Problem 1210: Minimum Moves to Reach Target with Rotations

Problem Information

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

Problem Description

In an

n*n

grid, there is a snake that spans 2 cells and starts moving from the top left corner at

(0, 0)

and

(0, 1)

. The grid has empty cells represented by zeros and blocked cells represented by ones. The snake wants to reach the lower right corner at

(n-1, n-2)

and

(n-1, n-1)

.

In one move the snake can:

Move one cell to the right if there are no blocked cells there. This move keeps the horizontal/vertical position of the snake as it is.

Move down one cell if there are no blocked cells there. This move keeps the horizontal/vertical position of the snake as it is.

Rotate clockwise if it's in a horizontal position and the two cells under it are both empty. In that case the snake moves from

(r, c)

and

(r, c+1)

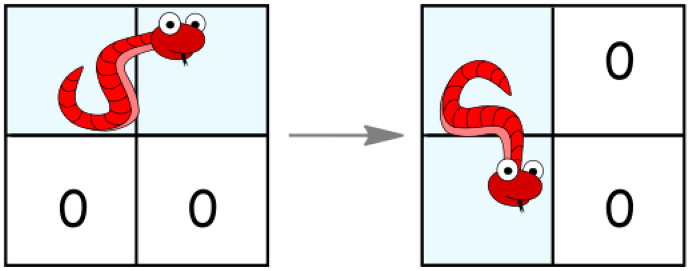
to

(r, c)

and

(r+1, c)

.



Rotate counterclockwise if it's in a vertical position and the two cells to its right are both empty. In that case the snake moves from

(r, c)

and

(r+1, c)

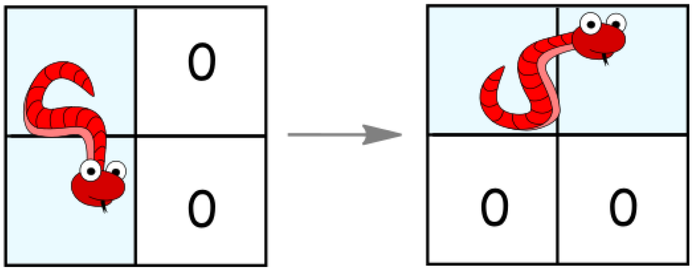
to

(r, c)

and

(r, c+1)

.



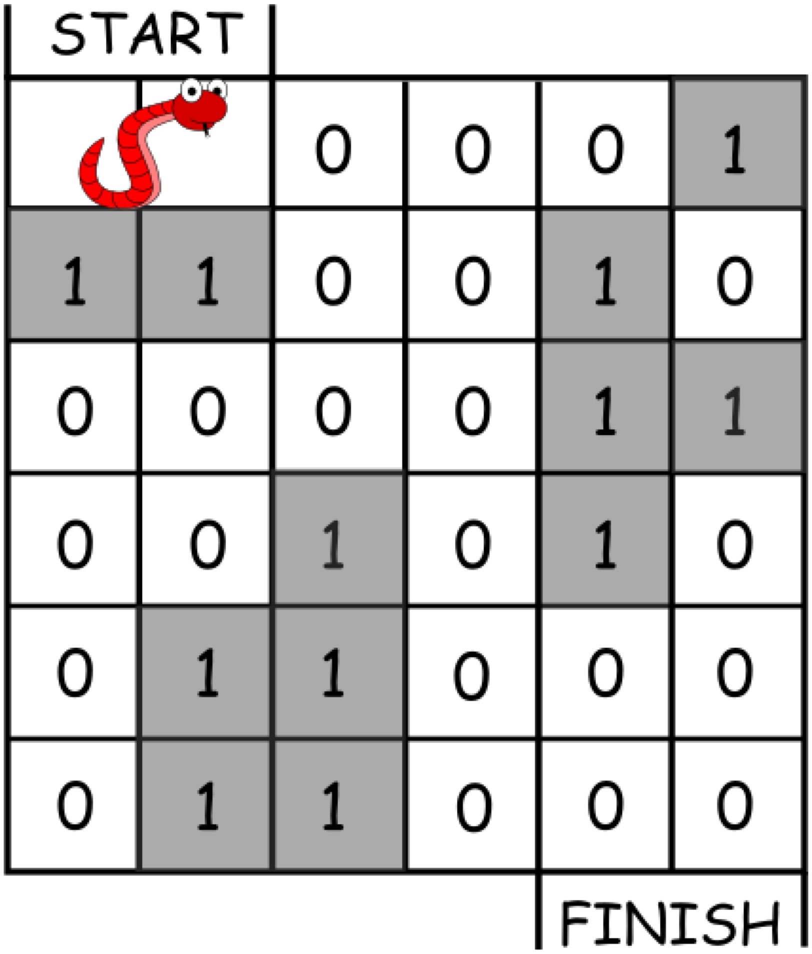
Return the minimum number of moves to reach the target.

If there is no way to reach the target, return

-1

.

Example 1:



		0	0	0	1
1	1	0	0	1	0
0	0	0	0	1	1
0	0	1	0	1	0
0	1	1	0	0	0
0	1	1	0	0	0

Input:

grid = [[0,0,0,0,0,1], [1,1,0,0,1,0],   [0,0,0,0,1,1],   [0,0,1,0,1,0],   [0,1,1,0,0,0],   [0,1,1,0,0,0]]

Output:

11

Explanation:

One possible solution is [right, right, rotate clockwise, right, down, down, down, down, rotate counterclockwise, right, down].

Example 2:

Input:

grid = [[0,0,1,1,1,1], [0,0,0,0,1,1], [1,1,0,0,0,1], [1,1,1,0,0,1], [1,1,1,0,0,1], [1,1,1,0,0,0]]

Output:

9

Constraints:

2 <= n <= 100

0 <= grid[i][j] <= 1

It is guaranteed that the snake starts at empty cells.

Code Snippets

C++:

```
class Solution {
public:
int minimumMoves(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public int minimumMoves(int[][] grid) {

}
}
```

Python3:

```
class Solution:
def minimumMoves(self, grid: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def minimumMoves(self, grid):
"""
:type grid: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @return {number}
*/
var minimumMoves = function(grid) {

};
```

TypeScript:

```
function minimumMoves(grid: number[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumMoves(int[][] grid) {

}
}
```

C:

```



int minimumMoves(int** grid, int gridSize, int* gridColSize){

}
```

Go:

```
func minimumMoves(grid [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minimumMoves(grid: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minimumMoves(_ grid: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_moves(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Integer}
def minimum_moves(grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function minimumMoves($grid) {

}
}
```

Scala:

```
object Solution {
def minimumMoves(grid: Array[Array[Int]]): Int = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Moves to Reach Target with Rotations
* Difficulty: Hard
* 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:
int minimumMoves(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Moves to Reach Target with Rotations
* Difficulty: Hard
* 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 int minimumMoves(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Moves to Reach Target with Rotations
Difficulty: Hard
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 minimumMoves(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumMoves(self, grid):
"""
:type grid: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Moves to Reach Target with Rotations
* Difficulty: Hard
* 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[][]} grid
* @return {number}
*/
var minimumMoves = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Moves to Reach Target with Rotations
* Difficulty: Hard
* 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 minimumMoves(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Moves to Reach Target with Rotations
* Difficulty: Hard
* 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 int MinimumMoves(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Moves to Reach Target with Rotations
* Difficulty: Hard
* 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
*/



int minimumMoves(int** grid, int gridSize, int* gridColSize){

}
```

Go Solution:

```
// Problem: Minimum Moves to Reach Target with Rotations
// Difficulty: Hard
// 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 minimumMoves(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumMoves(grid: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumMoves(_ grid: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Moves to Reach Target with Rotations
// Difficulty: Hard
// 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 minimum_moves(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer}
def minimum_moves(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function minimumMoves($grid) {

}
}
```

Scala Solution:

```
object Solution {
def minimumMoves(grid: Array[Array[Int]]): Int = {

}
}
```