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


```
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/5668639101419520/5671464854355968/5177043027230720/). 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

```
    Map<String, Integer> wordFrequencyMap = new Has
    for (String word : words)
      wordFrequencyMap.put(word, wordFrequencyMap.g

```

Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

5% completed

Search Course


```
    List<Integer> resultIndices = new ArrayList<Int
    int wordsCount = words.length, wordLength = wor

    for (int i = 0; i <= str.length() - wordsCount
      Map<String, Integer> wordsSeen = new HashMap<
      for (int j = 0; j < wordsCount; j++) {
        int nextWordIndex = i + j * wordLength;
        // get the next word from the string
        String word = str.substring(nextWordIndex,
        if (!wordFrequencyMap.containsKey(word)) //
          break;

        wordsSeen.put(word, wordsSeen.getOrDefault(

        // no need to process further if the word h
        if (wordsSeen.get(word) > wordFrequencyMap.
          break;

        if (j + 1 == wordsCount) // store index if
          resultIndices.add(i);
      }
    }

    return resultIndices;
  }

  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

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 $O(M + N)$.