# Solution Review: Problem Challenge 4

We'll cover the following



## 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

```python
def find_word_concatenation(str1, words):
  if len(words) == 0 or len(words[0]) == 0:
    return []

  word_frequency = {}

  for word in words:
    if word not in word_frequency:
      word_frequency[word] = 0
```

```python
    word_frequency[word] += 1

  result_indices = []
  words_count = len(words)
  word_length = len(words[0])

  for i in range((len(str1) - words_count * word_length)+1):
    words_seen = {}
    for j in range(0, words_count):
      next_word_index = i + j * word_length
      # Get the next word from the string
      word = str1[next_word_index: next_word_index + word_length]
      if word not in word_frequency:  # Break if we don't need this word
        break

      # Add the word to the 'words_seen' map
      if word not in words_seen:
        words_seen[word] = 0
      words_seen[word] += 1

      # No need to process further if the word has higher frequency than required
      if words_seen[word] > word_frequency.get(word, 0):
        break

      if j + 1 == words_count:  # Store index if we have found all the words
        result_indices.append(i)

  return result_indices


def main():
  print(find_word_concatenation("catfoxcat", ["cat", "fox"]))
  print(find_word_concatenation("catcatfoxfox", ["cat", "fox"]))


main()
```

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 $O(M + N)$.