Grokking the Coding Interview: Patterns for Coding Questions

21% completed

Search Course

Intervals

- Introduction
- Merge Intervals (medium)
- Insert Interval (medium)
- Intervals Intersection (medium)
- Conflicting Appointments (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Cyclic Sort

- Introduction
- Cyclic Sort (easy)
- Find the Missing Number (easy)
- Find all Missing Numbers (easy)
- Find the Duplicate Number (easy)
- Find all Duplicate Numbers (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: In-place Reversal of a LinkedList

- Introduction
- Reverse a LinkedList (easy)
- Reverse a Sub-list (medium)
- Reverse every K-element Sub-list (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Breadth First Search

- Introduction
- Binary Tree Level Order Traversal (easy)
- Reverse Level Order Traversal (easy)
- Zigzag Traversal (medium)
- Level Averages in a Binary Tree (easy)
- Minimum Depth of a Binary Tree (easy)

Conflicting Appointments (medium)

We'll cover the following

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

Problem Statement

Given an array of intervals representing 'N' appointments, find out if a person can **attend all the appointments**.

Example 1:

```
Appointments: [[1,4], [2,5], [7,9]]
Output: false
Explanation: Since [1,4] and [2,5] overlap, a person cannot attend both of these appointments.
```

Example 2:

```
Appointments: [[6,7], [2,4], [8,12]]
Output: true
Explanation: None of the appointments overlap, therefore a person can attend all of them.
```

Example 3:

```
Appointments: [[4,5], [2,3], [3,6]]
Output: false
Explanation: Since [4,5] and [3,6] overlap, a person cannot attend both of these appointments.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
class Interval {
  constructor(start, end) {
    this.start = start;
    this.end = end;
  }

  print_interval() {
    process.stdout.write(`[${this.start}, ${this.end}]`);
  }
}

const can_attend_all_appointments = function(intervals) {
  // TODO: Write your code here
  return false;
};


console.log(`Can attend all appointments: ${can_attend_all_appointments([
  new Interval(1, 4),
  new Interval(2, 5),
  new Interval(7, 9),
])}`);

console.log(`Can attend all appointments: ${can_attend_all_appointments([
  new Interval(6, 7),
  new Interval(2, 4),
  new Interval(8, 12),
])}`);
```

RUN SAVE RESET

Solution

The problem follows the Merge Intervals pattern. We can sort all the intervals by start time and then check if any two intervals overlap. A person will not be able to attend all appointments if any two appointments overlap.

Code

Here is what our algorithm will look like:

Level Order Successor (easy)
Connect Level Order Siblings (medium)
Problem Challenge 1
Solution Review: Problem Challenge 1
Problem Challenge 2
Solution Review: Problem Challenge 2

Pattern: Tree Depth First Search
Introduction
Binary Tree Path Sum (easy)
All Paths for a Sum (medium)
Sum of Path Numbers (medium)
Path With Given Sequence (medium)
Count Paths for a Sum (medium)
Problem Challenge 1
Solution Review: Problem Challenge 1
Problem Challenge 2
Solution Review: Problem Challenge 2

Pattern: Two Heaps
Introduction
Find the Median of a Number Stream (medium)
Sliding Window Median (hard)
Maximize Capital (hard)
Problem Challenge 1
Solution Review: Problem Challenge 1

Pattern: Subsets
Introduction
Subsets (easy)
Subsets With Duplicates (easy)
Permutations (medium)
String Permutations by changing case (medium)
Balanced Parentheses (hard)
Unique Generalized Abbreviations (hard)
Problem Challenge 1
Solution Review: Problem Challenge 1
Problem Challenge 2
Solution Review: Problem Challenge 2
Problem Challenge 3
Solution Review: Problem Challenge 3

Pattern: Modified Binary Search
Introduction
Order-agnostic Binary Search (easy)

Java | Python3 | C++ | JS

```
class Interval {
  constructor(start, end) {
    this.start = start;
    this.end = end;
  }

  print_interval() {
    process.stdout.write(`[${this.start}, ${this.end}]`);
  }
}

function can_attend_all_appointments(intervals) {
  intervals.sort((a, b) => a.start - b.start);
  for (i = 1; i < intervals.length; i++) {
    if (intervals[i].start < intervals[i - 1].end) {
      // please note the comparison above, it is "<" and not "<="
      // while merging we needed "<=" comparison, as we will be merging the two
      // intervals having condition "intervals[i][start] === intervals[i - 1][end]" but
      // such intervals don't represent conflicting appointments as one starts right
      // after the other
      return false;
    }
  }
  return true;
}


console.log(`Can attend all appointments: ${can_attend_all_appointments([
```

RUN SAVE RESET

Time complexity

The time complexity of the above algorithm is $O(N * logN)$, where 'N' is the total number of appointments. Though we are iterating the intervals only once, our algorithm will take $O(N * logN)$ since we need to sort them in the beginning.

Space complexity

The space complexity of the above algorithm will be $O(N)$, which we need for sorting. For Java, `Arrays.sort()` uses Timsort, which needs $O(N)$ space.

Similar Problems

Problem 1: Given a list of appointments, find all the conflicting appointments.

Example:

```
Appointments: [[4,5], [2,3], [3,6], [5,7], [7,8]]
Output:
[4,5] and [3,6] conflict.
[3,6] and [5,7] conflict.
```


MARK AS COMPLETED
← Back
Intervals Intersection (medium)
Next →
Problem Challenge 1
Report an Issue