Problem 2912: Number of Ways to Reach Destination in the Grid

Problem Information

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

Problem Description

You are given two integers

n

and

m

which represent the size of a

1-indexed

grid. You are also given an integer

k

, a

1-indexed

integer array

source

and a

1-indexed

integer array

dest

, where

source

and

dest

are in the form

[x, y]

representing a cell on the given grid.

You can move through the grid in the following way:

You can go from cell

[x

1

, y

1

]

to cell

[x

2

, y

2

]

if either

x

1

== x

2

or

y

1

== y

2

.

Note that you

can't

move to the cell you are already in e.g.

x

1

== x

2

and

y

1

== y

2

.

Return

the number of ways you can reach

dest

from

source

by moving through the grid

exactly

k

times.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

n = 3, m = 2, k = 2, source = [1,1], dest = [2,2]

Output:

2

Explanation:

There are 2 possible sequences of reaching [2,2] from [1,1]: - [1,1] -> [1,2] -> [2,2] - [1,1] -> [2,1] -> [2,2]

Example 2:

Input:

n = 3, m = 4, k = 3, source = [1,2], dest = [2,3]

Output:

9

Explanation:

There are 9 possible sequences of reaching [2,3] from [1,2]: - [1,2] -> [1,1] -> [1,3] -> [2,3] - [1,2] -> [1,1] -> [2,1] -> [2,3] - [1,2] -> [1,3] -> [3,3] -> [2,3] - [1,2] -> [1,4] -> [1,3] -> [2,3] - [1,2] -> [1,4] -> [2,4] -> [2,3] - [1,2] -> [2,2] -> [2,1] -> [2,3] - [1,2] -> [2,2] -> [2,4] -> [2,3] - [1,2] -> [3,2] -> [2,2] -> [2,3] - [1,2] -> [3,2] -> [3,3] -> [2,3]

Constraints:

2 <= n, m <= 10

9

1 <= k <= 10

5

source.length == dest.length == 2

1 <= source[1], dest[1] <= n

1 <= source[2], dest[2] <= m

Code Snippets

C++:

```
class Solution {
public:
int numberOfWays(int n, int m, int k, vector<int>& source, vector<int>& dest)
{

}
};
```

Java:

```
class Solution {
public int numberOfWays(int n, int m, int k, int[] source, int[] dest) {

}
}
```

Python3:

```
class Solution:
def numberOfWays(self, n: int, m: int, k: int, source: List[int], dest:
List[int]) -> int:
```

Python:

```
class Solution(object):
def numberOfWays(self, n, m, k, source, dest):
"""
:type n: int
:type m: int
:type k: int
:type source: List[int]
:type dest: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} m
* @param {number} k
* @param {number[]} source
* @param {number[]} dest
* @return {number}
*/
var numberOfWays = function(n, m, k, source, dest) {

};
```

TypeScript:

```
function numberOfWays(n: number, m: number, k: number, source: number[],
dest: number[]): number {

};
```

C#:

```
public class Solution {
public int NumberOfWays(int n, int m, int k, int[] source, int[] dest) {

}
}
```

C:

```
int numberOfWays(int n, int m, int k, int* source, int sourceSize, int* dest, int destSize) {

}
```

Go:

```
func numberOfWays(n int, m int, k int, source []int, dest []int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfWays(n: Int, m: Int, k: Int, source: IntArray, dest: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func numberOfWays(_ n: Int, _ m: Int, _ k: Int, _ source: [Int], _ dest: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_ways(n: i32, m: i32, k: i32, source: Vec<i32>, dest: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} m
# @param {Integer} k
# @param {Integer[]} source
# @param {Integer[]} dest
# @return {Integer}
def number_of_ways(n, m, k, source, dest)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @param Integer $k
* @param Integer[] $source
* @param Integer[] $dest
* @return Integer
*/
function numberOfWays($n, $m, $k, $source, $dest) {

}
}
```

Dart:

```
class Solution {
int numberOfWays(int n, int m, int k, List<int> source, List<int> dest) {

}
}
```

Scala:

```
object Solution {
def numberOfWays(n: Int, m: Int, k: Int, source: Array[Int], dest:
Array[Int]): Int = {
}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_ways(n :: integer, m :: integer, k :: integer, source ::
[integer], dest :: [integer]) :: integer
def number_of_ways(n, m, k, source, dest) do

end
end
```

Erlang:

```
-spec number_of_ways(N :: integer(), M :: integer(), K :: integer(), Source
:: [integer()], Dest :: [integer()]) -> integer().
number_of_ways(N, M, K, Source, Dest) ->
.
```

Racket:

```
(define/contract (number-of-ways n m k source dest)
(-> exact-integer? exact-integer? exact-integer? (listof exact-integer?)
(listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Reach Destination in the Grid
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 numberOfWays(int n, int m, int k, vector<int>& source, vector<int>& dest)
{

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Reach Destination in the Grid
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 numberOfWays(int n, int m, int k, int[] source, int[] dest) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Reach Destination in the Grid
Difficulty: Hard
Tags: array, dp, math

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 numberOfWays(self, n: int, m: int, k: int, source: List[int], dest:
```

```
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numberOfWays(self, n, m, k, source, dest):
"""
:type n: int
:type m: int
:type k: int
:type source: List[int]
:type dest: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways to Reach Destination in the Grid
* Difficulty: Hard
* Tags: array, dp, math
*
* 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} n
* @param {number} m
* @param {number} k
* @param {number[]} source
* @param {number[]} dest
* @return {number}
*/
var numberOfWays = function(n, m, k, source, dest) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Reach Destination in the Grid
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 numberOfWays(n: number, m: number, k: number, source: number[],
dest: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Reach Destination in the Grid
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 NumberOfWays(int n, int m, int k, int[] source, int[] dest) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Reach Destination in the Grid
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 numberOfWays(int n, int m, int k, int* source, int sourceSize, int* dest,
int destSize) {

}
```

Go Solution:

```
// Problem: Number of Ways to Reach Destination in the Grid
// Difficulty: Hard
// Tags: array, dp, math
//
// 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 numberOfWays(n int, m int, k int, source []int, dest []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfWays(n: Int, m: Int, k: Int, source: IntArray, dest: IntArray):
Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfWays(_ n: Int, _ m: Int, _ k: Int, _ source: [Int], _ dest:
[Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Reach Destination in the Grid
// Difficulty: Hard
// Tags: array, dp, math
//
// 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 number_of_ways(n: i32, m: i32, k: i32, source: Vec<i32>, dest:
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} m
# @param {Integer} k
# @param {Integer[]} source
# @param {Integer[]} dest
# @return {Integer}
def number_of_ways(n, m, k, source, dest)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $m
* @param Integer $k
* @param Integer[] $source
* @param Integer[] $dest
* @return Integer
*/
function numberOfWays($n, $m, $k, $source, $dest) {

}

}
```

Dart Solution:

```
class Solution {
int numberOfWays(int n, int m, int k, List<int> source, List<int> dest) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfWays(n: Int, m: Int, k: Int, source: Array[Int], dest:
Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_ways(n :: integer, m :: integer, k :: integer, source ::
[integer], dest :: [integer]) :: integer
def number_of_ways(n, m, k, source, dest) do

end
end
```

Erlang Solution:

```
-spec number_of_ways(N :: integer(), M :: integer(), K :: integer(), Source
:: [integer()], Dest :: [integer()]) -> integer().
number_of_ways(N, M, K, Source, Dest) ->
.
```

Racket Solution:

```
(define/contract (number-of-ways n m k source dest)
(-> exact-integer? exact-integer? exact-integer? (listof exact-integer?)
(listof exact-integer?) exact-integer?)
)
```