Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

25% completed

Search Course

Intervals

- Introduction
- Merge Intervals (medium)
- Insert Interval (medium)
- Intervals Intersection (medium)
- Conflicting Appointments (medium)
- Problem Challenge 1

Intervals Intersection (medium)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
- Time complexity
- Space complexity

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:

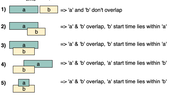
Java | Python3 | JS | C++

```
class IntervalsIntersection {

  public static Interval[] merge(Interval[] arr1, Interval[] arr2) {
    List<Interval> intervalsIntersection = new ArrayList<Interval>();
    // TODO: Write your code here
    return intervalsIntersection.toArray(new Interval[intervalsIntersection.size()]);
  }

  public static void main(String[] args) {
    Interval[] input1 = new Interval[] { new Interval(1, 3), new Interval(5, 6), new Interve
    Interval[] input2 = new Interval[] { new Interval(2, 3), new Interval(5, 7) };
    Interval[] result = IntervalsIntersection.merge(input1, input2);
    System.out.print("Intervals Intersection: ");
    for (Interval interval : result)
      System.out.print("[" + interval.start + "," + interval.end + "] ");
    System.out.println();

    input1 = new Interval[] { new Interval(1, 3), new Interval(5, 7), new Interval(9, 12) };
    input2 = new Interval[] { new Interval(5, 10) };
    result = IntervalsIntersection.merge(input1, input2);
    System.out.print("Intervals Intersection: ");
    for (Interval interval : result)
      System.out.print("[" + interval.start + "," + interval.end + "] ");
  }
}
```

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

```
import java.util.*;

class Interval {
  int start;
  int end;

  public Interval(int start, int end) {
    this.start = start;
    this.end = end;
  }
};

class IntervalsIntersection {

  public static Interval[] merge(Interval[] arr1, Interval[] arr2) {
    List<Interval> result = new ArrayList<Interval>();
    int i = 0, j = 0;
    while (i < arr1.length && j < arr2.length) {
      // check if the interval arr1[i] intersects with arr2[j]
      // check if one of the interval's start time lies within the other interval
      if ((arr1[i].start >= arr2[j].start && arr1[i].start <= arr2[j].end)
          || (arr2[j].start >= arr1[i].start && arr2[j].start <= arr1[i].end)) {
        // store the intersection part
        result.add(new Interval(Math.max(arr1[i].start, arr2[j].start), Math.min(arr1[i].end

      // move next from the interval which is finishing first
      if (arr1[i].end < arr2[j].end)
```

Run | Save | Reset

for Coding Questions

25% completed

Search Course

Pattern: Merge Intervals

- Introduction
- Merge Intervals (medium)
- Insert Interval (medium)
- Intervals Intersection (medium)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
- Time complexity
- Space complexity

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++

```
class IntervalsIntersection {

  public static Interval[] merge(Interval[] arr1, Interval[] arr2) {
    List<Interval> intervalsIntersection = new ArrayList<Interval>();
    // TODO: Write your code here
    return intervalsIntersection.toArray(new Interval[intervalsIntersection.size()]);
  }

  public static void main(String[] args) {
    Interval[] input1 = new Interval[] { new Interval(1, 3), new Interval(5, 6), new Interve
    Interval[] input2 = new Interval[] { new Interval(2, 3), new Interval(5, 7) };
    Interval[] result = IntervalsIntersection.merge(input1, input2);
    System.out.print("Intervals Intersection: ");
    for (Interval interval : result)
      System.out.print("[" + interval.start + "," + interval.end + "] ");
    System.out.println();

    input1 = new Interval[] { new Interval(1, 3), new Interval(5, 7), new Interval(9, 12) };
    input2 = new Interval[] { new Interval(5, 10) };
    result = IntervalsIntersection.merge(input1, input2);
    System.out.print("Intervals Intersection: ");
    for (Interval interval : result)
      System.out.print("[" + interval.start + "," + interval.end + "] ");
  }
}
```

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

```
import java.util.*;

class Interval {
  int start;
  int end;

  public Interval(int start, int end) {
    this.start = start;
    this.end = end;
  }
};

class IntervalsIntersection {

  public static Interval[] merge(Interval[] arr1, Interval[] arr2) {
    List<Interval> result = new ArrayList<Interval>();
    int i = 0, j = 0;
    while (i < arr1.length && j < arr2.length) {
      // check if the interval arr1[i] intersects with arr2[j]
      // check if one of the interval's start time lies within the other interval
      if ((arr1[i].start >= arr2[j].start && arr1[i].start <= arr2[j].end)
          || (arr2[j].start >= arr1[i].start && arr2[j].start <= arr1[i].end)) {
        // store the intersection part
        result.add(new Interval(Math.max(arr1[i].start, arr2[j].start), Math.min(arr1[i].end

      // move next from the interval which is finishing first
      if (arr1[i].end < arr2[j].end)
```

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

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

Want to work at Google, Facebook, or Amazon? Get hired faster with anonymous mock interviews conducted by senior engineers from those companies. Detailed feedback helps you prep. See how

Back

Insert Interval (medium)

Next

Conflicting Appointments (medium)

Mark as Completed

Report an Issue