Problem 3193: Count the Number of Inversions

Problem Information

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

Problem Description

You are given an integer

n

and a 2D array

requirements

, where

requirements[i] = [end

i

, cnt

i

]

represents the end index and the

inversion

count of each requirement.

A pair of indices

(i, j)

from an integer array

nums

is called an

inversion

if:

i < j

and

nums[i] > nums[j]

Return the number of

permutations

perm

of

[0, 1, 2, ..., n - 1]

such that for

all

requirements[i]

,

perm[0..end

i

]

has exactly

cnt

i

inversions.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

n = 3, requirements = [[2,2],[0,0]]

Output:

2

Explanation:

The two permutations are:

[2, 0, 1]

Prefix

[2, 0, 1]

has inversions

(0, 1)

and

(0, 2)

.

Prefix

[2]

has 0 inversions.

[1, 2, 0]

Prefix

[1, 2, 0]

has inversions

(0, 2)

and

(1, 2)

.

Prefix

[1]

has 0 inversions.

Example 2:

Input:

n = 3, requirements = [[2,2],[1,1],[0,0]]

Output:

1

Explanation:

The only satisfying permutation is

[2, 0, 1]

:

Prefix

[2, 0, 1]

has inversions

(0, 1)

and

(0, 2)

.

Prefix

[2, 0]

has an inversion

(0, 1)

.

Prefix

[2]

has 0 inversions.

Example 3:

Input:

n = 2, requirements = [[0,0],[1,0]]

Output:

1

Explanation:

The only satisfying permutation is

[0, 1]

:

Prefix

[0]

has 0 inversions.

Prefix

[0, 1]

has an inversion

(0, 1)

.

Constraints:

2 <= n <= 300

1 <= requirements.length <= n

requirements[i] = [end

i

, cnt

i

]

0 <= end

i

<= n - 1

0 <= cnt

i

<= 400

The input is generated such that there is at least one

i

such that

end

i

== n - 1

.

The input is generated such that all

end

i

are unique.

Code Snippets

C++:

```
class Solution {
public:
int numberOfPermutations(int n, vector<vector<int>>& requirements) {

}
};
```

Java:

```
class Solution {
public int numberOfPermutations(int n, int[][] requirements) {

}
}
```

Python3:

```
class Solution:
def numberOfPermutations(self, n: int, requirements: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def numberOfPermutations(self, n, requirements):
"""
:type n: int
:type requirements: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} requirements
* @return {number}
*/
var numberOfPermutations = function(n, requirements) {

};
```

TypeScript:

```
function numberOfPermutations(n: number, requirements: number[][]): number {

};
```

C#:

```
public class Solution {
public int NumberOfPermutations(int n, int[][] requirements) {

}
}
```

C:

```
int numberOfPermutations(int n, int** requirements, int requirementsSize,
int* requirementsColSize) {

}
```

Go:

```
func numberOfPermutations(n int, requirements [][]int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfPermutations(n: Int, requirements: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func numberOfPermutations(_ n: Int, _ requirements: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_permutations(n: i32, requirements: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} requirements
# @return {Integer}
def number_of_permutations(n, requirements)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $requirements
* @return Integer
*/
function numberOfPermutations($n, $requirements) {

}
}
```

Dart:

```
class Solution {
int numberOfPermutations(int n, List<List<int>> requirements) {

}
}
```

Scala:

```
object Solution {
def numberOfPermutations(n: Int, requirements: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_permutations(n :: integer, requirements :: [[integer]]) ::
integer
def number_of_permutations(n, requirements) do

end
end
```

Erlang:

```
-spec number_of_permutations(N :: integer(), Requirements :: [[integer()]])
-> integer().
number_of_permutations(N, Requirements) ->
.
```

Racket:

```
(define/contract (number-of-permutations n requirements)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count the Number of Inversions
* Difficulty: Hard
* Tags: array, 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 numberOfPermutations(int n, vector<vector<int>>& requirements) {

}
};
```

Java Solution:

```
/**
* Problem: Count the Number of Inversions
* Difficulty: Hard
* Tags: array, 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 numberOfPermutations(int n, int[][] requirements) {

}
}
```

Python3 Solution:

```
"""
Problem: Count the Number of Inversions
Difficulty: Hard
Tags: array, 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 numberOfPermutations(self, n: int, requirements: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numberOfPermutations(self, n, requirements):
"""
:type n: int
:type requirements: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count the Number of Inversions
* Difficulty: Hard
```

```
* Tags: array, 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} n
* @param {number[][]} requirements
* @return {number}
*/
var numberOfPermutations = function(n, requirements) {

};
```

TypeScript Solution:

```
/**
* Problem: Count the Number of Inversions
* Difficulty: Hard
* Tags: array, 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 numberOfPermutations(n: number, requirements: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Count the Number of Inversions
* Difficulty: Hard
* Tags: array, 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 NumberOfPermutations(int n, int[][] requirements) {

}
}
```

C Solution:

```
/*
* Problem: Count the Number of Inversions
* Difficulty: Hard
* Tags: array, 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 numberOfPermutations(int n, int** requirements, int requirementsSize,
int* requirementsColSize) {

}
```

Go Solution:

```
// Problem: Count the Number of Inversions
// Difficulty: Hard
// Tags: array, 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 numberOfPermutations(n int, requirements [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfPermutations(n: Int, requirements: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfPermutations(_ n: Int, _ requirements: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count the Number of Inversions
// Difficulty: Hard
// Tags: array, 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 number_of_permutations(n: i32, requirements: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} requirements
# @return {Integer}
def number_of_permutations(n, requirements)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $requirements
* @return Integer
*/
function numberOfPermutations($n, $requirements) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfPermutations(int n, List<List<int>> requirements) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfPermutations(n: Int, requirements: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_permutations(n :: integer, requirements :: [[integer]]) ::
integer
def number_of_permutations(n, requirements) do

end
end
```

Erlang Solution:

```
-spec number_of_permutations(N :: integer(), Requirements :: [[integer()]])
-> integer().
```

```
number_of_permutations(N, Requirements) ->
.
```

Racket Solution:

```
(define/contract (number-of-permutations n requirements)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```