Problem 2580: Count Ways to Group Overlapping Ranges

Problem Information

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

Problem Description

You are given a 2D integer array

ranges

where

ranges[i] = [start

i

, end

i

]

denotes that all integers between

start

i

and

end

i

(both

inclusive

) are contained in the

i

th

range.

You are to split

ranges

into

two

(possibly empty) groups such that:

Each range belongs to exactly one group.

Any two

overlapping

ranges must belong to the

same

group.

Two ranges are said to be

overlapping

if there exists at least

one

integer that is present in both ranges.

For example,

[1, 3]

and

[2, 5]

are overlapping because

2

and

3

occur in both ranges.

Return

the

total number

of ways to split

ranges

into two groups

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

ranges = [[6,10],[5,15]]

Output:

2

Explanation:

The two ranges are overlapping, so they must be in the same group. Thus, there are two possible ways: - Put both the ranges together in group 1. - Put both the ranges together in group 2.

Example 2:

Input:

ranges = [[1,3],[10,20],[2,5],[4,8]]

Output:

4

Explanation:

Ranges [1,3], and [2,5] are overlapping. So, they must be in the same group. Again, ranges [2,5] and [4,8] are also overlapping. So, they must also be in the same group. Thus, there are four possible ways to group them: - All the ranges in group 1. - All the ranges in group 2. - Ranges [1,3], [2,5], and [4,8] in group 1 and [10,20] in group 2. - Ranges [1,3], [2,5], and [4,8] in group 2 and [10,20] in group 1.

Constraints:

1 <= ranges.length <= 10

5

ranges[i].length == 2

0 <= start

i

<= end

i

<= 10

9

Code Snippets

C++:

```
class Solution {
public:
int countWays(vector<vector<int>>& ranges) {

}
};
```

Java:

```
class Solution {
public int countWays(int[][] ranges) {

}
}
```

Python3:

```
class Solution:
def countWays(self, ranges: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countWays(ranges: number[][]): number {

};
```

C#:

```
public class Solution {
public int CountWays(int[][] ranges) {

}
}
```

C:

```
int countWays(int** ranges, int rangesSize, int* rangesColSize) {

}
```

Go:

```
func countWays(ranges [][]int) int {

}
```

Kotlin:

```
class Solution {
fun countWays(ranges: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func countWays(_ ranges: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_ways(ranges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} ranges
# @return {Integer}
def count_ways(ranges)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $ranges
* @return Integer
*/
function countWays($ranges) {

}
}
```

Dart:

```
class Solution {
int countWays(List<List<int>> ranges) {

}
}
```

Scala:

```
object Solution {
def countWays(ranges: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_ways(ranges :: [[integer]]) :: integer
def count_ways(ranges) do

end
end
```

Erlang:

```
-spec count_ways(Ranges :: [[integer()]]) -> integer().
count_ways(Ranges) ->
.
```

Racket:

```
(define/contract (count-ways ranges)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Ways to Group Overlapping Ranges
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int countWays(vector<vector<int>>& ranges) {

}
};
```

Java Solution:

```
/**
* Problem: Count Ways to Group Overlapping Ranges
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int countWays(int[][] ranges) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Count Ways to Group Overlapping Ranges
Difficulty: Medium
Tags: array, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Ways to Group Overlapping Ranges
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Ways to Group Overlapping Ranges
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function countWays(ranges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Count Ways to Group Overlapping Ranges
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int CountWays(int[][] ranges) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Count Ways to Group Overlapping Ranges
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int countWays(int** ranges, int rangesSize, int* rangesColSize) {

}
```

Go Solution:

```
// Problem: Count Ways to Group Overlapping Ranges
// Difficulty: Medium
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func countWays(ranges [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countWays(ranges: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func countWays(_ ranges: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Ways to Group Overlapping Ranges
// Difficulty: Medium
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn count_ways(ranges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} ranges
# @return {Integer}
def count_ways(ranges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $ranges
* @return Integer
*/
function countWays($ranges) {

}
}
```

Dart Solution:

```
class Solution {
int countWays(List<List<int>> ranges) {

}
}
```

Scala Solution:

```
object Solution {
def countWays(ranges: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_ways(ranges :: [[integer]]) :: integer
def count_ways(ranges) do

end
end
```

Erlang Solution:

```
-spec count_ways(Ranges :: [[integer()]]) -> integer().
count_ways(Ranges) ->
.
```

Racket Solution:

```
(define/contract (count-ways ranges)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```