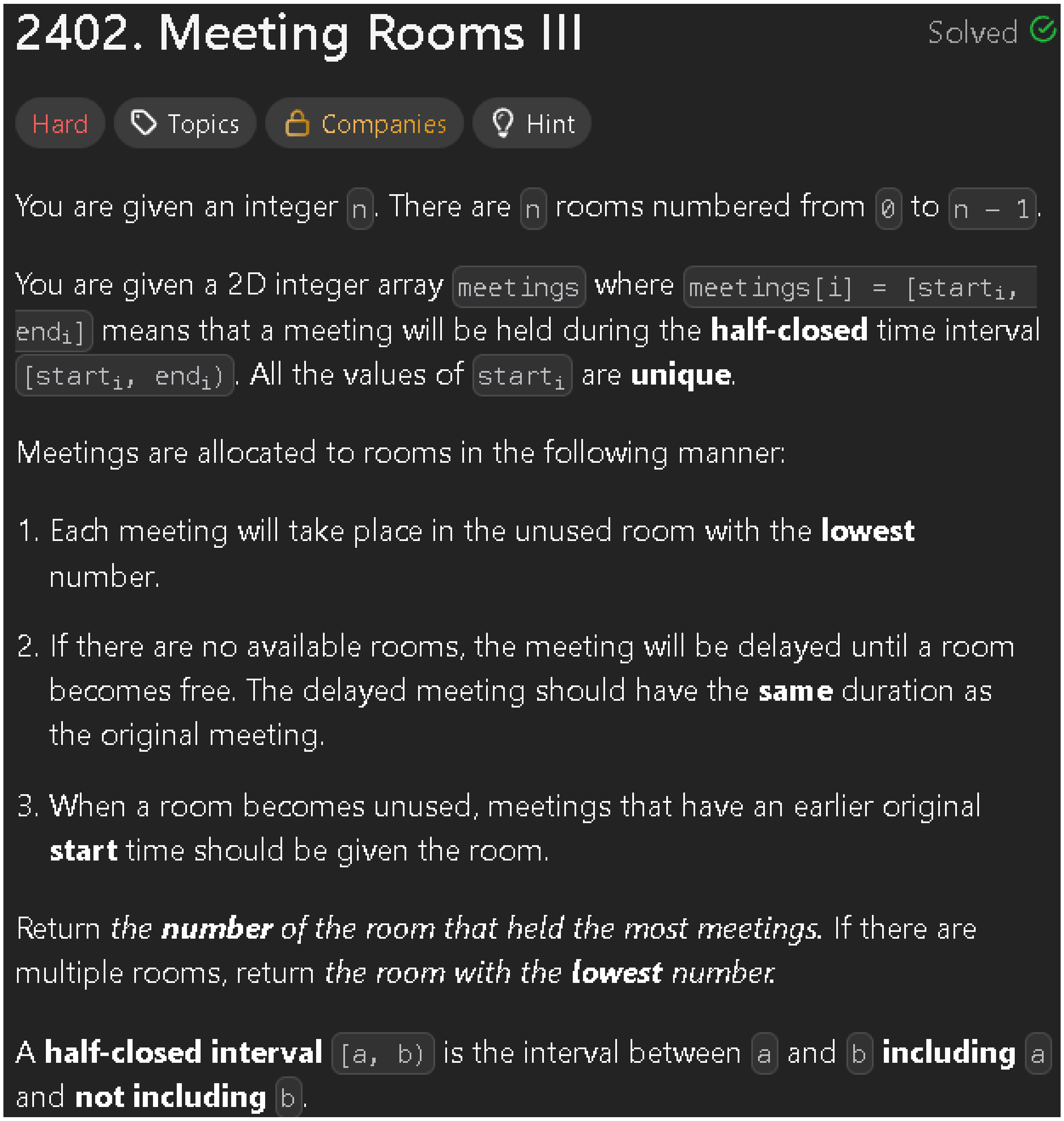


2402. Meeting Rooms III

Solved

Hard Topics Companies Hint

You are given an integer `n`. There are `n` rooms numbered from `0` to `n - 1`.

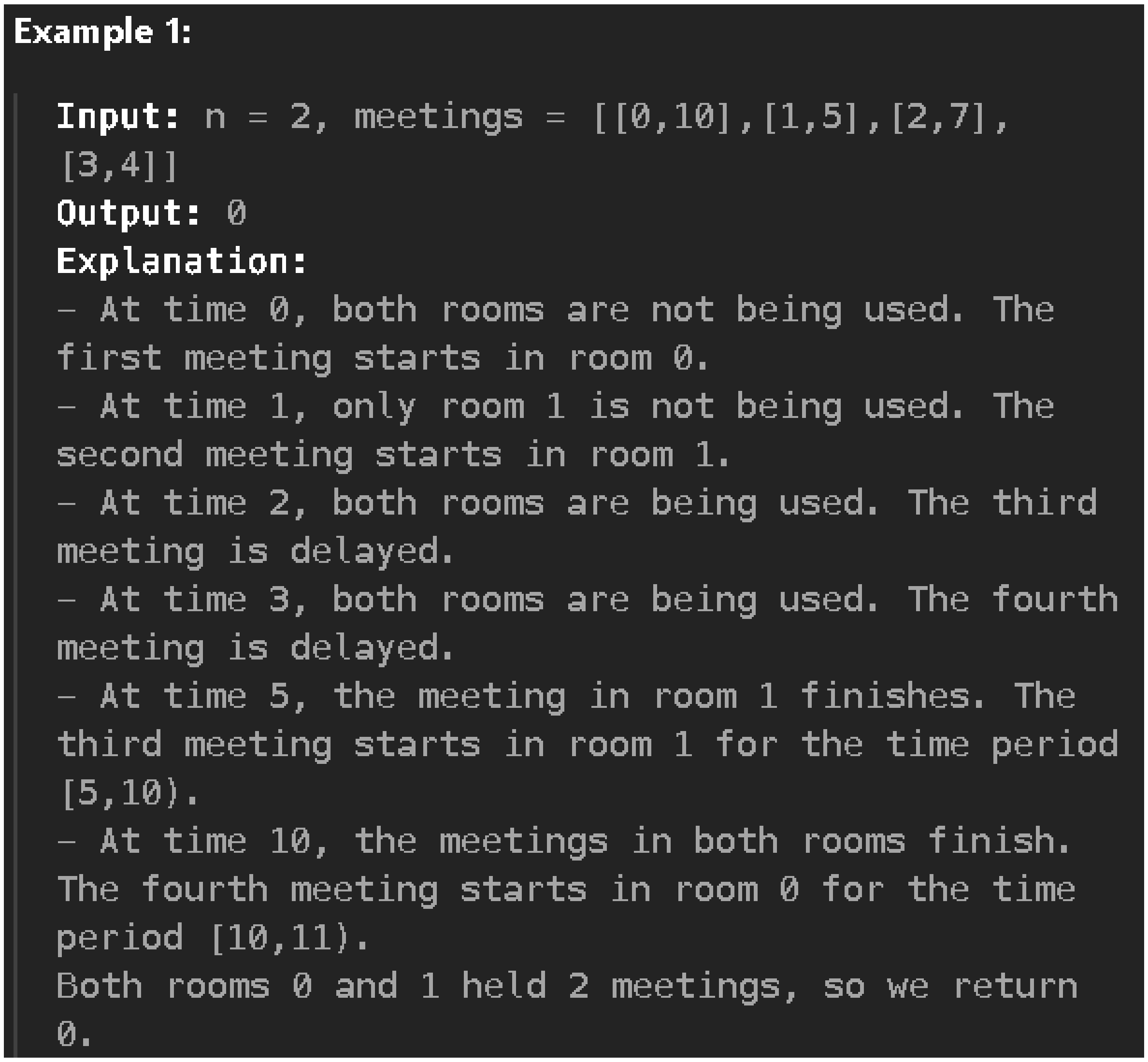
You are given a 2D integer array `meetings` where $\mathtt{meetings[i]} = [\mathtt{start}_i, \mathtt{end}_i]$ means that a meeting will be held during the **half-closed** time interval $[\mathtt{start}_i, \mathtt{end}_i)$. All the values of $\mathtt{start}_i$ are **unique**.

Meetings are allocated to rooms in the following manner:

1. Each meeting will take place in the unused room with the **lowest** number.
2. 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.
3. When a room becomes unused, meetings that have an earlier original **start** time should be given the room.

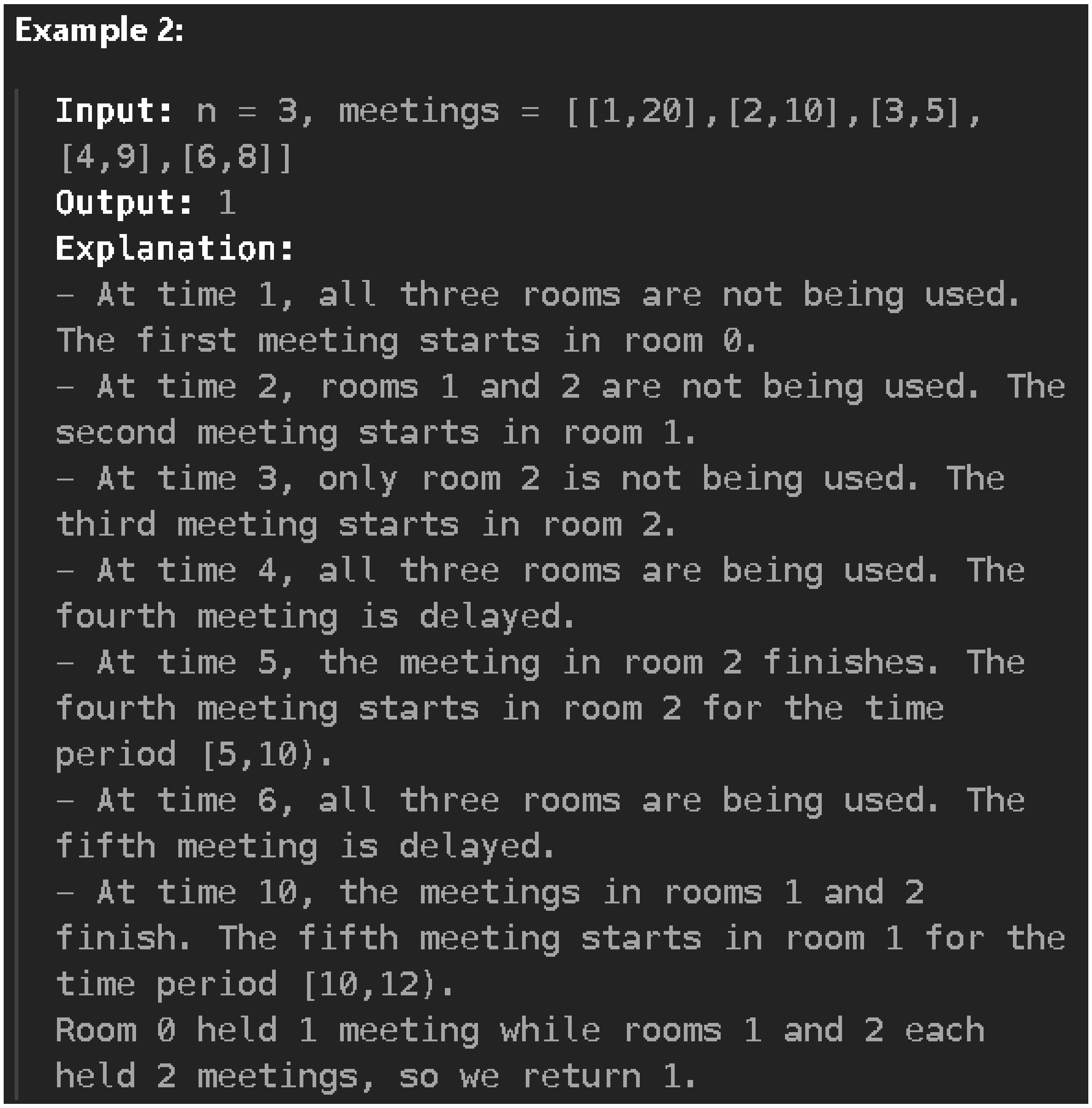
Return *the **number** of the room that held the most meetings.* If there are multiple rooms, return *the room with the **lowest** number.*

A **half-closed interval** `[a, b)` is the interval between `a` and `b` **including** `a` and **not including** `b`.



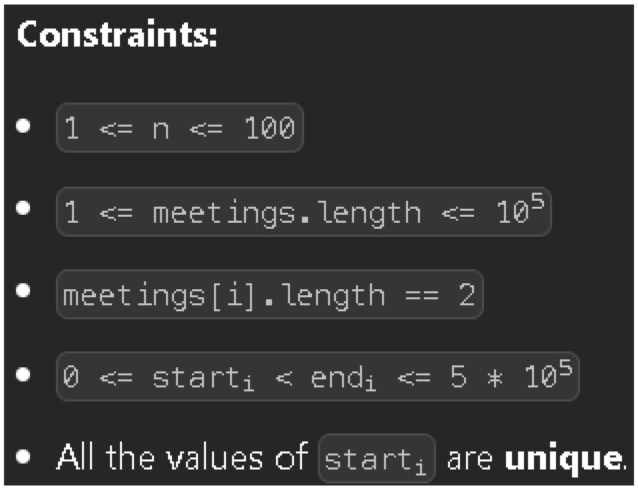
Example 1:

```
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**.

Python:

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

        count = [0] * n
        timer = [0] * n

        itr = 0

        while itr < len(meetings):
            start, end = meetings[itr]
            dur = end - start

            room = -1
            earliest = 10**18
            earliestRoom = -1

            for i in range(n):
                if timer[i] < earliest:
                    earliest = timer[i]
                    earliestRoom = i
                if timer[i] <= start:
                    room = i
                    break

            if room != -1:
                timer[room] = end
                count[room] += 1
            else:
                timer[earliestRoom] += dur
                count[earliestRoom] += 1

            itr += 1

        maxv = 0
        idx = 0
        for i in range(n):
            if count[i] > maxv:
                maxv = count[i]
                idx = i

        return idx
```

JavaScript:

```
var mostBooked = function(n, meetings) {
    meetings.sort((a, b) => a[0] - b[0]);

    const count = new Array(n).fill(0);
    const timer = new Array(n).fill(0);

    let itr = 0;

    while (itr < meetings.length) {
        const start = meetings[itr][0];
        const end = meetings[itr][1];
        const dur = end - start;

        let room = -1;
        let earliest = Number.MAX_SAFE_INTEGER;
        let earliestRoom = -1;

        for (let i = 0; i < n; i++) {
            if (timer[i] < earliest) {
                earliest = timer[i];
                earliestRoom = i;
            }
            if (timer[i] <= start) {
                room = i;
                break;
            }
        }

        if (room !== -1) {
            timer[room] = end;
            count[room]++;
        } else {
            timer[earliestRoom] += dur;
            count[earliestRoom]++;
        }

        itr++;
    }

    let max = 0, idx = 0;
    for (let i = 0; i < n; i++) {
        if (count[i] > max) {
            max = count[i];
            idx = i;
        }
    }

    return idx;
};
```

Java:

```
class Solution {
    public int mostBooked(int n, int[][] meetings) {
        Arrays.sort(meetings, (a, b) -> a[0] - b[0]);

        int[] count = new int[n];
        long[] timer = new long[n];

        int itr = 0;

        while (itr < meetings.length) {
            int[] curr = meetings[itr];
            int start = curr[0];
            int end = curr[1];
            long dur = end - start;

            int room = -1;
            long earliest = Long.MAX_VALUE;
            int earliestRoom = -1;

            for (int i = 0; i < n; i++) {
                if (timer[i] < earliest) {
                    earliest = timer[i];
                    earliestRoom = i;
                }
                if (timer[i] <= start) {
                    room = i;
                    break;
                }
            }

            if (room != -1) {
                timer[room] = end;
                count[room]++;
            } else {
                timer[earliestRoom] += dur;
                count[earliestRoom]++;
            }

            itr++;
        }

        int max = 0, idx = 0;
        for (int i = 0; i < n; i++) {
            if (count[i] > max) {
                max = count[i];
                idx = i;
            }
        }

        return idx;
    }
}
```