Solution Review: Problem Challenge 1

We'll cover the following

- Minimum Meeting Rooms (hard)
- Solution
- Code
  - Time complexity
  - Space complexity
- Similar Problems

Minimum Meeting Rooms (hard)

Given a list of intervals representing the start and end time of 'N' meetings, find the **minimum number of rooms** required to **hold all the meetings**.

Example 1:

```
Meetings: [[1,4], [2,5], [7,9]]
Output: 2
Explanation: Since [1,4] and [2,5] overlap, we need two rooms to hold these two meetings. [7,
9] can
occur in any of the two rooms later.
```

Example 2:

```
Meetings: [[6,7], [2,4], [8,12]]
Output: 1
Explanation: None of the meetings overlap, therefore we only need one room to hold all meetings.
```

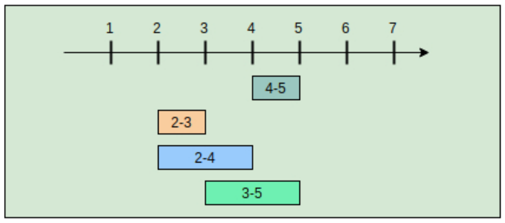
Example 3:

```
Meetings: [[1,4], [2,3], [3,6]]
Output:2
Explanation: Since [1,4] overlaps with the other two meetings [2,3] and [3,6], we need two room
s to
hold all the meetings.
```

Example 4:

```
Meetings: [[4,5], [2,3], [2,4], [3,5]]
Output: 2
Explanation: We will need one room for [2,3] and [3,5], and another room for [2,4] and [4,5].

Here is a visual representation of Example 4:
```



Solution

Solution

Let’s take the above-mentioned example (4) and try to follow our Merge Intervals approach:

Meetings: [[4,5], [2,3], [2,4], [3,5]]

Step 1: Sorting these meetings on their start time will give us: [[2,3], [2,4], [3,5], [4,5]]

Step 2: Merging overlapping meetings:

- [2,3] overlaps with [2,4], so after merging we’ll have => [[2,4], [3,5], [4,5]]
- [2,4] overlaps with [3,5], so after merging we’ll have => [[2,5], [4,5]]
- [2,5] overlaps [4,5], so after merging we’ll have => [2,5]

Since all the given meetings have merged into one big meeting ([2,5]), does this mean that they all are overlapping and we need a minimum of four rooms to hold these meetings? You might have already guessed that the answer is NO! As we can clearly see, some meetings are mutually exclusive. For example, [2,3] and [3,5] do not overlap and can happen in one room. So, to correctly solve our problem, we need to keep track of the mutual exclusiveness of the overlapping meetings.

Here is what our strategy will look like:

1. We will sort the meetings based on start time.
2. We will schedule the first meeting (let’s call it `m1`) in one room (let’s call it `r1`).
3. If the next meeting `m2` is not overlapping with `m1`, we can safely schedule it in the same room `r1`.
4. If the next meeting `m3` is overlapping with `m2` we can’t use `r1`, so we will schedule it in another room (let’s call it `r2`).
5. Now if the next meeting `m4` is overlapping with `m3`, we need to see if the room `r1` has become free. For this, we need to keep track of the end time of the meeting happening in it. If the end time of `m2` is before the start time of `m4`, we can use that room `r1`, otherwise, we need to schedule `m4` in another room `r3`.

We can conclude that we need to **keep track of the ending time of all the meetings currently happening** so that when we try to schedule a new meeting, we can see what meetings have already ended. We need to put this information in a data structure that can easily give us the smallest ending time. A **Min Heap** would fit our requirements best.

So our algorithm will look like this:

1. Sort all the meetings on their start time.
2. Create a min-heap to store all the active meetings. This min-heap will also be used to find the active meeting with the smallest end time.
3. Iterate through all the meetings one by one to add them in the min-heap. Let’s say we are trying to schedule the meeting `m1`.
4. Since the min-heap contains all the active meetings, so before scheduling `m1` we can remove all meetings from the heap that have ended before `m1`, i.e., remove all meetings from the heap that have an end time smaller than or equal to the start time of `m1`.
5. Now add `m1` to the heap.
6. The heap will always have all the overlapping meetings, so we will need rooms for all of them. Keep a counter to remember the maximum size of the heap at any time which will be the minimum number of rooms needed.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```python
from heapq import *


class Meeting:
  def __init__(self, start, end):
    self.start = start
    self.end = end

  def __lt__(self, other):
    # min heap based on meeting.end
    return self.end < other.end


def min_meeting_rooms(meetings):
  # sort the meetings by start time
  meetings.sort(key=lambda x: x.start)

  minRooms = 0
  minHeap = []
  for meeting in meetings:
    # remove all the meetings that have ended
    while(len(minHeap) > 0 and meeting.start >= minHeap[0].end):
      heappop(minHeap)
    # add the current meeting into min_heap
    heappush(minHeap, meeting)
    # all active meetings are in the min_heap, so we need rooms for all of them.
    minRooms = max(minRooms, len(minHeap))
  return minRooms


def main():
  print("Minimum meeting rooms required: " + str(min_meeting_rooms(
    [Meeting(4, 5), Meeting(2, 3), Meeting(2, 4), Meeting(3, 5)])))
  print("Minimum meeting rooms required: " +
        str(min_meeting_rooms([Meeting(1, 4), Meeting(2, 5), Meeting(7, 9)])))
  print("Minimum meeting rooms required: " +
        str(min_meeting_rooms([Meeting(6, 7), Meeting(2, 4), Meeting(8, 12)])))
  print("Minimum meeting rooms required: " +
        str(min_meeting_rooms([Meeting(1, 4), Meeting(2, 3), Meeting(3, 6)])))
  print("Minimum meeting rooms required: " + str(min_meeting_rooms(
    [Meeting(4, 5), Meeting(2, 3), Meeting(2, 4), Meeting(3, 5)])))


main()
```

Run Save Reset

Time complexity

The time complexity of the above algorithm is $O(N * logN)$, where 'N' is the total number of meetings. This is due to the sorting that we did in the beginning. Also, while iterating the meetings we might need to poll/offer meeting to the priority queue. Each of these operations can take $O(logN)$. Overall our algorithm will take $O(NlogN)$.

Space complexity

The space complexity of the above algorithm will be $O(N)$ which is required for sorting. Also, in the worst case scenario, we'll have to insert all the meetings into the **Min Heap** (when all meetings overlap) which will also take $O(N)$ space. The overall space complexity of our algorithm is $O(N)$.

Similar Problems

Problem 1: Given a list of intervals, find the point where the maximum number of intervals overlap.

Problem 2: Given a list of intervals representing the arrival and departure times of trains to a train station, our goal is to find the minimum number of platforms required for the train station so that no train has to wait.

Both of these problems can be solved using the approach discussed above.

← Back
Problem Challenge 1

Next →
Problem Challenge 2

✓ Completed