

Problem 2406: Divide Intervals Into Minimum Number of Groups

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a 2D integer array

intervals

where

$\text{intervals}[i] = [\text{left}$

i

, right

i

]

represents the

inclusive

interval

$[\text{left}$

i

, right

i

]

You have to divide the intervals into one or more

groups

such that each interval is in

exactly

one group, and no two intervals that are in the same group

intersect

each other.

Return

the

minimum

number of groups you need to make

Two intervals

intersect

if there is at least one common number between them. For example, the intervals

[1, 5]

and

[5, 8]

intersect.

Example 1:

Input:

intervals = [[5,10],[6,8],[1,5],[2,3],[1,10]]

Output:

3

Explanation:

We can divide the intervals into the following groups: - Group 1: [1, 5], [6, 8]. - Group 2: [2, 3], [5, 10]. - Group 3: [1, 10]. It can be proven that it is not possible to divide the intervals into fewer than 3 groups.

Example 2:

Input:

intervals = [[1,3],[5,6],[8,10],[11,13]]

Output:

1

Explanation:

None of the intervals overlap, so we can put all of them in one group.

Constraints:

$1 \leq \text{intervals.length} \leq 10$

5

$\text{intervals}[i].length == 2$

$1 \leq \text{left}$

i

$\leq \text{right}$

i

≤ 10

6

Code Snippets

C++:

```
class Solution {  
public:  
    int minGroups(vector<vector<int>>& intervals) {  
        }  
    };
```

Java:

```
class Solution {  
public int minGroups(int[][][] intervals) {  
    }  
}
```

Python3:

```
class Solution:  
    def minGroups(self, intervals: List[List[int]]) -> int:
```

Python:

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

JavaScript:

```
/**  
 * @param {number[][]} intervals  
 * @return {number}  
 */  
var minGroups = function(intervals) {  
  
};
```

TypeScript:

```
function minGroups(intervals: number[][]): number {  
  
};
```

C#:

```
public class Solution {  
    public int MinGroups(int[][] intervals) {  
  
    }  
}
```

C:

```
int minGroups(int** intervals, int intervalsSize, int* intervalsColSize) {  
  
}
```

Go:

```
func minGroups(intervals [][]int) int {  
}  
}
```

Kotlin:

```
class Solution {  
    fun minGroups(intervals: Array<IntArray>): Int {  
        }  
    }  
}
```

Swift:

```
class Solution {  
    func minGroups(_ intervals: [[Int]]) -> Int {  
        }  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn min_groups(intervals: Vec<Vec<i32>>) -> i32 {  
        }  
    }  
}
```

Ruby:

```
# @param {Integer[][]} intervals  
# @return {Integer}  
def min_groups(intervals)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $intervals  
     * @return Integer
```

```
*/  
function minGroups($intervals) {  
  
}  
}  
}
```

Dart:

```
class Solution {  
int minGroups(List<List<int>> intervals) {  
  
}  
}  
}
```

Scala:

```
object Solution {  
def minGroups(intervals: Array[Array[Int]]): Int = {  
  
}  
}
```

Elixir:

```
defmodule Solution do  
@spec min_groups(intervals :: [[integer]]) :: integer  
def min_groups(intervals) do  
  
end  
end
```

Erlang:

```
-spec min_groups(Intervals :: [[integer()]]) -> integer().  
min_groups(Intervals) ->  
.
```

Racket:

```
(define/contract (min-groups intervals)  
(-> (listof (listof exact-integer?)) exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Divide Intervals Into Minimum Number of Groups
 * Difficulty: Medium
 * Tags: array, greedy, sort, queue, heap
 *
 * 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 minGroups(vector<vector<int>>& intervals) {

    }
};
```

Java Solution:

```
/**
 * Problem: Divide Intervals Into Minimum Number of Groups
 * Difficulty: Medium
 * Tags: array, greedy, sort, queue, heap
 *
 * 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 minGroups(int[][] intervals) {

    }
}
```

Python3 Solution:

```
"""
Problem: Divide Intervals Into Minimum Number of Groups
Difficulty: Medium
Tags: array, greedy, sort, queue, heap
```

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

Python Solution:

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

JavaScript Solution:

```
/**
 * Problem: Divide Intervals Into Minimum Number of Groups
 * Difficulty: Medium
 * Tags: array, greedy, sort, queue, heap
 *
 * 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
 */

var minGroups = function(intervals) {
```

```
};
```

TypeScript Solution:

```
/**  
 * Problem: Divide Intervals Into Minimum Number of Groups  
 * Difficulty: Medium  
 * Tags: array, greedy, sort, queue, heap  
 *  
 * 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 minGroups(intervals: number[][]): number {  
  
};
```

C# Solution:

```
/*  
 * Problem: Divide Intervals Into Minimum Number of Groups  
 * Difficulty: Medium  
 * Tags: array, greedy, sort, queue, heap  
 *  
 * 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 MinGroups(int[][] intervals) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Divide Intervals Into Minimum Number of Groups  
 * Difficulty: Medium
```

```

* Tags: array, greedy, sort, queue, heap
*
* 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 minGroups(int** intervals, int intervalsSize, int* intervalsColSize) {

}

```

Go Solution:

```

// Problem: Divide Intervals Into Minimum Number of Groups
// Difficulty: Medium
// Tags: array, greedy, sort, queue, heap
//
// 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 minGroups(intervals [][]int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun minGroups(intervals: Array<IntArray>): Int {
        }
    }
}
```

Swift Solution:

```

class Solution {
    func minGroups(_ intervals: [[Int]]) -> Int {
        }
    }
}
```

Rust Solution:

```
// Problem: Divide Intervals Into Minimum Number of Groups
// Difficulty: Medium
// Tags: array, greedy, sort, queue, heap
//
// 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 min_groups(intervals: Vec<Vec<i32>>) -> i32 {
        }

    }
}
```

Ruby Solution:

```
# @param {Integer[][]} intervals
# @return {Integer}
def min_groups(intervals)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $intervals
     * @return Integer
     */
    function minGroups($intervals) {

    }
}
```

Dart Solution:

```
class Solution {
    int minGroups(List<List<int>> intervals) {
```

```
}
```

```
}
```

Scala Solution:

```
object Solution {  
    def minGroups(intervals: Array[Array[Int]]): Int = {  
  
    }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec min_groups(intervals :: [[integer]]) :: integer  
  def min_groups(intervals) do  
  
  end  
end
```

Erlang Solution:

```
-spec min_groups(Intervals :: [[integer()]]) -> integer().  
min_groups(Intervals) ->  
.
```

Racket Solution:

```
(define/contract (min-groups intervals)  
  (-> (listof (listof exact-integer?)) exact-integer?)  
  )
```