Problem 63: Unique Paths II

Problem Information

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

Problem Description

You are given an

m x n

integer array

grid

. There is a robot initially located at the

top-left corner

(i.e.,

grid[0][0]

). The robot tries to move to the

bottom-right corner

(i.e.,

grid[m - 1][n - 1]

). The robot can only move either down or right at any point in time.

An obstacle and space are marked as

1

or

0

respectively in

grid

. A path that the robot takes cannot include

any

square that is an obstacle.

Return

the number of possible unique paths that the robot can take to reach the bottom-right corner

.

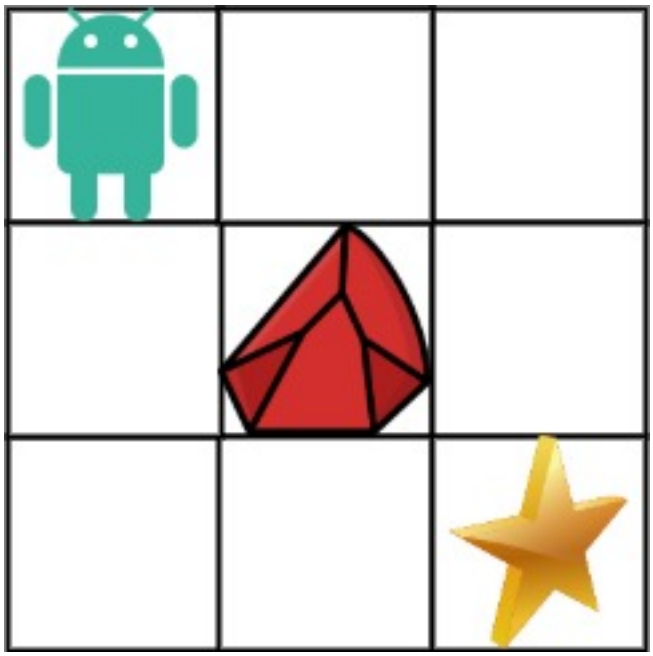
The testcases are generated so that the answer will be less than or equal to

2 * 10

9

.

Example 1:



Input:

obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]

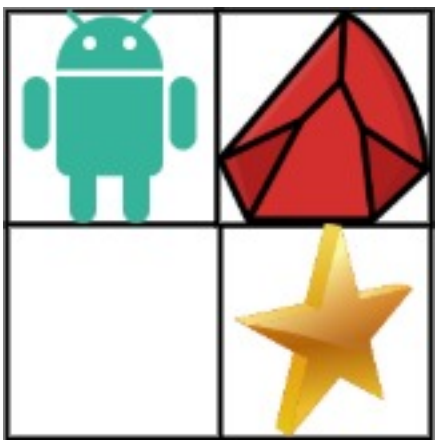
Output:

2

Explanation:

There is one obstacle in the middle of the 3x3 grid above. There are two ways to reach the bottom-right corner: 1. Right -> Right -> Down -> Down 2. Down -> Down -> Right -> Right

Example 2:



Input:

obstacleGrid = [[0,1],[0,0]]

Output:

1

Constraints:

m == obstacleGrid.length

n == obstacleGrid[i].length

1 <= m, n <= 100

obstacleGrid[i][j]

is

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {

}
};
```

Java:

```
class Solution {
public int uniquePathsWithObstacles(int[][] obstacleGrid) {

}
}
```

Python3:

```
class Solution:
def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function uniquePathsWithObstacles(obstacleGrid: number[][]): number {

};
```

C#:

```
public class Solution {
public int UniquePathsWithObstacles(int[][] obstacleGrid) {
```

```
}
}
```

C:

```
int uniquePathsWithObstacles(int** obstacleGrid, int obstacleGridSize, int*
obstacleGridColSize) {

}
```

Go:

```
func uniquePathsWithObstacles(obstacleGrid [][]int) int {

}
```

Kotlin:

```
class Solution {
fun uniquePathsWithObstacles(obstacleGrid: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func uniquePathsWithObstacles(_ obstacleGrid: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn unique_paths_with_obstacles(obstacle_grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} obstacle_grid
# @return {Integer}
def unique_paths_with_obstacles(obstacle_grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $obstacleGrid
* @return Integer
*/
function uniquePathsWithObstacles($obstacleGrid) {

}
}
```

Dart:

```
class Solution {
int uniquePathsWithObstacles(List<List<int>> obstacleGrid) {

}
}
```

Scala:

```
object Solution {
def uniquePathsWithObstacles(obstacleGrid: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec unique_paths_with_obstacles(obstacle_grid :: [[integer]]) :: integer
def unique_paths_with_obstacles(obstacle_grid) do

end
end
```

Erlang:

```
-spec unique_paths_with_obstacles(ObstacleGrid :: [[integer()]]) ->
integer().
unique_paths_with_obstacles(ObstacleGrid) ->
.
```

Racket:

```
(define/contract (unique-paths-with-obstacles obstacleGrid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Unique Paths II
* Difficulty: Medium
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {

}
};
```

Java Solution:

```
/**
* Problem: Unique Paths II
* Difficulty: Medium
* Tags: array, tree, dp
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int uniquePathsWithObstacles(int[][] obstacleGrid) {

}
}
```

Python3 Solution:

```
"""
Problem: Unique Paths II
Difficulty: Medium
Tags: array, tree, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Unique Paths II
```

```
* Difficulty: Medium
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} obstacleGrid
* @return {number}
*/
var uniquePathsWithObstacles = function(obstacleGrid) {

};
```

TypeScript Solution:

```
/**
* Problem: Unique Paths II
* Difficulty: Medium
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function uniquePathsWithObstacles(obstacleGrid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Unique Paths II
* Difficulty: Medium
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int UniquePathsWithObstacles(int[][] obstacleGrid) {

}
}
```

C Solution:

```
/*
* Problem: Unique Paths II
* Difficulty: Medium
* Tags: array, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int uniquePathsWithObstacles(int** obstacleGrid, int obstacleGridSize, int*
obstacleGridColSize) {

}
```

Go Solution:

```
// Problem: Unique Paths II
// Difficulty: Medium
// Tags: array, tree, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func uniquePathsWithObstacles(obstacleGrid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun uniquePathsWithObstacles(obstacleGrid: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func uniquePathsWithObstacles(_ obstacleGrid: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Unique Paths II
// Difficulty: Medium
// Tags: array, tree, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn unique_paths_with_obstacles(obstacle_grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} obstacle_grid
# @return {Integer}
def unique_paths_with_obstacles(obstacle_grid)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[][] $obstacleGrid
* @return Integer
*/
function uniquePathsWithObstacles($obstacleGrid) {

}
}
```

Dart Solution:

```
class Solution {
int uniquePathsWithObstacles(List<List<int>> obstacleGrid) {

}
}
```

Scala Solution:

```
object Solution {
def uniquePathsWithObstacles(obstacleGrid: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec unique_paths_with_obstacles(obstacle_grid :: [[integer]]) :: integer
def unique_paths_with_obstacles(obstacle_grid) do

end
end
```

Erlang Solution:

```
-spec unique_paths_with_obstacles(ObstacleGrid :: [[integer()]]) ->
integer().
unique_paths_with_obstacles(ObstacleGrid) ->
.
```

Racket Solution:

```
(define/contract (unique-paths-with-obstacles obstacleGrid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```