759. Employee Free Time

Leetcode Link

Problem Explanation:

Given a list of employee work schedules with each employee having a list of non-overlapping intervals representing their working hours, we are tasked with finding the common free time for all employees, or in other words, the times when all employees are not working.

The input is a nested list of intervals, each interval as `[start, end]`, with `start < end`. The intervals are non-overlapping and are already sorted in ascending order. The output should also be a list of sorted intervals.

For example, consider `schedule = [[[1,3],[6,7]],[[2,4]],[[2,5],[9,12]]]`. Here, Employee 1 works from 1 to 3 and 6 to 7. Employee 2 works from 2 to 4 and Employee 3 works from 2 to 5 and 9 to 12. The common free time for all employees is `[5,6]` and `[7,9]` as these are the intervals when all employees are free.

Solution Approach:

The proposed solution concatenates all work schedules into a single list, sorts this list and then identifies the gaps between work intervals, which represent the common free time for all employees.

First, the solution creates an empty array to store the result. Then, it goes through the work shifts of each employee and combines all the intervals to a single list.

Next, this combined list of work intervals is sorted using the start of the work intervals. This will ensure that the work schedules are in chronological order.

After that, the solution stores the end time of the first shift in a variable called `prevEnd`, and then iterates through each work interval.

If the start of a work interval is greater than `prevEnd`, this means there's a gap between the end of the previous work shift and the start of the current work shift. This gap represents a common free time, which is then added to the result list.

Finally, `prevEnd` is updated with the maximum end time of the current work interval or the `prevEnd`.

Through this approach, the algorithm successfully identifies the common free time of all employees.

Python Solution

```

python
# Definition for an Interval.
# class Interval:
#     def __init__(self, start: int = None, end: int = None):
#         self.start = start
#         self.end = end

class Solution:
    def employeeFreeTime(self, schedule: '[[Interval]]') -> '[Interval]':
        # Flattening the schedule
        intervals = [interval for employee in schedule for interval in employee]
        # Sorting by start of each Interval
        intervals.sort(key=lambda x: x.start)
        res, end = [], intervals[0].end
        # Checking for free time between intervals
        for i in intervals[1:]:
            if end < i.start:
                res.append(Interval(end, i.start))
            end = max(end, i.end)
        return res
```

JavaScript Solution:

```

javascript
function Interval(start, end) {
    this.start = start;
    this.end = end;
}

function employeeFreeTime(schedule) {
    // Flattening the schedule
    let intervals = [].concat(...schedule);
    // Sorting by start of each Interval
    intervals.sort((a, b) => a.start - b.start);
    let res = [], end = intervals[0].end;
    // Checking for free time between intervals
    for (let i = 1; i < intervals.length; i++) {
        if (end < intervals[i].start) {
            res.push(new Interval(end, intervals[i].start));
        }
        end = Math.max(end, intervals[i].end);
    }
    return res;
}
```

Java Solution:

```

java
/**
 * Definition for an interval.
 * public class Interval {
 *     int start;
 *     int end;
 *     Interval() { start = 0; end = 0; }
 *     Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
    public List<Interval> employeeFreeTime(List<List<Interval>> schedule) {
        List<Interval> res = new ArrayList<>();
        List<Interval> intervals = new ArrayList<>();
        // Flattening the schedule
        for (List<Interval> employee : schedule)
            for (Interval interval : employee)
                intervals.add(interval);
        // Sorting by start of each Interval
        Collections.sort(intervals, (a, b) -> a.start - b.start);
        int end = intervals.get(0).end;
        // Checking for free time between intervals
        for (Interval interval : intervals) {
            if (interval.start > end)
                res.add(new Interval(end, interval.start));
            end = Math.max(end, interval.end);
        }
        return res;
    }
}
```

These solutions work by first flattening the schedule list, then sorting the intervals by their start times. They then keep track of the end time of the current interval. If the start of the next interval is greater than this end time, it means there is a gap where all employees are free, and this gap is then added to the results. These solutions have a time complexity of O(N log N) due to the sorting operation, where N is the total number of intervals. The space complexity is also O(N) for storing the intervals and the result.