My Calendar III

Implement a `MyCalendarThree` class to store your events. A new event can **always** be added.

Your class will have one method, `book(int start, int end)`. Formally, this represents a booking on the half open interval `[start, end)`, the range of real numbers `x` such that `start .`

A *K-booking* happens when **K** events have some non-empty intersection (ie., there is some time that is common to all K events.)

For each call to the method `MyCalendar.book`, return an integer `K` representing the largest integer such that there exists a `K`-booking in the calendar.

Your class will be called like this: `MyCalendarThree cal = new MyCalendarThree(); MyCalendarThree.book(start, end)`

Example 1:

```
MyCalendarThree();
MyCalendarThree.book(10, 20); // returns 1
MyCalendarThree.book(50, 60); // returns 1
MyCalendarThree.book(10, 40); // returns 2
MyCalendarThree.book(5, 15); // returns 3
MyCalendarThree.book(5, 10); // returns 3
MyCalendarThree.book(25, 55); // returns 3
Explanation:
The first two events can be booked and are disjoint, so the maximum K-booking is a 1-booking.
The third event [10, 40) intersects the first event, and the maximum K-booking is a 2-booking.
The remaining events cause the maximum K-booking to be only a 3-booking.
Note that the last event locally causes a 2-booking, but the answer is still 3 because
eg. [10, 20), [10, 40), and [5, 15) are still triple booked.
```

Note:

- The number of calls to `MyCalendarThree.book` per test case will be at most `1000`.
- In calls to `MyCalendarThree.book(start, end)`, `start` and `end` are integers in the range `[0, 10^9]`.

Solution 1

The logic is: let's walk though the start and end time points one by one in sorting order. If the point is start, increase one. If the point is end, decrease one. The sum is always greater or equal than 0, and it is the overlap number between the previous time to the next time.

This method can be used to solve My Calendar I and II as well.

```
class MyCalendarThree {
public:
    MyCalendarThree() {

    }

    int book(int start, int end) {
        m[start]++;
        m[end]--;
        int res = 0;
        int cur = 0;
        for (auto & event: m)
        {
            cur += event.second;
            if (cur > res)
            {
                res = cur;
            }
        }
        return res;
    }
private:
    map<int, int> m;
};

/**
 * Your MyCalendarThree object will be instantiated and called as such:
 * MyCalendarThree obj = new MyCalendarThree();
 * int param_1 = obj.book(start,end);
 */
```

written by cygnus original link here

Solution 2

Summarize

`This is to find the maximum number of concurrent ongoing event at any time.`

We can log the `start` & `end` of each event on the timeline, each `start` add a new ongoing event at that time, each `end` terminate an ongoing event. Then we can scan the timeline to figure out the maximum number of ongoing event at any time.

Java

```
class MyCalendarThree {
    private TreeMap<Integer, Integer> times = new TreeMap<>();
    public int book(int s, int e) {
        times.put(s, times.getOrDefault(s, 0) + 1); // 1 new event will be starting
at times[s]
        times.put(e, times.getOrDefault(e, 0) - 1); // 1 new event will be ending a
t times[e];
        int ongoing = 0, k = 0;
        for (int v : times.values())
            k = Math.max(k, ongoing += v);
        return k;
    }
}
```

C++

```
class MyCalendarThree {
    map<int, int> times;
public:
    int book(int s, int e) {
        times[s]++; // 1 new event will be starting at times[s]
        times[e]--; // 1 new event will be ending at times[e];
        int ongoing = 0, k = 0;
        for (pair<int, int> t : times)
            k = max(k, ongoing += t.second);
        return k;
    }
};
```

written by alexander original link here

Solution 3

The logic is same as My Calendar. Just treat start and end separately.
Traversal whole tree and save the biggest count.

```
class MyCalendarThree {
    class Node{
        private int k, v;
        private Node left, right;

        public Node(int k, int v) {
            this.k = k;
            this.v = v;
        }
    }

    private Node root;
    private int curt, count;

    public MyCalendarThree() {

    }

    private Node insert(Node node, int k, int v) {
        if (node == null) {
            node = new Node(k, v);
            return node;
        } else if (node.k == k) {
            node.v += v;
        } else if (node.k < k) {
            node.right = insert(node.right, k, v);
        } else {
            node.left = insert(node.left, k, v);
        }
        return node;
    }

    private void count(Node node) {
        if (node == null) {
            return;
        }
        count(node.left);
        curt += node.v;
        count = Math.max(count, curt);
        count(node.right);
    }

    public int book(int start, int end) {
        root = insert(root, start, 1);
        root = insert(root, end, -1);
        curt = count = 0;
        count(root);
        return count;
    }
}
```

written by harper07 original link here

From Leetcoder.