CP INTERNAL PRACTICAL

23DCS040

Q: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:

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.

CODE:

```
import java.util.*;
public class cp {


   public static int mostbooked(int n, int meetings[][])
   {
       Arrays.sort(meetings ,(a,b)-> Integer.compare(a[0], b[0]));

      int[] endTime = new int[n];
       int[] count = new int[n];



      for (int[] mt : meetings) {
          int start = mt[0];
          int duration = mt[1] - mt[0];


          int chosen = -1;
          for (int i = 0; i < n; i++) {
              if (endTime[i] <= start) {
                  chosen = i;
                  break;
              }
          }
          if (chosen != -1) {

              endTime[chosen] = start + duration;
              count[chosen]++;
          } else {

              int earliestroom = 0;
              for (int i = 1; i < n; i++) {
                  if (endTime[i] < endTime[earliestroom]) earliestroom = i;
              }

              int newStart = endTime[earliestroom];
              endTime[earliestroom] = newStart + duration;
              count[earliestroom]++;
          }
      }
       int best = 0;
      for (int i = 1; i < n; i++) {
          if (count[i] > count[best]) best = i;
      }
      return best;
```

```
    }
    public static void main(String[]args)
    {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number of rooms: ");
        int n = sc.nextInt();


        System.out.print("Enter number of meetings: ");
        int m = sc.nextInt();

         int[][] meetings = new int[m][2];
        System.out.println("Enter meetings (start end):");
        for (int i = 0; i < m; i++) {
            meetings[i][0] = sc.nextInt();
            meetings[i][1] = sc.nextInt();
        }
        int result = mostbooked(n, meetings);
      System.out.println("number of rooms that held most meetings is :"+ result);
    }



}
```

OUTPUT:

```
Enter number of rooms: 2
Enter number of meetings: 4
Enter meetings (start end):
0 10
1 5
2 7
3 4
number of rooms that held most meetings is :0
```

```
Enter number of rooms: 3
Enter number of meetings: 5
Enter meetings (start end):
1 20
2 10
3 5
4 9
6 8
number of rooms that held most meetings is :1
```

Q:2

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Examples:

Example 1:

- Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
- Output: 6
- Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

- Input: height = [4,2,0,3,2,5]
- Output: 9

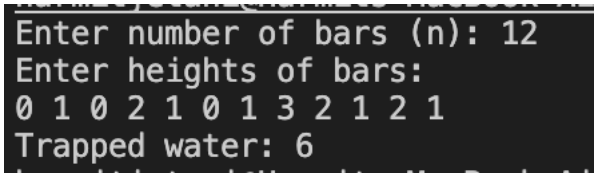
Constraints:

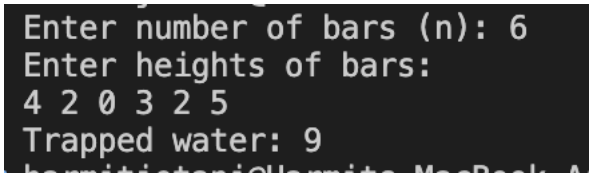
- n == height.length
- $1 \leq n \leq 2 \times 10^4$
- $0 \leq height[i] \leq 10^5$

CODE:

```java
import java.util.*;
public class cp1 {


    public static int trap(int[] height) {
        if (height == null || height.length == 0) return 0;
        int left = 0, right = height.length - 1;
        int leftMax = 0, rightMax = 0, ans = 0;
        while (left <= right) {
            if (height[left] <= height[right]) {

                if (height[left] >= leftMax) leftMax = height[left];

                else ans += leftMax - height[left];
                left++;
            } else {

                if (height[right] >= rightMax) rightMax = height[right];

                else ans += rightMax - height[right];
                right--;
            }
        }
        return ans;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number of bars (n): ");
        int n = sc.nextInt();
        int[] height = new int[n];
        System.out.println("Enter heights of bars:");
        for (int i = 0; i < n; i++)
        {
            height[i] = sc.nextInt();
        }
        System.out.println("Trapped water: " + trap(height));
    }
}
```

OUTPUT:


Enter number of bars (n): 12
Enter heights of bars:
0 1 0 2 1 0 1 3 2 1 2 1
Trapped water: 6


Enter number of bars (n): 6
Enter heights of bars:
4 2 0 3 2 5
Trapped water: 9