Problem 2328: Number of Increasing Paths in a Grid

Problem Information

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

Problem Description

You are given an

m x n

integer matrix

grid

, where you can move from a cell to any adjacent cell in all

4

directions.

Return

the number of

strictly

increasing

paths in the grid such that you can start from

any

cell and end at

any

cell.

Since the answer may be very large, return it

modulo

10

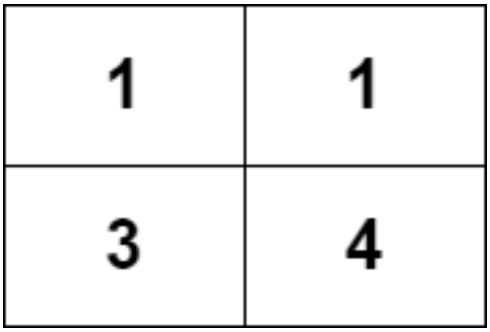
9

+ 7

.

Two paths are considered different if they do not have exactly the same sequence of visited cells.

Example 1:



Input:

grid = [[1,1],[3,4]]

Output:

8

Explanation:

The strictly increasing paths are: - Paths with length 1: [1], [1], [3], [4]. - Paths with length 2: [1 -> 3], [1 -> 4], [3 -> 4]. - Paths with length 3: [1 -> 3 -> 4]. The total number of paths is 4 + 3 + 1 = 8.

Example 2:

Input:

grid = [[1],[2]]

Output:

3

Explanation:

The strictly increasing paths are: - Paths with length 1: [1], [2]. - Paths with length 2: [1 -> 2]. The total number of paths is 2 + 1 = 3.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 1000

1 <= m * n <= 10

5

1 <= grid[i][j] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int countPaths(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int countPaths(List<List<int>> grid) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec count_paths(grid :: [[integer]]) :: integer
def count_paths(grid) do

end
end
```

Erlang:

```
-spec count_paths(Grid :: [[integer()]]) -> integer().
count_paths(Grid) ->
.
```

Racket:

```
(define/contract (count-paths grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Increasing Paths in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* 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 countPaths(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Increasing Paths in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* 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 countPaths(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Increasing Paths in a Grid
Difficulty: Hard
Tags: array, graph, dp, sort, search

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 countPaths(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Increasing Paths in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* 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[][]} grid
* @return {number}
*/
var countPaths = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Increasing Paths in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* 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 countPaths(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Increasing Paths in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* 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 CountPaths(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Number of Increasing Paths in a Grid
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* 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 countPaths(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Number of Increasing Paths in a Grid
// Difficulty: Hard
// Tags: array, graph, dp, sort, search
//
// 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 countPaths(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Increasing Paths in a Grid
// Difficulty: Hard
// Tags: array, graph, dp, sort, search
//
// 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 count_paths(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int countPaths(List<List<int>> grid) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_paths(grid :: [[integer]]) :: integer
def count_paths(grid) do

end
end
```

Erlang Solution:

```
-spec count_paths(Grid :: [[integer()]]) -> integer().
count_paths(Grid) ->
.
```

Racket Solution:

```
(define/contract (count-paths grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```