729. My Calendar I

You are implementing a program to use as your calendar. We can add a new event if adding the event will not cause a **double booking**.

A **double booking** happens when two events have some non-empty intersection (i.e., some moment is common to both events.).

The event can be represented as a pair of integers `start` and `end` that represents a booking on the half-open interval `[start, end)`, the range of real numbers `x` such that `start <= x < end`.

Implement the `MyCalendar` class:

- `MyCalendar()` Initializes the calendar object.
- `boolean book(int start, int end)` Returns `true` if the event can be added to the calendar successfully without causing a **double booking**. Otherwise, return `false` and do not add the event to the calendar.

Example 1:

```
Input
["MyCalendar", "book", "book", "book"]
[[], [10, 20], [15, 25], [20, 30]]
Output
[null, true, false, true]

Explanation
MyCalendar myCalendar = new MyCalendar();
myCalendar.book(10, 20); // return True
myCalendar.book(15, 25); // return False, It can not be booked because time 15 is
already booked by another event.
myCalendar.book(20, 30); // return True, The event can be booked, as the first
event takes every time less than 20, but not including 20.
```

Constraints:

- $0 <= start < end <= 10^9$
- At most `1000` calls will be made to `book`.

729. My Calendar I

/*

Brute force: for each book query,
check if the new event (start to end) overlap or not
with all previous events.

Time complexity: $O\left(n^2\right)$
Space complexity: $O\left(n\right)$
n : total number of events

*/

```
class MyCalendar {
    private:
        // Store all events
        std::vector<std::pair<int,int>> booked;
    public:
        MyCalendar() {

        }

        // Could be called n times
        bool book(int start, int end) {
            // For each previous event (s to e)...
            for(auto& [s,e]: booked){
                // if it overlap with the new event
                // (start to end), don't schedule the new event
                if(s<end && e>start) return false;
            }

            // Otherwise, add the new event to the list
            booked.push_back({start,end});

            // Accept scheduling the new event.
            return true;
        }
    };
```

729. My Calendar I

```
/*
    Active Interval counting
    Time complexity: O(nm)
    Space complexity: O(m)
    n: total number of events
    m: number of starting points and and points
*/
class MyCalendar {
map<int,int> dp;
public:
    MyCalendar() {

    }

    bool book(int start, int end) {
        dp[start]++;
        dp[end]--;
        int s = 0;
        for (auto v: dp){
            s += v.second;
            if (s >= 2) {
                dp[start]--;
                dp[end]++;
                return false;
            }
        }
        return true;
    }
};
```

729. My Calendar I

/*

Binary search: for each book query,
check if the new event (start to end) overlap or not
with all previous events.

Time complexity: $O(nlogn)$
Space complexity: $O(n)$
n : total number of events

*/

```
class MyCalendar {
    private:
        std::map<int,int> booked;
    public:
        MyCalendar() {


        }


        bool book(int start, int end) {
           //Find next event
           auto next=booked.lower_bound(start);

           // If next event overlaps with the next one, don't
           //schedule it
           if(next!=booked.end() && next->first<end) return false;

           // If next event overlaps with the next one, don't
          // schedule it
           if(next!=booked.begin() && std::prev(next)->second>start)
                                  return false;


           // Otherwise, schedule the new event (from start to end)
           booked[start]=end;
           return true;
        }
    };
```