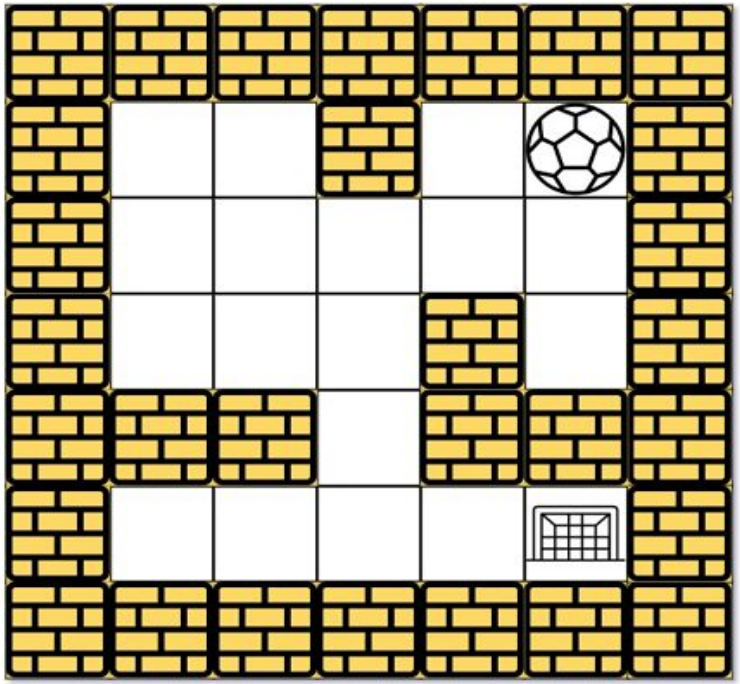
- The same search space tree can also be explored in a Breadth First Search manner.
  - In this case, we try to explore the search space on a level by level basis.
  - i.e., we try to move in all the directions at every step.
  - When all the directions have been explored and we still don't reach the destination, then only we proceed to the new set of traversals from the new positions obtained.
- In order to implement this, we make use of a queue.
  - We start with the ball at the start position.
  - For every current position, we add all the new positions possible by traversing in all the four directions(till reaching the wall or boundary) into the queue to act as the new start positions and mark these positions as True in the visited array.
  - When all the directions have been covered up, we remove a position value, ss, from the front of the queue and again continue the same process with ss acting as the new start position.

- Further, in order to choose the direction of travel, we make use of a dir array, which contains 4 entries.
  - Each entry represents a one-dimensional direction of travel.
  - To travel in a particular direction, we keep on adding the particular entry of the dirs array till we hit a wall or a boundary.
  - For a particular start position, we do this process of dir addition for all all the four directions possible.
- If we hit the destination position at any moment, we return a True directly indicating that the destination position can be reached starting from the start position.

  - The following animation depicts the process:



STEP:2.2 Question 490 Leetcode: Maze

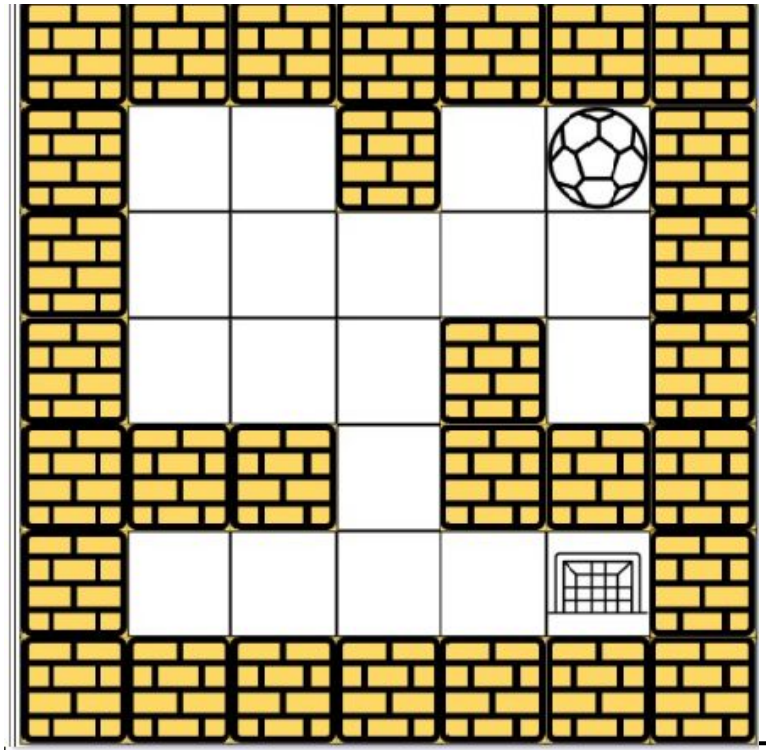


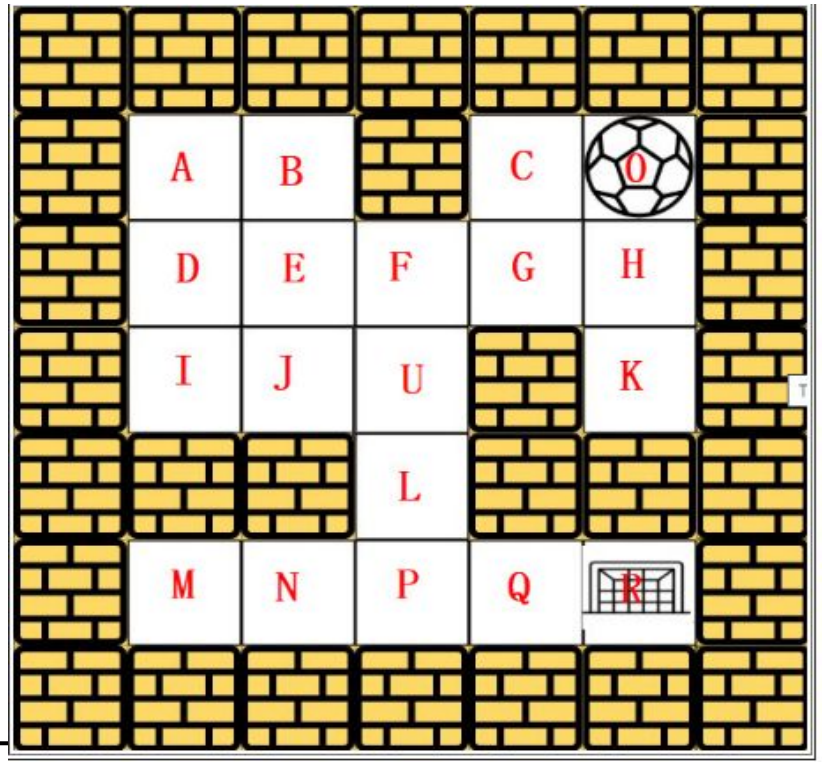
- **Given**: Empty spaces represented as 0 and wall as 1,ball rolls in direction **up, down, left, right** and stops when hits the wall and choose next direction.
- m*n maze
- Start position=[start row, start col]
- destination=[dest row, dest col]
- Return True=ball stops else false

constrain

- m=maze.length
- n==maze[i].length
- 1<=m, n<=100
- Maze[i[j is 0 or 1
- start.length==2
- destination.length==2
- 0<=start row, dest row<=m
- 0<=start col, dest col<=n
- Both ball and destination exist in empty spaces and they will not be in the same position initially
- The maze contains at least 2 empty spaces

SOLUTION using BFS approach





Enhancement Ideas- Compare Algorithm

	TIME COMPLEXITY	SPACE
DEPTH FIRST SEARCH	O(MN)	O(MN)
BREADTH FIRST SEARCH	O(MN)	O(MN)

Test case Diagram

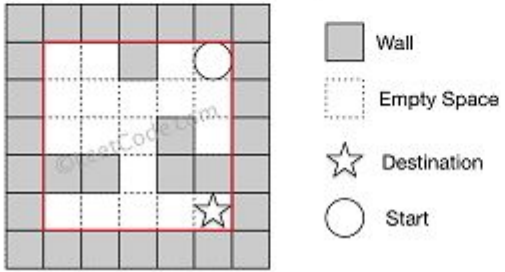
Input 1: a maze represented by a 2D array

```
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
```

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (4, 4)

Output: true

Explanation: One possible way is : left -> down -> left -> down -> right -> down -> right.



Test case Diagram

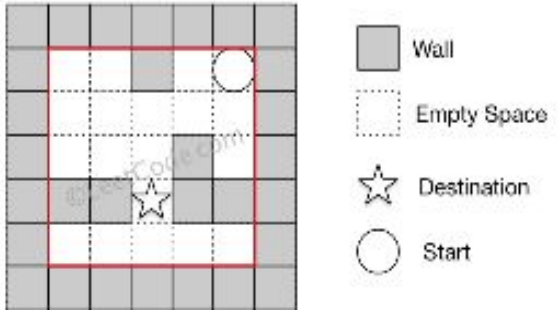
Input 1: a maze represented by a 2D array

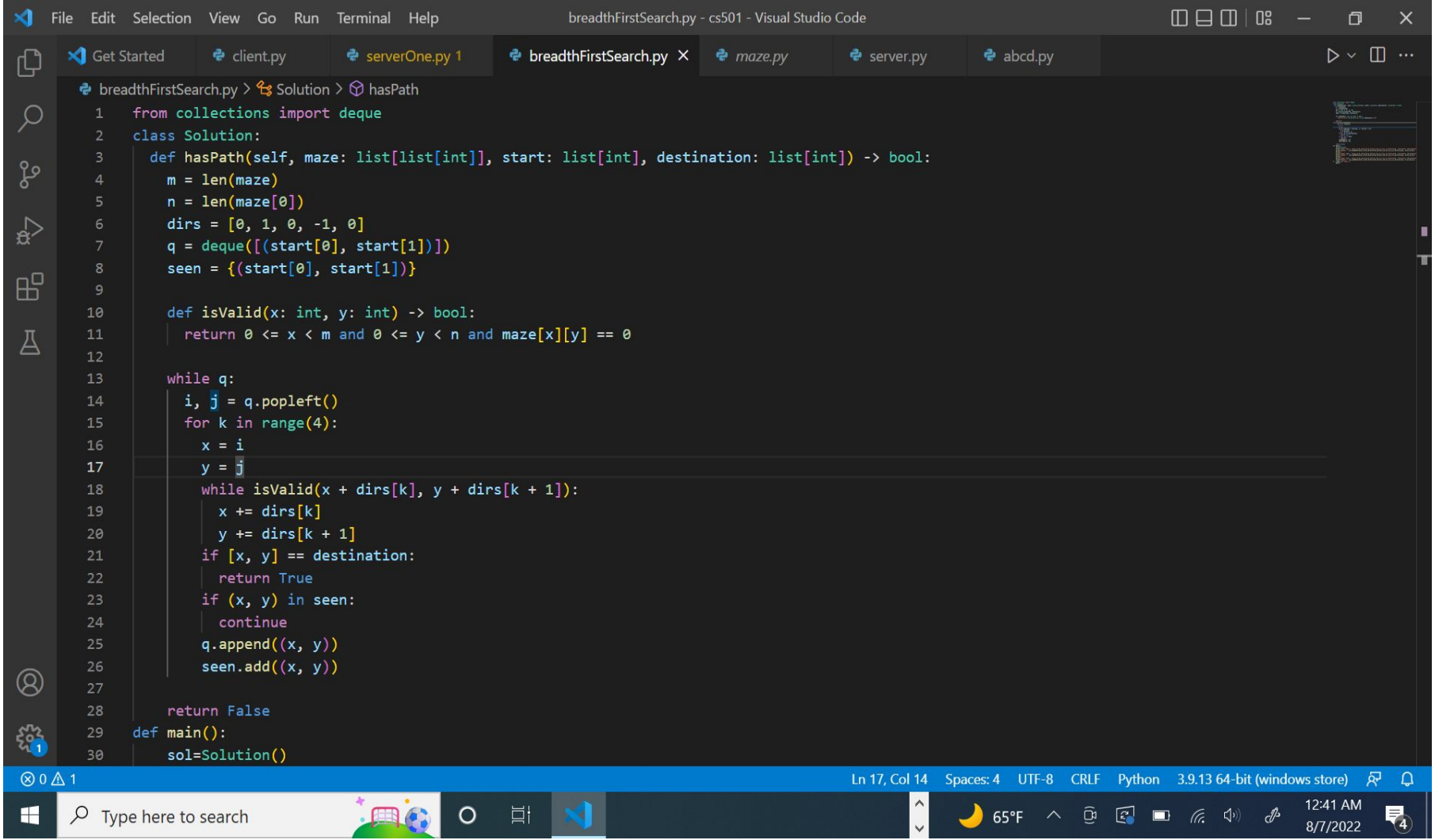
```
0 0 1 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 1 1
0 0 0 0 0
```

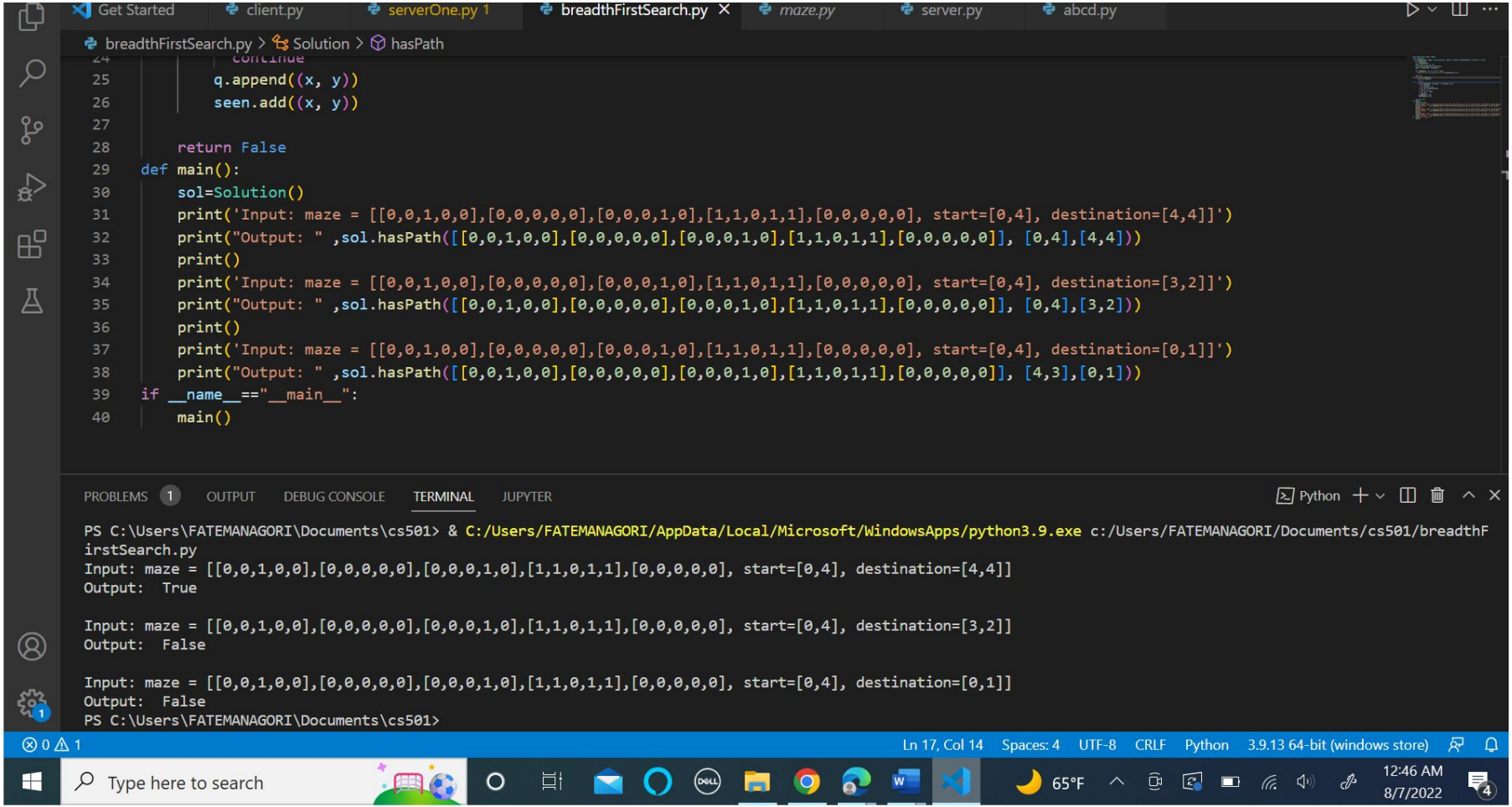
Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (3, 2)

Output: false

Explanation: There is no way for the ball to stop at the destination.




File Edit Selection View Go Run Terminal Help
breadthFirstSearch.py - cs501 - Visual Studio Code
Get Started
client.py
serverOne.py 1
breadthFirstSearch.py
maze.py
server.py
abcd.py
breadthFirstSearch.py > Solution > hasPath
from collections import deque
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]
q = deque([(start[0], start[1])])
seen = {(start[0], start[1])}
def isValid(x: int, y: int) -> bool:
return 0 <= x < m and 0 <= y < n and maze[x][y] == 0
while q:
i, j = q.popleft()
for k in range(4):
x = i
y = j
while isValid(x + dirs[k], y + dirs[k + 1]):
x += dirs[k]
y += dirs[k + 1]
if [x, y] == destination:
return True
if (x, y) in seen:
continue
q.append((x, y))
seen.add((x, y))
return False
def main():
sol=Solution()
Ln 17, Col 14
Spaces: 4
UTF-8
CRLF
Python
3.9.13 64-bit (windows store)
Type here to search
65°F
12:41 AM
8/7/2022


Get Started
client.py
serverOne.py 1
breadthFirstSearch.py
maze.py
server.py
abcd.py
breadthFirstSearch.py > Solution > hasPath
continue
q.append((x, y))
seen.add((x, y))
return False
def main():
sol=Solution()
print('Input: 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("Output: " ,sol.hasPath([[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],[4,4]))
print()
print('Input: 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("Output: " ,sol.hasPath([[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]))
print()
print('Input: 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=[0,1]]')
print("Output: " ,sol.hasPath([[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]], [4,3],[0,1]))
if __name__=="__main__":
main()
PROBLEMS 1
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
Python
PS C:\Users\FATEMANAGORI\Documents\cs501> & C:/Users/FATEMANAGORI/AppData/Local/Microsoft/WindowsApps/python3.9.exe c:/Users/FATEMANAGORI/Documents/cs501/breadthFirstSearch.py
Input: 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]]
Output: True
Input: 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]]
Output: False
Input: 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=[0,1]]
Output: False
PS C:\Users\FATEMANAGORI\Documents\cs501>
Ln 17, Col 14
Spaces: 4
UTF-8
CRLF
Python
3.9.13 64-bit (windows store)
Type here to search
65°F
12:46 AM
8/7/2022

Conclusion

The shortest path can be find using Depth-First Search and Breadth-First Search . However In this project Breadth-First Search is used to find shortest path successfully. For the Leetcode question both the approaches takes same time and space, but in general For mazes specifically (if we define a maze as there being only one way to reach a cell from the starting point without backtracking, meaning it's essentially a tree), BFS will generally use more memory, as we'll need to keep multiple paths in memory at the same time, where DFS only needs to keep track of a single path at any given time.

Reference

- Prof Chang's Class Material CS501 BFS
- LeetCode