759. Employee Free Time

Jan. 6, 2018 | 23.6K views

Average Rating: 3.92 (37 votes)

We are given a list `schedule` of employees, which represents the working time for each employee.

Each employee has a list of non-overlapping `Intervals`, and these intervals are in sorted order.

Return the list of finite intervals representing **common, positive-length free time** for *all* employees, also in sorted order.

(Even though we are representing `Intervals` in the form `[x, y]`, the objects inside are `Intervals`, not lists or arrays. For example, `schedule[0][0].start = 1`, `schedule[0][0].end = 2`, and `schedule[0][0][0]` is not defined). Also, we wouldn't include intervals like [5, 5] in our answer, as they have zero length.

Example 1:

```
Input: schedule = [[[1,2],[5,6]],[[1,3]],[[4,10]]]
Output: [[3,4]]
Explanation: There are a total of three employees, and all common
free time intervals would be [-inf, 1], [3, 4], [10, inf].
We discard any intervals that contain inf as they aren't finite.
```

Example 2:

```
Input: schedule = [[[1,3],[6,7]],[[2,4]],[[2,5],[9,12]]]
Output: [[5,6],[7,9]]
```

Constraints:

- `1 <= schedule.length , schedule[i].length <= 50`
- `0 <= schedule[i].start < schedule[i].end <= 10^8`

Approach #1: Events (Line Sweep) [Accepted]

Intuition

If some interval overlaps *any* interval (for any employee), then it won't be included in the answer. So we could reduce our problem to the following: given a set of intervals, find all places where there are no intervals.

To do this, we can use an "events" approach present in other interval problems. For each interval `[s, e]`, we can think of this as two events: `balance++` when `time = s`, and `balance--` when `time = e`. We want to know the regions where `balance == 0`.

Algorithm

For each interval, create two events as described above, and sort the events. Now for each event occuring at time `t`, if the `balance` is `0`, then the preceding segment `[prev, t]` did not have any intervals present, where `prev` is the previous value of `t`.

Java | Python — Copy

```python
class Solution(object):
    def employeeFreeTime(self, avails):
        OPEN, CLOSE = 0, 1

        events = []
        for emp in avails:
            for iv in emp:
                events.append((iv.start, OPEN))
                events.append((iv.end, CLOSE))

        events.sort()
        ans = []
        prev = None
        bal = 0
        for t, cmd in events:
            if bal == 0 and prev is not None:
                ans.append(Interval(prev, t))

            bal += 1 if cmd is OPEN else -1
            prev = t

        return ans
```

Complexity Analysis

- Time Complexity: $O(C \log C)$, where C is the number of intervals across all employees.
- Space Complexity: $O(C)$.

Approach #2: Priority Queue [Accepted]

Intuition

Say we are at some time where no employee is working. That work-free period will last until the next time some employee has to work.

So let's maintain a heap of the next time an employee has to work, and it's associated job. When we process the next time from the heap, we can add the next job for that employee.

Algorithm

Keep track of the latest time `anchor` that we don't know of a job overlapping that time.

When we process the earliest occurring job not yet processed, it occurs at time `t`, by employee `e_id`, and it was that employee's `e_jx`'th job. If `anchor < t`, then there was a free interval `Interval(anchor, t)`.

Java | Python — Copy

```python
class Solution(object):
    def employeeFreeTime(self, avails):
        ans = []
        pq = [(emp[0].start, ei, 0) for ei, emp in enumerate(avails)]
        heapq.heapify(pq)
        anchor = min(iv.start for emp in avails for iv in emp)
        while pq:
            t, e_id, e_jx = heapq.heappop(pq)
            if anchor < t:
                ans.append(Interval(anchor, t))
            anchor = max(anchor, avails[e_id][e_jx].end)
            if e_jx + 1 < len(avails[e_id]):
                heapq.heappush(pq, (avails[e_id][e_jx+1].start, e_id, e_jx+1))

        return ans
```

Complexity Analysis

- Time Complexity: $O(C \log N)$, where N is the number of employees, and C is the number of jobs across all employees. The maximum size of the heap is N, so each push and pop operation is $O(\log N)$, and there are $O(C)$ such operations.
- Space Complexity: $O(N)$ in additional space complexity.

Analysis written by: @awice.

Rate this article:

Previous Next

Comments: 19

Sort By

Type comment here... (Markdown is supported)

Preview Post

hari_prasath ★26 February 26, 2019 5:18 AM

INFERENCE:
The Solution logic looks similar to Merge K sorted Linked List with few teaks after removing an element from the heap and adding it to the result

15 Share Reply

Suralin ★149 September 15, 2018 11:14 PM

Simpler approach from merging intervals:

```
def employeeFreeTime(self, schedule):
    ints = sorted((i for s in schedule for i in s), key=lambda x: x.start)
    res, end = [], ints[0].end
```

Read More

14 Share Reply

SHOW 2 REPLIES

VaibhalavskyOSHMKUFA ★218 October 11, 2018 12:51 PM

Actually we can use the solution of Merge Intervals as a building block for the solution for this problem. After merged all intervals, just go through again to find the gaps.

6 Share Reply

SHOW 5 REPLIES

bwv988 ★273 January 28, 2018 1:53 AM

Line sweeping is awesome! Good to know we can use it here.

4 Share Reply

arnold_y ★4 November 13, 2018 4:10 AM

You can't guarantee that the List is not LinkedList. In Approach #2, the get method could cost you O(n).

3 Share Reply

etids ★8 January 25, 2020 1:36 AM

My heap approach seems simpler.

```
def employeeFreeTime(self, schedule: '[[Interval]]') -> '[Interval]':
    ans = []
    pq = [(i.start, i.end) for ee in schedule for i in ee]
```

Read More

2 Share Reply

xuyichen2010 ★3 November 28, 2019 7:57 PM

The variable naming is a bit confusing...

2 Share Reply

izyluke ★14 September 21, 2019 11:50 PM

Why can't we just construct the complement of interval and get their intersection? That would just cost O(n).

1 Share Reply

jjkim ★2 December 25, 2018 10:38 AM

Line 16 of Approach 1 sol in Python should actually be `if bal == 0 and prev is not None and prev != t:`

1 Share Reply

daboss ★5 January 1, 2020 7:56 PM

It would be wonderful if you could add some comments at the code since its used as an explanation and not just submit the code like that.

0 Share Reply