Problem 490: The Maze

Problem Information

Difficulty: Medium
Acceptance Rate: 59.88%
Paid Only: Yes
Tags: Array, Depth-First Search, Breadth-First Search, Matrix

Problem Description

There is a ball in a `maze` with empty spaces (represented as `0`) and walls (represented as `1`). The ball can go through the empty spaces by rolling **up, down, left or right** , but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction.

Given the `m x n` `maze`, the ball's `start` position and the `destination`, where `start = [startrow, startcol]` and `destination = [destinationrow, destinationcol]`, return `true` if the ball can stop at the destination, otherwise return `false`.

You may assume that **the borders of the maze are all walls** (see examples).

Example 1:

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

Example 2:

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 **Explanation:** There is no way for the ball to stop at the destination. Notice that you can pass through the destination but you cannot stop there.

Example 3:

Input: 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] **Output:** false

Constraints:

* `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 <= startrow, destinationrow < m` * `0 <= startcol, destinationcol < n` * Both the ball and the destination exist in an empty space, and they will not be in the same position initially. * The maze contains **at least 2 empty spaces**.

Code Snippets

C++:

```
class Solution {
public:
bool hasPath(vector<vector<int>>& maze, vector<int>& start, vector<int>&
destination) {

}
};
```

Java:

```
class Solution {
public boolean hasPath(int[][] maze, int[] start, int[] destination) {

}
}
```

Python3:

```
class Solution:
def hasPath(self, maze: List[List[int]], start: List[int], destination:
List[int]) -> bool:
```