Problem 505: The Maze II

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

the shortest

distance

for the ball to stop at the destination

. If the ball cannot stop at

destination

, return

-1

.

The

distance

is the number of

empty spaces

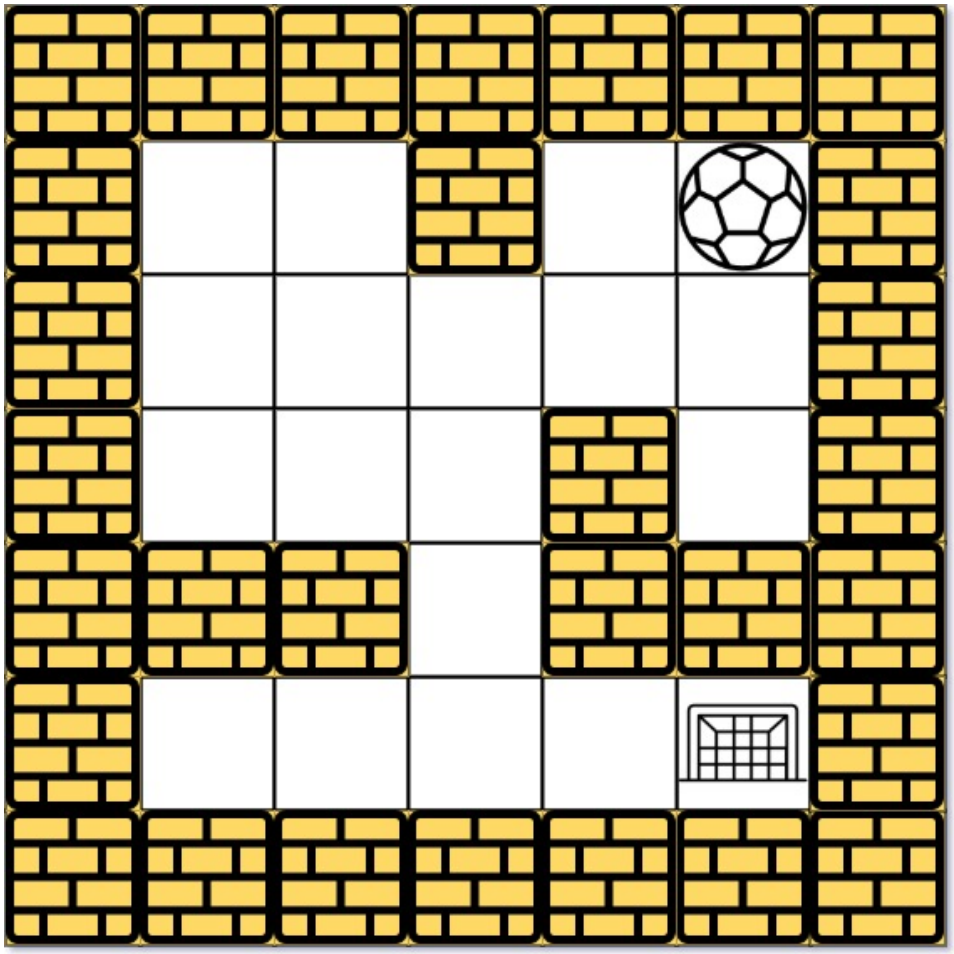
traveled by the ball from the start position (excluded) to the destination (included).

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]

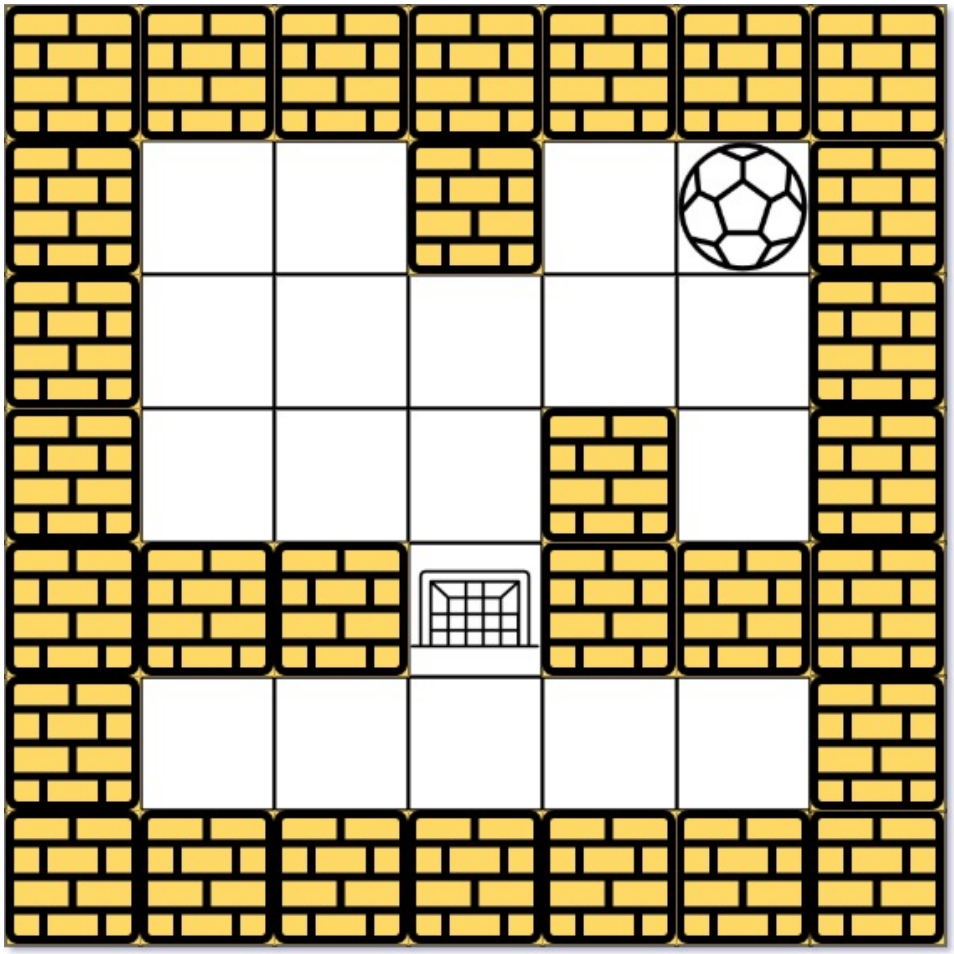
Output:

12

Explanation:

One possible way is : left -> down -> left -> down -> right -> down -> right. The length of the path is 1 + 1 + 3 + 1 + 2 + 2 + 2 = 12.

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:

-1

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:

-1

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:
int shortestDistance(vector<vector<int>>& maze, vector<int>& start,
vector<int>& destination) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (shortest-distance maze start destination)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Maze II
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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:
int shortestDistance(vector<vector<int>>& maze, vector<int>& start,
vector<int>& destination) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: The Maze II
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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 int shortestDistance(int[][] maze, int[] start, int[] destination) {

}
}
```

Python3 Solution:

```
"""
Problem: The Maze II
Difficulty: Medium
Tags: array, graph, search, queue, heap

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: The Maze II
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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 {number}
*/
var shortestDistance = function(maze, start, destination) {

};
```

TypeScript Solution:

```
/**
* Problem: The Maze II
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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 shortestDistance(maze: number[][], start: number[], destination:
number[]): number {

};
```

C# Solution:

```
/*
* Problem: The Maze II
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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 int ShortestDistance(int[][] maze, int[] start, int[] destination) {

}
}
```

C Solution:

```
/*
* Problem: The Maze II
* Difficulty: Medium
* Tags: array, graph, search, queue, heap
*
* 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
*/

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

}
```

Go Solution:

```
// Problem: The Maze II
// Difficulty: Medium
// Tags: array, graph, search, queue, heap
//
// 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 shortestDistance(maze [][]int, start []int, destination []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: The Maze II
// Difficulty: Medium
// Tags: array, graph, search, queue, heap
//
// 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 shortest_distance(maze: Vec<Vec<i32>>, start: Vec<i32>, destination:
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (shortest-distance maze start destination)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```