LeetCode 436

https://leetcode.com/problems/find-right-interval/description/

Yifeng Zeng

Description

436. Find Right Interval

Given a set of intervals, for each of the interval i, check if there exists an interval j whose start point is bigger than or equal to the end point of the interval i, which can be called that j is on the "right" of i.

For any interval i, you need to store the minimum interval j's index, which means that the interval j has the minimum start point to build the "right" relationship for interval i. If the interval j doesn't exist, store -1 for the interval i. Finally, you need output the stored value of each interval as an array.

Note:

You may assume the interval's end point is always bigger than its start point.

You may assume none of these intervals have the same start point.

Example 3:

Input: [[1,4], [2,3], [3,4]]

Output: [-1, 2, -1]

Explanation:

There is no satisfied "right" interval for [1,4] and [3,4]. For [2,3], the interval [3,4] has minimum-"right" start point.

Idea Report

The primitive idea is to select two intervals and compare them, which takes O(n^2) time. We can think of nlogn which is sort. But is we sort by start, or sort by end, we cannot find a good way to solve the problem, so we can try to sort both start and end, and we need store where was this start, and end (using Point.i) and which interval it belongs to (using Point.intervalIndex). And we need to differentiate it is a start or end (using Point.isStart). We sort Point by Point.i, and we put a end Point before a start Point. So for an input {[1,4],[2,3],[3,4]}, we can get the Point array list: [1,1,0],[4,0,0],[2,1,1],[2,0,1],[3,1,2],[4,0,2], after sorting is: [1,1,0],[2,0,1],[2,1,1],[3,1,2],[4,0,0],[4,0,2]. We scan from the end to begining. And initialize our rightInterval as -1. For a Point which is an end Point, we just assign it to the rightInterval (res[cur.intervalIndex] = rightInterval), for a start Point, we update it with it's intervalIndex (rightInterval = cur.intervalIndex).

Code

```
class Solution {
    class Point {
        int i;
        int isStart;
        int intervalIndex;
        public Point(int i, int isStart, int intervalIndex) {
            this.i = i;
            this.isStart = isStart; // 1 is true, 0 is false
            this.intervalIndex = intervalIndex;
        }
    }

    public int[] findRightInterval(Interval[] intervals) {
        List<Point> list = new ArrayList<>();
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        for (int i = 0; i < intervals.length; i++) {
            list.add(new Point(intervals[i].start, 1, i));
            list.add(new Point(intervals[i].end, 0, i));
            min = Math.min(min, intervals[i].start);
            max = Math.max(max, intervals[i].end);
        }

        Collections.sort(list, (a, b) -> (a.i == b.i) ?
                         a.isStart - b.isStart : a.i - b.i);

        int[] res = new int[intervals.length];
        int rightInterval = -1;
        for (int i = list.size() - 1; i >= 0; i--) {
            Point cur = list.get(i);
            if (cur.isStart == 0) {
                res[cur.intervalIndex] = rightInterval;
            } else {
                rightInterval = cur.intervalIndex;
            }
        }
        return res;
    }
}
```

Summary

- Sweepline method.
- Similar to 252. Meeting Rooms, 253. Meeting Rooms II
- Because of time limitation, I still need to add other approach and revise the writing.

Follow up

Qinyuan introduced double pointer model, O(nlogn) sort, O(n) scan.

```
class Solution {
    // AC
    class Point {
        int i;
        int intervalIndex;
        public Point(int i, int index) {
            this.i = i;
            this.intervalIndex = index;
        }
    }

    public int[] findRightInterval(Interval[] intervals) {
        List<Point> start = new ArrayList<>();
        List<Point> end = new ArrayList<>();
        for (int i = 0; i < intervals.length; i++) {
            start.add(new Point(intervals[i].start, i));
            end.add(new Point(intervals[i].end, i));
        }

        Collections.sort(start, (a, b) -> a.i - b.i);
        Collections.sort(end, (a, b) -> a.i - b.i);
        int[] res = new int[intervals.length];
        Arrays.fill(res, -1);
        int j = 0;
        for (int i = 0; i < end.size(); i++) {
            Point pointEnd = end.get(i);
            while (j < start.size() && pointEnd.i > start.get(j).i) {
                j++;
            }
            if (j >= start.size()) {
                break;
            }
            res[pointEnd.intervalIndex] = start.get(j).intervalIndex;
        }
        return res;
    }
}
```

TreeMap.ceilingEntry()

```
class Solution {
    public int[] findRightInterval(Interval[] intervals) {
        int[] res = new int[intervals.length];
        TreeMap<Integer, Integer> relationships = new TreeMap<>();

        for (int i = 0; i < intervals.length; i++) {
            relationships.put(intervals[i].start, i);
        }

        for (int i = 0; i < intervals.length; i++) {
            Map.Entry<Integer, Integer> entry =
                relationships.ceilingEntry(intervals[i].end);
            res[i] = entry == null ? -1 : entry.getValue();
        }

        return res;
    }
}
```

Bucket sort

```
class Solution {
    public int[] findRightInterval(Interval[] intervals) {
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        for (int i = 0; i < intervals.length; i++) {
```

```java
            min = Math.min(min, intervals[i].start);
            max = Math.max(max, intervals[i].end);
        }

        int[] bucket = new int[max - min + 1];
        Arrays.fill(bucket, -1);
        for (int i = 0; i < intervals.length; i++) {
            bucket[intervals[i].start - min] = i;
        }

        for (int i = bucket.length - 2; i >= 0; i--) {
            if (bucket[i] == -1) {
                bucket[i] = bucket[i + 1];
            }
        }

        int[] res = new int[intervals.length];
        for (int i = 0; i < intervals.length; i++) {
            res[i] = bucket[intervals[i].end - min];
        }

        return res;
    }
}
```