Problem 3323: Minimize Connected Groups by Inserting Interval

Problem Information

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

Problem Description

You are given a 2D array

intervals

, where

intervals[i] = [start

i

, end

i

]

represents the start and the end of interval

i

. You are also given an integer

k

.

You must add

exactly one

new interval

[start

new

, end

new

]

to the array such that:

The length of the new interval,

end

new

- start

new

, is at most

k

.

After adding, the number of

connected groups

in

intervals

is

minimized

.

A

connected group

of intervals is a maximal collection of intervals that, when considered together, cover a continuous range from the smallest point to the largest point with no gaps between them. Here are some examples:

A group of intervals

[[1, 2], [2, 5], [3, 3]]

is connected because together they cover the range from 1 to 5 without any gaps.

However, a group of intervals

[[1, 2], [3, 4]]

is not connected because the segment

(2, 3)

is not covered.

Return the

minimum

number of connected groups after adding

exactly one

new interval to the array.

Example 1:

Input:

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

Output:

2

Explanation:

After adding the interval

[3, 5]

, we have two connected groups:

[[1, 3], [3, 5], [5, 6]]

and

[[8, 10]]

.

Example 2:

Input:

intervals = [[5,10],[1,1],[3,3]], k = 1

Output:

3

Explanation:

After adding the interval

[1, 1]

, we have three connected groups:

[[1, 1], [1, 1]]

,

[[3, 3]]

, and

[[5, 10]]

.

Constraints:

1 <= intervals.length <= 10

5

intervals[i] == [start

i

, end

i

]

1 <= start

i

<= end

i

<= 10

9

1 <= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minConnectedGroups(vector<vector<int>>& intervals, int k) {

}
};
```

Java:

```
class Solution {
public int minConnectedGroups(int[][] intervals, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} intervals
* @param {number} k
* @return {number}
*/
var minConnectedGroups = function(intervals, k) {

};
```

TypeScript:

```
function minConnectedGroups(intervals: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int MinConnectedGroups(int[][] intervals, int k) {

}
}
```

C:

```
int minConnectedGroups(int** intervals, int intervalsSize, int*
intervalsColSize, int k) {

}
```

Go:

```
func minConnectedGroups(intervals [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minConnectedGroups(intervals: Array<IntArray>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minConnectedGroups(_ intervals: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_connected_groups(intervals: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} intervals
# @param {Integer} k
# @return {Integer}
def min_connected_groups(intervals, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $intervals
```

```
* @param Integer $k
* @return Integer
*/
function minConnectedGroups($intervals, $k) {

}
}
```

Dart:

```
class Solution {
int minConnectedGroups(List<List<int>> intervals, int k) {

}
}
```

Scala:

```
object Solution {
def minConnectedGroups(intervals: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_connected_groups(intervals :: [[integer]], k :: integer) :: integer
def min_connected_groups(intervals, k) do

end
end
```

Erlang:

```
-spec min_connected_groups(Intervals :: [[integer()]], K :: integer()) ->
integer().
min_connected_groups(Intervals, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimize Connected Groups by Inserting Interval
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 minConnectedGroups(vector<vector<int>>& intervals, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize Connected Groups by Inserting Interval
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 minConnectedGroups(int[][] intervals, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimize Connected Groups by Inserting Interval
Difficulty: Medium
Tags: array, sort, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimize Connected Groups by Inserting Interval
* Difficulty: Medium
* Tags: array, sort, search
*
* 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[][]} intervals
* @param {number} k
* @return {number}
*/
var minConnectedGroups = function(intervals, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize Connected Groups by Inserting Interval
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 minConnectedGroups(intervals: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimize Connected Groups by Inserting Interval
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 MinConnectedGroups(int[][] intervals, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimize Connected Groups by Inserting Interval
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 minConnectedGroups(int** intervals, int intervalsSize, int*
intervalsColSize, int k) {

}
```

Go Solution:

```
// Problem: Minimize Connected Groups by Inserting Interval
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 minConnectedGroups(intervals [][]int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minConnectedGroups(intervals: Array<IntArray>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minConnectedGroups(_ intervals: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimize Connected Groups by Inserting Interval
// Difficulty: Medium
// Tags: array, sort, search
//
// 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_connected_groups(intervals: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} intervals
# @param {Integer} k
# @return {Integer}
def min_connected_groups(intervals, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $intervals
* @param Integer $k
* @return Integer
*/
```

```
function minConnectedGroups($intervals, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minConnectedGroups(List<List<int>> intervals, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minConnectedGroups(intervals: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_connected_groups(intervals :: [[integer]], k :: integer) :: integer
def min_connected_groups(intervals, k) do

end
end
```

Erlang Solution:

```
-spec min_connected_groups(Intervals :: [[integer()]], K :: integer()) ->
integer().
min_connected_groups(Intervals, K) ->
.
```

Racket Solution:

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

```
)
```