costia.medium.com

Leetcode 1353. Maximum Number of Events That Can Be Attended (Medium)

Kostya

4-5 minutes

Given an array of `events` where `events[i] = [startDayi, endDayi]`. Every event `i` starts at `startDayi` and ends at `endDayi`.

You can attend an event `i` at any day `d` where `startTimei <= d <= endTimei`. Notice that you can only attend one event at any time `d`.

Return *the maximum number of events* you can attend.

Full problem statement here.

It took me several attempts.

Attempt #1. Greedy [Time Limit Exceeded]

Intuitive idea to sort events such as events that has sooner end day are located in front, and when they have the same end date, sort by start date in ascending order. Then iterate through days 1…100_001 and for each day find still not used day that fells into event's interval.

```
public int maxEvents(int[][] events) {
    Arrays.sort(events, (a,b) ->  {
        if(a[1] == b[1]) {
            return a[0] - b[0];
        }
        return a[1] - b[1];
    });
    for(int[] row: events) System.out.println(Arrays.toString(row));
            int[] day = new int[100001];

            for(int i = 0; i < events.length; ++i) {
        for(int j = events[i][0]; j <= events[i][1];j++) {
            if(day[j]==0){
                day[j]=1;
                break;
            }
        }
    }

            int sum=0;
    for(int k: day) sum+= k;

            return sum;
  }
```

This solution works fine on regular samples, but failed with [Time Limit Exceeded] on input like [[1,1], [1,2], [1,3]… [1,100001]]. Reason: *O(n²)* complexity in the worst case scenario, which we have with such input since on every iteration we forced to go from 1 to N.

Attempt #2. Tree set [Time Limit Exceeded]

Java TreeSet class has a powerfull feature that allows to find not only exact match for the element, but also subset of smaller/bigger elements. We can employ that idea and for every event interval [a;b] get subset of still available days, remove one of them and increment # of attended events:

```
public int maxEvents(int[][] events) {
        Arrays.sort(events, (a,b) ->  {
        if(a[1] == b[1]) {
            return a[0] - b[0];
        }
        return a[1] - b[1];
    });

            int attendedDays = 0;
    TreeSet<Integer> availableDays = new TreeSet<>();
    for(int i=1;i<=100_000;++i) availableDays.add(i);
    for(int[] event: events) {
        SortedSet<Integer> interval =
availableDays.subSet(event[0], event[1]+1);
        if(interval.size()>0 && interval.first() != null){
            int toRemove = interval.first();
            availableDays.remove(toRemove);
            attendedDays++;
        }
    }

            return attendedDays;
  }
```

Again, it works for small test cases, but fail with [Time Limit Exceeded] on different input now. Does it have even worse

O(time)?

Attempt #3. Greedy with Priority queue (Accepted!)

The idea is similar to (1): we iterate over 100_001 days and for all events that started on that day we add the last day to the priority queue; this queue contains elements from smallest to biggest, so the head points to the closest expiration day. Now, when we poll from the queue, we visit some event on day *i*. Also, before starting logic on day *i*, we first clean up queue and remove all elements that finished before *i* and we weren't able to attend them.

```
public int maxEvents(int[][] events) {
        Arrays.sort(events, (a,b) ->  {
        if(a[0] == b[0]) {
            return a[1] - b[1];
        }
        return a[0] - b[0];
    });

         int j = 0;
    int attended = 0;

    // by fefault, from small to big
    // PriorityQueue contains last day
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for(int i=1; i<100_001; i++) {
        //  remove from pq everything < currect day
        while(!pq.isEmpty() && pq.peek() < i)
            pq.poll();
                //  for all events started on day d, add the last day
to queue
```

```
        for(; j<events.length && events[j][0] == i; j++) {
            pq.offer(events[j][1]);
        }
                if(!pq.isEmpty()) {
            pq.poll();
            attended++;
        }
    }
            return attended;
}
```