Problem 62: Unique Paths

Problem Information

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

Problem Description

There is a robot on an

m x n

grid. The robot is 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.

Given the two integers

m

and

n

, return

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

.

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

2 * 10

9

.

Example 1:

Input:

m = 3, n = 7

Output:

28

Example 2:

Input:

m = 3, n = 2

Output:

3

Explanation:

From the top-left corner, there are a total of 3 ways to reach the bottom-right corner: 1. Right -> Down -> Down 2. Down -> Down -> Right 3. Down -> Right -> Down

Constraints:

1 <= m, n <= 100

Code Snippets

C++:

```
class Solution {
public:
int uniquePaths(int m, int n) {

}
};
```

Java:

```
class Solution {
public int uniquePaths(int m, int n) {

}
}
```

Python3:

```
class Solution:
def uniquePaths(self, m: int, n: int) -> int:
```

Python:

```
class Solution(object):
def uniquePaths(self, m, n):
"""
:type m: int
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} m
* @param {number} n
* @return {number}
*/
var uniquePaths = function(m, n) {

};
```

TypeScript:

```
function uniquePaths(m: number, n: number): number {

};
```

C#:

```
public class Solution {
public int UniquePaths(int m, int n) {

}
}
```

C:

```
int uniquePaths(int m, int n) {

}
```

Go:

```
func uniquePaths(m int, n int) int {

}
```

Kotlin:

```
class Solution {
fun uniquePaths(m: Int, n: Int): Int {

}
}
```

Swift:

```
class Solution {
func uniquePaths(_ m: Int, _ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn unique_paths(m: i32, n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} m
# @param {Integer} n
# @return {Integer}
def unique_paths(m, n)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer $m
* @param Integer $n
* @return Integer
*/
function uniquePaths($m, $n) {

}
}
```

Dart:

```
class Solution {
int uniquePaths(int m, int n) {

}
}
```

Scala:

```
object Solution {
def uniquePaths(m: Int, n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec unique_paths(m :: integer, n :: integer) :: integer
def unique_paths(m, n) do

end
end
```

Erlang:

```
-spec unique_paths(M :: integer(), N :: integer()) -> integer().
unique_paths(M, N) ->
.
```

Racket:

```
(define/contract (unique-paths m n)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Unique Paths
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int uniquePaths(int m, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Unique Paths
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int uniquePaths(int m, int n) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Unique Paths
Difficulty: Medium
Tags: dp, math

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def uniquePaths(self, m: int, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def uniquePaths(self, m, n):
"""
:type m: int
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Unique Paths
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
/**
* @param {number} m
* @param {number} n
* @return {number}
*/
var uniquePaths = function(m, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Unique Paths
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function uniquePaths(m: number, n: number): number {

};
```

C# Solution:

```
/*
* Problem: Unique Paths
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int UniquePaths(int m, int n) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Unique Paths
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int uniquePaths(int m, int n) {

}
```

Go Solution:

```
// Problem: Unique Paths
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func uniquePaths(m int, n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun uniquePaths(m: Int, n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func uniquePaths(_ m: Int, _ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Unique Paths
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn unique_paths(m: i32, n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} n
# @return {Integer}
def unique_paths(m, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @return Integer
*/
function uniquePaths($m, $n) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int uniquePaths(int m, int n) {

}
}
```

Scala Solution:

```
object Solution {
def uniquePaths(m: Int, n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec unique_paths(m :: integer, n :: integer) :: integer
def unique_paths(m, n) do

end
end
```

Erlang Solution:

```
-spec unique_paths(M :: integer(), N :: integer()) -> integer().
unique_paths(M, N) ->
.
```

Racket Solution:

```
(define/contract (unique-paths m n)
(-> exact-integer? exact-integer? exact-integer?)
)
```