





Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

5% completed

Q Search Course

ITITEL VIEW/ LIVIZ DX TÂ COEO)

Longest Substring with Same Letters after Replacement (hard)

(/courses/grokking-the-coding-interview/R8DVgjq78yR)

Longest Subarray with Ones after Replacement (hard) (/courses/grokking-the-codinginterview/B6VypRxPolJ)

Problem Challenge 1 (/courses/grokking-the-coding-interview/N8vB7OVYo2D)

Solution Review: Problem
Challenge 1
(/courses/grokking-the-coding-interview/N0o9QnPLKNv)

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:

Back To Course Home

Grokking the Coding Interview: Patterns for **Coding Questions**

5% completed



Search Course

ITILET VIEW/ TIMZ DX 19EDEUJ

Longest Substring with Same Letters after Replacement (hard)

(/courses/grokking-the-codinginterview/R8DVgjq78yR)

Longest Subarray with Ones after Replacement (hard) (/courses/grokking-the-codinginterview/B6VypRxPolJ)

Problem Challenge 1 (/courses/grokking-the-codinginterview/N8vB7OVYo2D)

Solution Review: Problem Challenge 1 (/courses/grokking-the-codinginterview/N0o9QnPLKNv)

Input: String="catcatfoxfox", Words=["cat", "fox"]

Output: [3]

Explanation: The only substring containing both the words is "catfox".



Solution

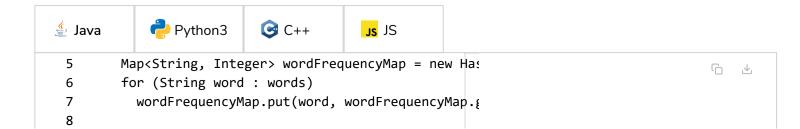
This problem follows the **Sliding Window** pattern and has a lot of similarities with Maximum Sum Subarray of Size K

(https://www.educative.io/collection/page/5668639101419520/5671464854355968/51770 43027230720/). 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:



Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

5% completed

Search Course

ITILET VIEW/ TIMZDX TREDEO!

Longest Substring with Same Letters after Replacement (hard)

(/courses/grokking-the-coding-interview/R8DVgjq78yR)

Longest Subarray with Ones after Replacement (hard) (/courses/grokking-the-codinginterview/B6VypRxPolJ)

Problem Challenge 1 (/courses/grokking-the-codinginterview/N8vB7OVYo2D)

Solution Review: Problem
Challenge 1
(/courses/grokking-the-coding-interview/N0o9QnPLKNv)

```
9
        List<Integer> resultIndices = new ArrayList<Int
        int wordsCount = words.length, wordLength = wor
10
11
        for (int i = 0; i <= str.length() - wordsCount</pre>
12
          Map<String, Integer> wordsSeen = new HashMap
13
          for (int j = 0; j < wordsCount; j++) {
14
15
            int nextWordIndex = i + j * wordLength;
            // get the next word from the string
16
            String word = str.substring(nextWordIndex,
17
18
            if (!wordFrequencyMap.containsKey(word)) //
19
              break;
20
            wordsSeen.put(word, wordsSeen.getOrDefault(
21
22
            // no need to process further if the word h
23
            if (wordsSeen.get(word) > wordFrequencyMap.
24
25
              break;
26
            if (j + 1 == wordsCount) // store index if
27
              resultIndices.add(i);
28
29
30
        }
31
        return resultIndices;
32
33
34
35
      public static void main(String[] args) {
 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.

Space Complexity #



Grokking the Coding Interview: Patterns for Coding Questions

5% completed

Q Search Course

ILLEL VIEW LINIZDX TAEDEO

interview/R8DVgjq78yR)

Longest Substring with Same Letters after Replacement (hard) (/courses/grokking-the-coding-

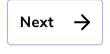
Longest Subarray with Ones after Replacement (hard) (/courses/grokking-the-codinginterview/B6VypRxPolJ)

Problem Challenge 1 (/courses/grokking-the-codinginterview/N8vB7OVYo2D)

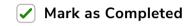
Solution Review: Problem
Challenge 1
(/courses/grokking-the-coding-interview/N0o9QnPLKNv)

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).





hallenge 4 Introduction



5% completed, meet the criteria and claim your course certificate!

Claim Certificate

① Report an Issue

? Ask a Question

(https://discuss.educative.io/tag/solution-review-problem-challenge-4_pattern-sliding-window_grokking-the-coding-interview-patterns-for-coding-questions)