Intervals Intersection (medium)

We'll cover the following

Problem Statement

Given two lists of intervals, find the **intersection of these two lists**. Each list consists of **disjoint intervals sorted on their start time**.

Example 1:

```
Input: arr1=[[1, 3], [5, 6], [7, 9]], arr2=[[2, 3], [5, 7]]
Output: [2, 3], [5, 6], [7, 7]
Explanation: The output list contains the common intervals between the two lists.
```

Example 2:

```
Input: arr1=[[1, 3], [5, 7], [9, 12]], arr2=[[5, 10]]
Output: [5, 7], [9, 10]
Explanation: The output list contains the common intervals between the two lists.
```

Try it yourself

Try solving this question here:

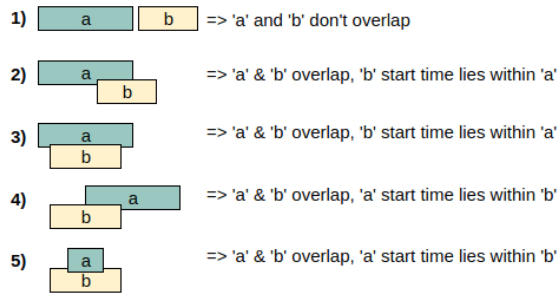
Java | Python3 | JS | C++

```python
def merge(intervals_a, intervals_b):
  result = []
  # TODO: Write your code here
  return result


def main():
  print("Intervals Intersection: " + str(merge([[1, 3], [5, 6], [7, 9]], [[2, 3], [5, 7]])))
  print("Intervals Intersection: " + str(merge([[1, 3], [5, 7], [9, 12]], [[5, 10]])))


main()

```

Run | Save | Reset

Solution

This problem follows the Merge Intervals pattern. As we have discussed under Insert Interval, there are five overlapping possibilities between two intervals 'a' and 'b'. A close observation will tell us that whenever the two intervals overlap, one of the interval's start time lies within the other interval. This rule can help us identify if any two intervals overlap or not.



Now, if we have found that the two intervals overlap, how can we find the overlapped part?

Again from the above diagram, the overlapping interval will be equal to:

```
start = max(a.start, b.start)
end = min(a.end, b.end)
```

That is, the highest start time and the lowest end time will be the overlapping interval.

So our algorithm will be to iterate through both the lists together to see if any two intervals overlap. If two intervals overlap, we will insert the overlapped part into a result list and move on to the next interval which is finishing early.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```python
def merge(intervals_a, intervals_b):
  result = []
  i, j, start, end = 0, 0, 0, 1

  while i < len(intervals_a) and j < len(intervals_b):
    # check if intervals overlap and intervals_a[i]'s start time lies within the other intervals_b[j]
    a_overlaps_b = intervals_a[i][start] >= intervals_b[j][start] and \
                   intervals_a[i][start] <= intervals_b[j][end]

    # check if intervals overlap and intervals_a[j]'s start time lies within the other intervals_b[i]
    b_overlaps_a = intervals_b[j][start] >= intervals_a[i][start] and \
                   intervals_b[j][start] <= intervals_a[i][end]

    # store the the intersection part
    if (a_overlaps_b or b_overlaps_a):
      result.append([max(intervals_a[i][start], intervals_b[j][start]), min(
        intervals_a[i][end], intervals_b[j][end])])

    # move next from the interval which is finishing first
    if intervals_a[i][end] < intervals_b[j][end]:
      i += 1
    else:
      j += 1

  return result


def main():
  print("Intervals Intersection: " + str(merge([[1, 3], [5, 6], [7, 9]], [[2, 3], [5, 7]])))
  print("Intervals Intersection: " + str(merge([[1, 3], [5, 7], [9, 12]], [[5, 10]])))


main()
```

Run | Save | Reset

Time complexity

As we are iterating through both the lists once, the time complexity of the above algorithm is $O(N + M)$, where 'N' and 'M' are the total number of intervals in the input arrays respectively.

Space complexity

Space complexity

Ignoring the space needed for the result list, the algorithm runs in constant space $O(1)$.

Back
Insert Interval (medium)

Next
Conflicting Appointments (medium)

Completed

Report an Issue
Ask a Question