My Calendar I

Implement a `MyCalendar` class to store your events. A new event can be added if adding the event will not cause a double booking.

Your class will have the 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 *double booking* happens when two events have some non-empty intersection (ie., there is some time that is common to both events.)

For each call to the method `MyCalendar.book`, return `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.

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

Example 1:

```
MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(15, 25); // returns false
MyCalendar.book(20, 30); // returns true
Explanation:
The first event can be booked.  The second can't because time 15 is already booked by
 another event.
The third event can be booked, as the first event takes every time less than 20, but
not including 20.
```

Note:

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

Solution 1

```
class MyCalendar {

    TreeMap<Integer, Integer> calendar;

    public MyCalendar() {
        calendar = new TreeMap<>();
    }

    public boolean book(int start, int end) {
        Integer floorKey = calendar.floorKey(start);
        if (floorKey != null && calendar.get(floorKey) > start) return false;
        Integer ceilingKey = calendar.ceilingKey(start);
        if (ceilingKey != null && ceilingKey < end) return false;

        calendar.put(start, end);
        return true;
    }
}
```

written by shawngao original link here

Solution 2

Solution 1: Check every existed book for overlap

`overlap` of 2 interval `a` `b` is `(max(a0, b0), min(a1, b1))`
detail is in: https://discuss.leetcode.com/topic/111198

Java

```
class MyCalendar {
    private List<int[]> books = new ArrayList<>();
    public boolean book(int start, int end) {
        for (int[] b : books)
            if (Math.max(b[0], start) < Math.min(b[1], end)) return false;
        books.add(new int[]{ start, end });
        return true;
    }
}
```

C++

```
class MyCalendar {
    vector<pair<int, int>> books;
public:
    bool book(int start, int end) {
        for (pair<int, int> p : books)
            if (max(p.first, start) < min(end, p.second)) return false;
        books.push_back({start, end});
        return true;
    }
};
```

Solution 2: Keep existing books sorted and only check 2 books start right before & after the new book starts

Another way to check overlap of 2 intervals is `a started with b`, or, `b started within a`.

Keep the intervals sorted,
if the interval started right before the new interval contains the start, or
if the interval started right after the new interval started within the `new interval`.

```
   floor      ceiling
... |----| ... |----| ...
       |---------|
      s         e
if s < floor.end or e > ceiling.start, there is an overlap.

Another way to think of it:
If there is an interval start within the new book (must be the ceilingEntry) at all,
or
books: |----|   |--|
            s |------| e

books: |----|   |----|
            s |----| e
If the new book start within an interval (must be the floorEntry) at all
books: |-------|   |--|
       s |---| e

books: |----|   |----|
        s |----| e
There is a overlap
```

Java

TreeSet

```
class MyCalendar {
    TreeSet<int[]> books = new TreeSet<int[]>((int[] a, int[] b) -> a[0] - b[0]);

    public boolean book(int s, int e) {
        int[] book = new int[] { s, e }, floor = books.floor(book), ceiling = books
.ceiling(book);
        if (floor != null && s < floor[1]) return false; // (s, e) start within flo
or
        if (ceiling != null && ceiling[0] < e) return false; // ceiling start withi
n (s, e)
        books.add(book);
        return true;
    }
}
```

TreeMap

```
class MyCalendar {
    TreeMap<Integer, Integer> books = new TreeMap<>();

    public boolean book(int s, int e) {
        java.util.Map.Entry<Integer, Integer> floor = books.floorEntry(s), ceiling
= books.ceilingEntry(s);
        if (floor != null && s < floor.getValue()) return false; // (s, e) start wi
thin floor
        if (ceiling != null && ceiling.getKey() < e) return false; // ceiling start
within (s, e)
        books.put(s, e);
        return true;
    }
}
```

C++

ordered set

```
class MyCalendar {
    set<pair<int, int>> books;
public:
    bool book(int s, int e) {
        auto next = books.lower_bound({s, e}); // first element with key not go befo
re k (i.e., either it is equivalent or goes after).
        if (next != books.end() && next->first < e) return false; // a existing boo
k started within the new book (next)
        if (next != books.begin() && s < (--next)->second) return false; // new boo
k started within a existing book (prev)
        books.insert({ s, e });
        return true;
    }
};
```

ordered map

```
class MyCalendar {
    map<int, int> books;
public:
    bool book(int s, int e) {
        auto next = books.lower_bound(s); // first element with key not go before k
(i.e., either it is equivalent or goes after).
        if (next != books.end() && next->first < e) return false; // a existing boo
k started within the new book (next)
        if (next != books.begin() && s < (--next)->second) return false; // new boo
k started within a existing book (prev)
        books[s] = e;
        return true;
    }
};
```

written by alexander original link here

Solution 3

```
class Node:
    def __init__(self,s,e):
        self.e = e
        self.s = s
        self.left = None
        self.right = None


class MyCalendar(object):

    def __init__(self):
        self.root = None

    def book_helper(self,s,e,node):
        if s>=node.e:
            if node.right:
                return self.book_helper(s,e,node.right)
            else:
                node.right = Node(s,e)
                return True
        elif e<=node.s:
            if node.left:
                return self.book_helper(s,e,node.left)
            else:
                node.left = Node(s,e)
                return True
        else:
            return False

    def book(self, start, end):
        """
        :type start: int
        :type end: int
        :rtype: bool
        """
        if not self.root:
            self.root = Node(start,end)
            return True
        return self.book_helper(start,end,self.root)


        ```
```

written by persianPanda original link here
```