Problem 2402: Meeting Rooms III

Problem Information

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

Problem Description

You are given an integer

n

. There are

n

rooms numbered from

0

to

n - 1

.

You are given a 2D integer array

meetings

where

meetings[i] = [start

i

, end

i

]

means that a meeting will be held during the

half-closed

time interval

[start

i

, end

i

)

. All the values of

start

i

are

unique

.

Meetings are allocated to rooms in the following manner:

Each meeting will take place in the unused room with the lowest

number.

If there are no available rooms, the meeting will be delayed until a room becomes free. The delayed meeting should have the

same

duration as the original meeting.

When a room becomes unused, meetings that have an earlier original

start

time should be given the room.

Return

the

number

of the room that held the most meetings.

If there are multiple rooms, return

the room with the

lowest

number.

A

half-closed interval

[a, b)

is the interval between

a

and

b

including

a

and

not including

b

.

Example 1:

Input:

n = 2, meetings = [[0,10],[1,5],[2,7],[3,4]]

Output:

0

Explanation:

- At time 0, both rooms are not being used. The first meeting starts in room 0. - At time 1, only room 1 is not being used. The second meeting starts in room 1. - At time 2, both rooms are being used. The third meeting is delayed. - At time 3, both rooms are being used. The fourth meeting is delayed. - At time 5, the meeting in room 1 finishes. The third meeting starts in room 1 for the time period [5,10). - At time 10, the meetings in both rooms finish. The fourth meeting starts in room 0 for the time period [10,11). Both rooms 0 and 1 held 2 meetings, so we return 0.

Example 2:

Input:

n = 3, meetings = [[1,20],[2,10],[3,5],[4,9],[6,8]]

Output:

1

Explanation:

- At time 1, all three rooms are not being used. The first meeting starts in room 0. - At time 2, rooms 1 and 2 are not being used. The second meeting starts in room 1. - At time 3, only room 2 is not being used. The third meeting starts in room 2. - At time 4, all three rooms are being used. The fourth meeting is delayed. - At time 5, the meeting in room 2 finishes. The fourth meeting starts in room 2 for the time period [5,10). - At time 6, all three rooms are being used. The fifth meeting is delayed. - At time 10, the meetings in rooms 1 and 2 finish. The fifth meeting starts in room 1 for the time period [10,12). Room 0 held 1 meeting while rooms 1 and 2 each held 2 meetings, so we return 1.

Constraints:

1 <= n <= 100

1 <= meetings.length <= 10

5

meetings[i].length == 2

0 <= start

i

< end

i

<= 5 * 10

5

All the values of

start

i

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int mostBooked(int n, vector<vector<int>>& meetings) {

}
};
```

Java:

```
class Solution {
public int mostBooked(int n, int[][] meetings) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function mostBooked(n: number, meetings: number[][]): number {

};
```

C#:

```
public class Solution {
public int MostBooked(int n, int[][] meetings) {

}
}
```

C:

```
int mostBooked(int n, int** meetings, int meetingsSize, int* meetingsColSize)
{

}
```

Go:

```
func mostBooked(n int, meetings [][]int) int {

}
```

Kotlin:

```
class Solution {
fun mostBooked(n: Int, meetings: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} meetings
# @return {Integer}
def most_booked(n, meetings)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $meetings
* @return Integer
*/
function mostBooked($n, $meetings) {

}
}
```

Dart:

```
class Solution {
int mostBooked(int n, List<List<int>> meetings) {

}
}
```

Scala:

```
object Solution {
def mostBooked(n: Int, meetings: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec most_booked(n :: integer, meetings :: [[integer]]) :: integer
def most_booked(n, meetings) do

end
end
```

Erlang:

```
-spec most_booked(N :: integer(), Meetings :: [[integer()]]) -> integer().
most_booked(N, Meetings) ->
.
```

Racket:

```
(define/contract (most-booked n meetings)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Meeting Rooms III
* Difficulty: Hard
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int mostBooked(int n, vector<vector<int>>& meetings) {

}
};
```

Java Solution:

```
/**
* Problem: Meeting Rooms III
* Difficulty: Hard
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int mostBooked(int n, int[][] meetings) {

}
}
```

Python3 Solution:

```
"""
Problem: Meeting Rooms III
Difficulty: Hard
Tags: array, hash, sort, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Meeting Rooms III
* Difficulty: Hard
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Meeting Rooms III
* Difficulty: Hard
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function mostBooked(n: number, meetings: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Meeting Rooms III
* Difficulty: Hard
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MostBooked(int n, int[][] meetings) {

}
}
```

C Solution:

```
/*
* Problem: Meeting Rooms III
* Difficulty: Hard
* Tags: array, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int mostBooked(int n, int** meetings, int meetingsSize, int* meetingsColSize)
{

}
```

Go Solution:

```
// Problem: Meeting Rooms III
// Difficulty: Hard
// Tags: array, hash, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func mostBooked(n int, meetings [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun mostBooked(n: Int, meetings: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Meeting Rooms III
// Difficulty: Hard
// Tags: array, hash, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn most_booked(n: i32, meetings: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} meetings
# @return {Integer}
def most_booked(n, meetings)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $meetings
* @return Integer
*/
function mostBooked($n, $meetings) {

}
}
```

Dart Solution:

```
class Solution {
int mostBooked(int n, List<List<int>> meetings) {

}
}
```

Scala Solution:

```
object Solution {
def mostBooked(n: Int, meetings: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec most_booked(n :: integer, meetings :: [[integer]]) :: integer
def most_booked(n, meetings) do

end
end
```

Erlang Solution:

```
-spec most_booked(N :: integer(), Meetings :: [[integer()]]) -> integer().
most_booked(N, Meetings) ->
.
```

Racket Solution:

```
(define/contract (most-booked n meetings)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```