















Grokking the Coding Interview: Patterns for Coding Questions

6% completed

- ☐ Smallest Subarray with a given sum (easy)
- ☒ Longest Substring with K Distinct Characters (medium)
- ☒ Fruits into Baskets (medium)
- ☒ No-repeat Substring (hard)
- ☒ Longest Substring with Same Letters after Replacement (hard)
- ☒ Longest Subarray with Ones after Replacement (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
- ☐ Problem Challenge 4
- ☒ Solution Review: Problem Challenge 4

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 

Solution Review: Problem Challenge 4

We'll cover the following 

- Words Concatenation (hard)
- Solution
- Code
 - Time Complexity
 - Space Complexity

Words Concatenation (hard)

Given a string and a list of words, find all the starting indices of substrings in the given string that are a **concatenation of all the given words** exactly once **without any overlapping** of words. It is given that all words are of the same length.

Example 1:

```
Input: String="catfoxcat", Words=["cat", "fox"]
Output: [0, 3]
Explanation: The two substring containing both the words are "catfox" & "foxcat".
```

Example 2:

```
Input: String="catcatfoxfox", Words=["cat", "fox"]
Output: [3]
Explanation: The only substring containing both the words is "catfox".
```





Solution

This problem follows the **Sliding Window** pattern and has a lot of similarities with [Maximum Sum Subarray of Size K](#). We will keep track of all the words in a **HashMap** and try to match them in the given string. Here are the set of steps for our algorithm:

1. Keep the frequency of every word in a **HashMap**.
2. Starting from every index in the string, try to match all the words.
3. In each iteration, keep track of all the words that we have already seen in another **HashMap**.
4. If a word is not found or has a higher frequency than required, we can move on to the next character in the string.
5. Store the index if we have found all the words.


Code

Here is what our algorithm will look like:

 Java
 Python3
 C++
 JS

```

1 function find_word_concatenation(str, words) {
2   if (words.length === 0 || words[0].length === 0) {
3     return [];
4   }
5
6   wordFrequency = {};
7
8   words.forEach((word) => {
9     if (!(word in wordFrequency)) {
10      wordFrequency[word] = 0;
11    }
12    wordFrequency[word] += 1;
13  });
14
15  const resultIndices = [],
16    wordsCount = words.length,
17    wordLength = words[0].length;
18
19  for (i = 0; i < (str.length - wordsCount * wordLength) + 1; i++) {
20    const wordsSeen = {};
21    for (j = 0; j < wordsCount; j++) {
22      next_word_index = i + j * wordLength;
23      // Get the next word from the string
24      word = str.substring(next_word_index, next_word_index + wordLength);
25      if (!(word in wordFrequency)) { // Break if we don't need this word
26        break;
27      }
28    }
  
```

RUN
SAVE
RESET


Time Complexity

The time complexity of the above algorithm will be $O(N * M * Len)$ where 'N' is the number of characters in the given string, 'M' is the total number of words, and 'Len' is the length of a word.

Refer a Friend

Create

Pattern : 0/1 Knapsack (Dynamic Programming)

Pattern: Topological Sort (Graph)

Miscellaneous

Conclusions

Where to Go from Here

Mark Course as Completed

Space Complexity

The space complexity of the algorithm is $O(M)$ since at most, we will be storing all the words in the two **HashMaps**. In the worst case, we also need $O(N)$ space for the resulting list. So, the overall space complexity of the algorithm will be $O(M + N)$.

←

Back

Problem Challenge 4

MARK AS COMPLETED

Next →

Introduction

Report an Issue