Grokking the Coding Interview: Patterns for Coding Questions

85% completed

Search Course

Introduction

Pattern: Sliding Window

Pattern: Two Pointers

Pattern: Fast & Slow pointers

Pattern: Merge Intervals

Pattern: Cyclic Sort

Pattern: In-place Reversal of a LinkedList

Pattern: Tree Breadth First Search

Pattern: Tree Depth First Search

Pattern: Two Heaps

Pattern: Subsets

Pattern: Modified Binary Search

Pattern: Bitwise XOR

Pattern: Top 'K' Elements

Pattern: K-way merge

- Introduction
- Merge K Sorted Lists (medium)
- Kth Smallest Number in M Sorted Lists (Medium)
- Kth Smallest Number in a Sorted Matrix (Hard)
- Smallest Number Range (Hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern : 0/1 Knapsack (Dynamic Programming)

- Introduction

Smallest Number Range (Hard)

We'll cover the following

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

Problem Statement

Given 'M' sorted arrays, find the smallest range that includes at least one number from each of the 'M' lists.

Example 1:

```
Input: L1=[1, 5, 8], L2=[4, 12], L3=[7, 8, 10]
Output: [4, 7]
Explanation: The range [4, 7] includes 5 from L1, 4 from L2 and 7 from L3.
```

Example 2:

```
Input: L1=[1, 9], L2=[4, 12], L3=[7, 10, 16]
Output: [9, 12]
Explanation: The range [9, 12] includes 9 from L1, 12 from L2 and 10 from L3.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
const find_smallest_range = function(lists) {
  // TODO: Write your code here
  return [-1, -1];
};


console.log(`Smallest range is: ${find_smallest_range([[1, 5, 8], [4, 12], [7, 8, 10]])}`)

```

RUN SAVE RESET

Solution

This problem follows the **K-way merge** pattern and we can follow a similar approach as discussed in Merge K Sorted Lists.

We can start by inserting the first number from all the arrays in a min-heap. We will keep track of the largest number that we have inserted in the heap (let's call it `currentMaxNumber`).

In a loop, we'll take the smallest (top) element from the min-heap and `currentMaxNumber` has the largest element that we inserted in the heap. If these two numbers give us a smaller range, we'll update our range. Finally, if the array of the top element has more elements, we'll insert the next element to the heap.

We can finish searching the minimum range as soon as an array is completed or, in other terms, the heap has less than 'M' elements.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
const Heap = require('./collections/heap'); //http://www.collectionsjs.com

function find_smallest_range(lists) {
  const minHeap = new Heap([], null, ((a, b) => b[0] - a[0]));
  let rangeStart = 0,
    rangeEnd = Infinity,
    currentMaxNumber = -Infinity;

  // put the 1st element of each array in the max heap
  lists.forEach((list) => {
    minHeap.push([list[0], 0, list]);
    currentMaxNumber = Math.max(currentMaxNumber, list[0]);
  });


  // take the smallest(top) element from the min heap, if it gives us smaller range, update the ranges
  // if the array of the top element has more elements, insert the next element in the heap
  while (minHeap.length === lists.length) {
```

0/1 Knapsack (medium)
Equal Subset Sum Partition (medium)
Subset Sum (medium)
Minimum Subset Sum Difference (hard)
Problem Challenge 1
Solution Review: Problem Challenge 1
Problem Challenge 2
Solution Review: Problem Challenge 2

Pattern: Topological Sort (Graph)

Introduction
Topological Sort (medium)
Tasks Scheduling (medium)
Tasks Scheduling Order (medium)
All Tasks Scheduling Orders (hard)
Alien Dictionary (hard)
Problem Challenge 1
Solution Review: Problem Challenge 1
Problem Challenge 2
Solution Review: Problem Challenge 2

Miscellaneous

Kth Smallest Number (hard)

Conclusions

Where to Go from Here


```
  const [num, i, list] = minHeap.pop();
  if (rangeEnd - rangeStart > currentMaxNumber - num) {
    rangeStart = num;
    rangeEnd = currentMaxNumber;
  }
  if (list.length > i + 1) {
    // insert the next element in the heap
    minHeap.push([list[i + 1], i + 1, list]);
    currentMaxNumber = Math.max(currentMaxNumber, list[i + 1]);
  }
```

RUN SAVE RESET

Close

Output 8.083s

```
Smallest range is: 4,7
```

Time complexity

Since, at most, we'll be going through all the elements of all the arrays and will remove/add one element in the heap in each step, the time complexity of the above algorithm will be $O(N * logM)$ where 'N' is the total number of elements in all the 'M' input arrays.

Space complexity

The space complexity will be $O(M)$ because, at any time, our min-heap will be store one number from all the 'M' input arrays.

Back
Kth Smallest Number in a Sorted Matri...

Next
Problem Challenge 1

Mark as Completed