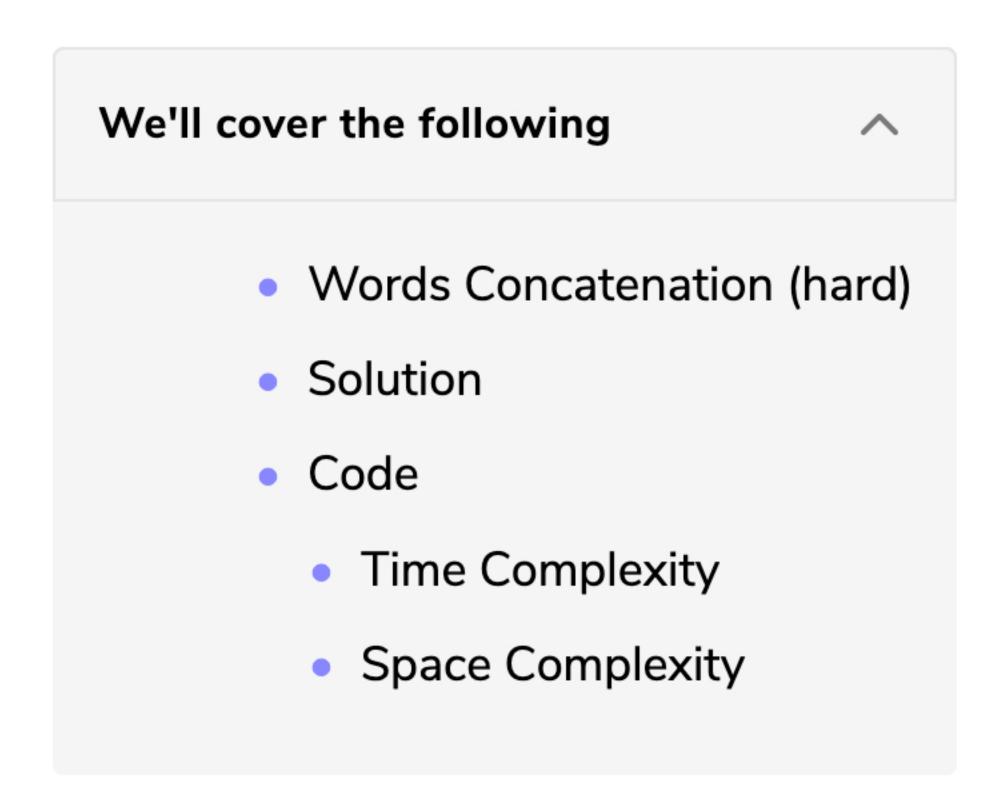


Introduction

Pair with Target Sum (easy)

Remove Duplicates (easy)

Solution Review: Problem Challenge 4



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.

₩

? Ask a Question

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:

```
Python3
                        ⊗ C++
                                    Js JS
👙 Java
16
            // get the next word from the string
                                                                                                     ₽
            String word = str.substring(nextWordIndex, nextWordIndex + wordLength);
             if (!wordFrequencyMap.containsKey(word)) // break if we don't need this word
18
19
              break;
20
21
            wordsSeen.put(word, wordsSeen.getOrDefault(word, 0) + 1); // add the word to the 'wordsSeen' map
22
            // no need to process further if the word has higher frequency than required
23
            if (wordsSeen.get(word) > wordFrequencyMap.getOrDefault(word, 0))
24
25
              break;
26
            if (j + 1 == wordsCount) // store index if we have found all the words
              resultIndices.add(i);
28
29
30
31
32
        return resultIndices;
33
34
      public static void main(String[] args) {
36
        List<Integer> result = WordConcatenation.findWordConcatenation("catfoxcat", new String[] { "cat", "
        System.out.println(result);
        result = WordConcatenation.findWordConcatenation("catcatfoxfox", new String[] { "cat", "fox" });
38
39
        System.out.println(result);
40
42
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

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).

