

Grokking the Coding Interview: Patterns for Coding Questions

80% completed



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



- Introduction
- Top 'K' Numbers (easy)
- Kth Smallest Number (easy)
- 'K' Closest Points to the Origin (easy)
- Connect Ropes (easy)
- Top 'K' Frequent Numbers (medium)
- Frequency Sort (medium)
- Kth Largest Number in a Stream (medium)
- 'K' Closest Numbers (medium)
- Maximum Distinct Elements (medium)
- Sum of Elements (medium)
- Rearrange String (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: K-wav merae

Rearrange String (hard)

We'll cover the following

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

Problem Statement

Given a string, find if its letters can be rearranged in such a way that no two same characters come next to each other.

Example 1:

```
Input: "aappp"
Output: "papap"
Explanation: In "papap", none of the repeating characters come next to each other.
```

Example 2:

```
Input: "Programming"
Output: "rgmrpmPiano" or "gmringmrPoa" or "gmrPagimnor", etc.
Explanation: None of the repeating characters come next to each other.
```

Example 3:

```
Input: "aapa"
Output: ""
Explanation: In all arrangements of "aapa", atleast two 'a' will come together e.g., "apaa", "paaa".
```

Try it yourself

Try solving this question here:

Java

Python3

JS

C++

```
1 const rearrange_string = function(str) {
2   // TODO: Write your code here
3   return "";
4 };
5
6
7 console.log('Rearranged string: ${rearrange_string("aappp")}')
8 console.log('Rearranged string: ${rearrange_string("Programming")}')
9 console.log('Rearranged string: ${rearrange_string("aapa")}')
10
```

RUN

SAVE

RESET

Solution

This problem follows the [Top 'K' Numbers](#) pattern. We can follow a greedy approach to find an arrangement of the given string where no two same characters come next to each other.

We can work in a stepwise fashion to create a string with all characters from the input string. Following a greedy approach, we should first append the most frequent characters to the output strings, for which we can use a **Max Heap**. By appending the most frequent character first, we have the best chance to find a string where no two same characters come next to each other.

So in each step, we should append one occurrence of the highest frequency character to the output string. We will not put this character back in the heap to ensure that no two same characters are adjacent to each other. In the next step, we should process the next most frequent character from the heap in the same way and then, at the end of this step, insert the character from the previous step back to the heap after decrementing its frequency.

Following this algorithm, if we can append all the characters from the input string to the output string, we would have successfully found an arrangement of the given string where no two same characters appeared adjacent to each other.

Code

Here is what our algorithm will look like:

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

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

Mark Course as Completed

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

JavaPython3C++JS JS

```
20    });
21
22    let previousChar = null,
23        previousFrequency = 0,
24        resultString = [];
25    while (maxHeap.length > 0) {
26        const [frequency, char] = maxHeap.pop();
27        // add the previous entry back in the heap if its frequency is greater than zero
28        if (previousChar !== null && previousFrequency > 0) {
29            maxHeap.push([previousFrequency, previousChar]);
30        }
31        // append the current character to the result string and decrement its count
32        resultString.push(char);
33        previousChar = char;
34        previousFrequency = frequency - 1; // decrement the frequency
35    }
36
37    // if we were successful in appending all the characters to the result string, return it
38    if (resultString.length === str.length) {
39        return resultString.join('');
40    }
41    return '';
42 }
43
44 console.log(`Rearranged string: ${rearrange_string('aappp')}`);
45 console.log(`Rearranged string: ${rearrange_string('Programming')}`);
```

RUN

SAVE

RESET

Close

13.044s

Output

Rearranged string: papap
Rearranged string: rgmrgmoaPin
Rearranged string:

Time complexity

The time complexity of the above algorithm is $O(N * \log N)$ where 'N' is the number of characters in the input string.

Space complexity

The space complexity will be $O(N)$, as in the worst case, we need to store all the 'N' characters in the **HashMap**.

Interviewing soon? We've partnered with **Hired** so that companies apply to you instead of you applying to them. [See how](#)

← Back

Sum of Elements (medium)

Problem Challenge 1

Next →

Mark as Completed

Report an Issue

Ask a Question