Problem 490: The Maze

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

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 = [start

row

, start

col

]

and

destination = [destination

row

, destination

col

]

, return

true

if the ball can stop at the destination, otherwise return

false

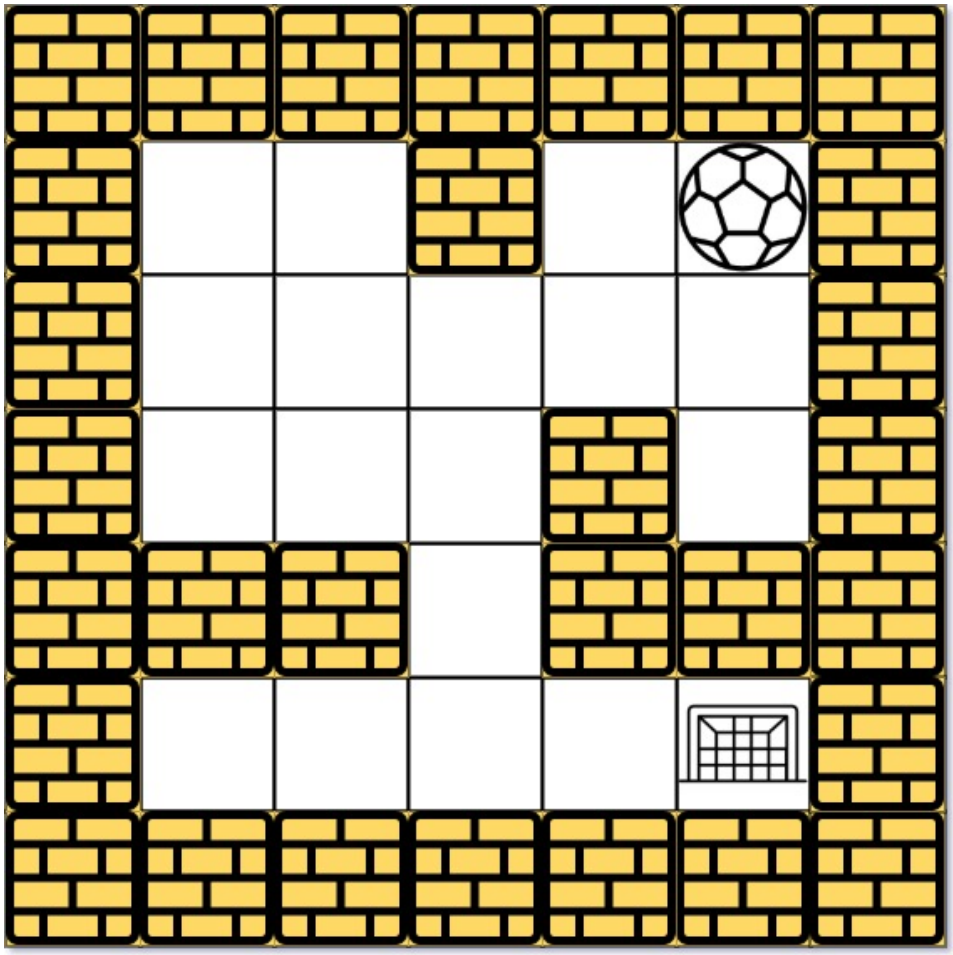
.

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]

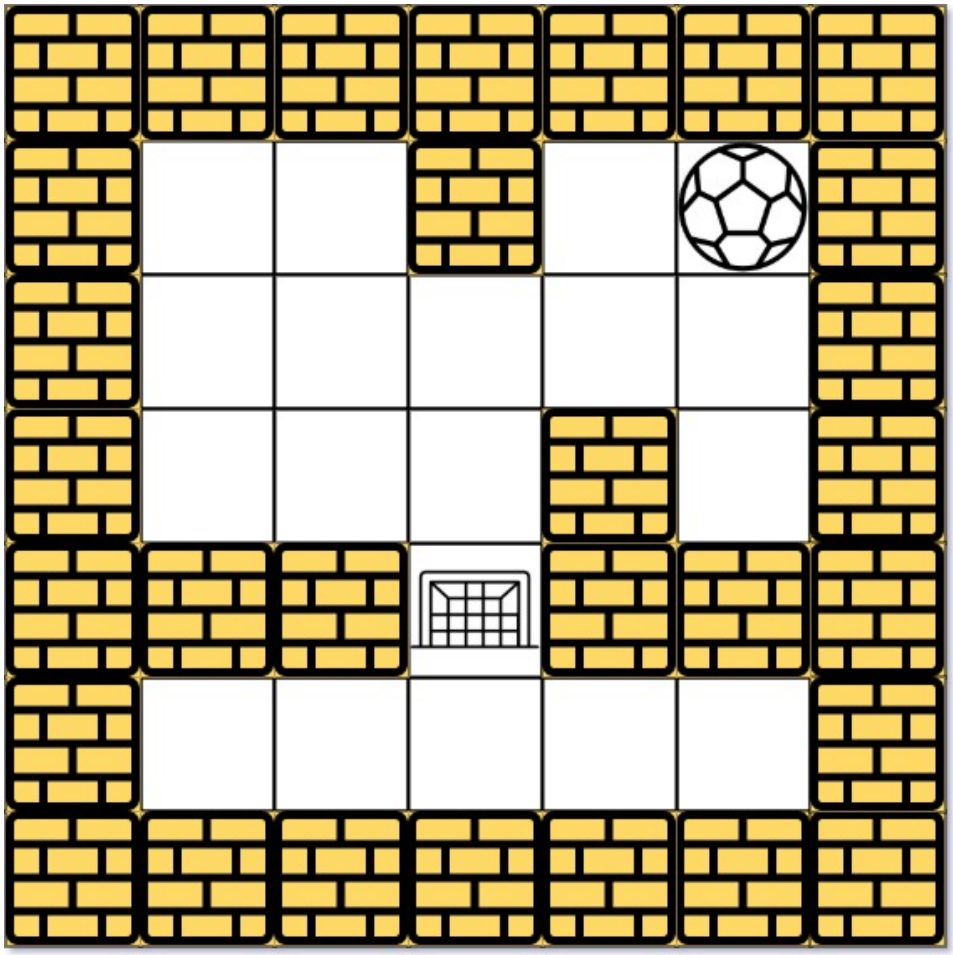
Output:

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 <= start

row

, destination

row

< m

0 <= start

col

, destination

col

< 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:
```

Python:

```
class Solution(object):
def hasPath(self, maze, start, destination):
"""
:type maze: List[List[int]]
:type start: List[int]
:type destination: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[][]} maze
* @param {number[]} start
* @param {number[]} destination
* @return {boolean}
*/
var hasPath = function(maze, start, destination) {

};
```

TypeScript:

```
function hasPath(maze: number[][], start: number[], destination: number[]):
boolean {

};
```

C#:

```
public class Solution {
public bool HasPath(int[][] maze, int[] start, int[] destination) {

}
}
```

C:

```
bool hasPath(int** maze, int mazeSize, int* mazeColSize, int* start, int
startSize, int* destination, int destinationSize) {

}
```

Go:

```
func hasPath(maze [][]int, start []int, destination []int) bool {

}
```

Kotlin:

```
class Solution {
fun hasPath(maze: Array<IntArray>, start: IntArray, destination: IntArray):
Boolean {

}
}
```

Swift:

```
class Solution {
func hasPath(_ maze: [[Int]], _ start: [Int], _ destination: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn has_path(maze: Vec<Vec<i32>>, start: Vec<i32>, destination: Vec<i32>)
-> bool {
```

```
}
}
```

Ruby:

```
# @param {Integer[][]} maze
# @param {Integer[]} start
# @param {Integer[]} destination
# @return {Boolean}
def has_path(maze, start, destination)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $maze
* @param Integer[] $start
* @param Integer[] $destination
* @return Boolean
*/
function hasPath($maze, $start, $destination) {

}
}
```

Dart:

```
class Solution {
bool hasPath(List<List<int>> maze, List<int> start, List<int> destination) {

}
}
```

Scala:

```
object Solution {
def hasPath(maze: Array[Array[Int]], start: Array[Int], destination:
Array[Int]): Boolean = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec has_path(maze :: [[integer]], start :: [integer], destination ::
[integer]) :: boolean
def has_path(maze, start, destination) do

end
end
```

Erlang:

```
-spec has_path(Maze :: [[integer()]], Start :: [integer()], Destination ::
[integer()]) -> boolean().
has_path(Maze, Start, Destination) ->
.
```

Racket:

```
(define/contract (has-path maze start destination)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Maze
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

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

}
};
```

Java Solution:

```
/**
* Problem: The Maze
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: The Maze
Difficulty: Medium
Tags: array, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
```

```
def hasPath(self, maze: List[List[int]], start: List[int], destination:
List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def hasPath(self, maze, start, destination):
"""
:type maze: List[List[int]]
:type start: List[int]
:type destination: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: The Maze
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} maze
* @param {number[]} start
* @param {number[]} destination
* @return {boolean}
*/
var hasPath = function(maze, start, destination) {

};
```

TypeScript Solution:

```
/**
* Problem: The Maze
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function hasPath(maze: number[][], start: number[], destination: number[]):
boolean {

};
```

C# Solution:

```
/*
* Problem: The Maze
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool HasPath(int[][] maze, int[] start, int[] destination) {

}
}
```

C Solution:

```
/*
* Problem: The Maze
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

bool hasPath(int** maze, int mazeSize, int* mazeColSize, int* start, int
startSize, int* destination, int destinationSize) {

}
```

Go Solution:

```
// Problem: The Maze
// Difficulty: Medium
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func hasPath(maze [][]int, start []int, destination []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun hasPath(maze: Array<IntArray>, start: IntArray, destination: IntArray):
Boolean {

}
}
```

Swift Solution:

```
class Solution {
func hasPath(_ maze: [[Int]], _ start: [Int], _ destination: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: The Maze
// Difficulty: Medium
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn has_path(maze: Vec<Vec<i32>>, start: Vec<i32>, destination: Vec<i32>)
-> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} maze
# @param {Integer[]} start
# @param {Integer[]} destination
# @return {Boolean}
def has_path(maze, start, destination)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $maze
* @param Integer[] $start
* @param Integer[] $destination
* @return Boolean
*/
function hasPath($maze, $start, $destination) {

}
}
```

Dart Solution:

```
class Solution {
bool hasPath(List<List<int>> maze, List<int> start, List<int> destination) {

}
}
```

Scala Solution:

```
object Solution {
def hasPath(maze: Array[Array[Int]], start: Array[Int], destination:
Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec has_path(maze :: [[integer]], start :: [integer], destination ::
[integer]) :: boolean
def has_path(maze, start, destination) do

end
end
```

Erlang Solution:

```
-spec has_path(Maze :: [[integer()]], Start :: [integer()], Destination ::
[integer()]) -> boolean().
has_path(Maze, Start, Destination) ->
.
```

Racket Solution:

```
(define/contract (has-path maze start destination)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?) boolean?)
)
```