My Calendar II

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

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 *triple booking* happens when **three** events have some non-empty intersection (ie., there is some time that is common to all 3 events.)

For each call to the method `MyCalendar.book`, return `true` if the event can be added to the calendar successfully without causing a **triple** 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(50, 60); // returns true
MyCalendar.book(10, 40); // returns true
MyCalendar.book(5, 15); // returns false
MyCalendar.book(5, 10); // returns true
MyCalendar.book(25, 55); // returns true
Explanation:
The first two events can be booked.  The third event can be double booked.
The fourth event (5, 15) can't be booked, because it would result in a triple booking
.
The fifth event (5, 10) can be booked, as it does not use time 10 which is already do
uble booked.
The sixth event (25, 55) can be booked, as the time in [25, 40) will be double booked
 with the third event;
the time [40, 50) will be single booked, and the time [50, 55) will be double booked
with the second event.
```

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

The big idea is pretty simple:
Each time of `book`, instead of `fail` a book when there is 1 or more `overlap` with existing books as in `MyCalendar I`, we just want to make sure these overlaps does not overlap - having `overlap` is now ok, but `overlapped` period cannot be `overlapped` again.
So we just need to keep track of all the `overlaps` with any previous `books`

`MyCalendar I` can be reused to track the `overlaps` during each book.

How to calculate overlap of 2 intervals

Assume `a` start earlier than `b`, (if not reverse), there could be 3 case, but in any case, an overlap(either positive or negative) can always be represented as:
`(max(a0, b0), min(a1, b1))`

```
case 1: b ends before a ends:
a: a0 |-------------| a1
b:     b0 |-----| b1

case 2: b ends after a ends:
a: a0 |--------| a1
b:     b0 |--------| b1

case 3: b starts after a ends: (negative overlap)
a: a0 |----| a1
b:              b0 |----| b1
```

Java

```java
class MyCalendarTwo {
    private List<int[]> books = new ArrayList<>();
    public boolean book(int s, int e) {
        MyCalendar overlaps = new MyCalendar();
        for (int[] b : books)
            if (Math.max(b[0], s) < Math.min(b[1], e)) // overlap exist
                if (!overlaps.book(Math.max(b[0], s), Math.min(b[1], e))) return fa
lse; // overlaps overlapped
        books.add(new int[]{ s, e });
        return true;
    }

    private static class MyCalendar {
        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;
    }
};

class MyCalendarTwo {
    vector<pair<int, int>> books;
public:
    bool book(int start, int end) {
        MyCalendar overlaps;
        for (pair<int, int> p : books) {
            if (max(p.first, start) < min(end, p.second)) { // overlap exist
                pair<int, int> overlapped = getOverlap(p.first, p.second, start, en
d);
                if (!overlaps.book(overlapped.first, overlapped.second)) return fal
se; // overlaps overlapped
            }
        }
        books.push_back({ start, end });
        return true;
    }

    pair<int, int> getOverlap(int s0, int e0, int s1, int e1) {
        return { max(s0, s1), min(e0, e1)};
    }
};
```

Another way to calculate overlap of 2 intervals

`a started with b`, or, `b started within a`:

```
a:                    |---------|
b:
a0<b0 & a1<b0:  |----|
a0<b0 & a1>b0:  |------------| (a started within b)
a0<b0 & a1>b1:  |--------------------| (a started within b)
a0>b0 & a0<b1:            |----|  (b started within a)
a0>b0 & a0>b1:            |---------| (b started within a)
a0>b1 & a1>b1:                      |----|
```

written by alexander original link here

Solution 2

We store an array `self.overlaps` of intervals that are double booked, and `self.calendar` for intervals which have been single booked. We use the line `start < j and end > i` to check if the ranges `[start, end)` and `[i, j)` overlap.

The clever idea is we do not need to "clean up" ranges in `calendar`: if we have `[1, 3]` and `[2, 4]`, this will be `calendar = [[1,3],[2,4]]` and `overlaps = [[2,3]]`. We don't need to spend time transforming the calendar to `calendar = [[1,4]]`.

This solution is by @zestypanda .

```
class MyCalendarTwo:
    def __init__(self):
        self.overlaps = []
        self.calendar = []

    def book(self, start, end):
        for i, j in self.overlaps:
            if start < j and end > i:
                return False
        for i, j in self.calendar:
            if start < j and end > i:
                self.overlaps.append((max(start, i), min(end, j)))
        self.calendar.append((start, end))
        return True
```

written by awice original link here

Solution 3

```
public class MyCalendarTwo {

    private List<int[]> list = new ArrayList<>();

    public boolean book(int start, int end) {
        MyCalendar c = new MyCalendar();
        for (int[] i : list) {
            if (i[0] < start && i[1] > start) {
                if (!c.book(start, i[1])) {
                    return false;
                }
            } else if (i[0] >= start && i[0] < end) {
                if (!c.book(i[0], Math.min(i[1], end))) {
                    return false;
                }
            }
        }
        list.add(new int[] {start, end});
        return true;
    }

    private class MyCalendar {

        TreeMap<Integer, Integer> tm = new TreeMap<>();

        public boolean book(int start, int end) {
            Integer i = tm.lowerKey(end);
            if (i != null && i >= start) {
                return false;
            }
            i = tm.lowerKey(start);
            if (i != null && tm.get(i) > start) {
                return false;
            }
            tm.put(start, end);
            return true;
        }
    }
}
```

written by 2499370956 original link here