Daily Coding Problem #172

Problem

This problem was asked by Dropbox.

Given a string `s` and a list of words `words`, where each word is the same length, find all starting indices of substrings in `s` that is a concatenation of every word in `words` exactly once.

For example, given s = "dogcatcatcodecatdog" and words = ["cat", "dog"], return [0, 13], since "dogcat" starts at index 0 and "catdog" starts at index 13.

Given s = "barfoobazbitbyte" and words = ["dog", "cat"], return [] since there are no substrings composed of "dog" and "cat" in `s`.

The order of the indices does not matter.

Solution

One brute force solution here would be to iterate over every index of the string `s`, and check if the substring starting at that index matches the concatenation of any permutation of `words`.

```
from itertools import permutations

def starting_concatenations(s, words):
    result = []

    possible_substrings = [''.join(p) for p in permutations(words)]

    for i in range(len(s)):
        for substring in possible_substrings:
            if substring == s[i:i + len(substring)]:
                result.append(i)
```

```
    return result
```

Since we'd need to iterate over each permutation of words on each character, this would take a whopping O(n * w!) time and space, where n is the length of `s` and `w` is the size of `words`.

Since generating the permutations is the bottleneck here, we want to ideally speed up checking for matches. What we can do is keep a dictionary of word counts and search every `k`th word. Once we scan every k-sized substring we subtract it from the count in the dictionary. If we encounter a substring that isn't in `words` then we immediately move our index up.

We'll do this for every index in `range(len(k))` so as to cover every possible substring.

```
from collections import Counter

def starting_concatenations(s, words):
    if not words:
        return []

    k = len(words[0])
    result = []

    for i in range(k):
        c = Counter(words)
        for j in range(i + k, len(s) + 1, k):
            word = s[j - k: j]
            c[word] -= 1

            # No possible match: restore words and move i up.
            while c[word] < 0:
                c[s[i:i + k]] += 1
                i += k

            # Matched all words
            if i + k * len(words) == j:
                result.append(i)
    return result
```

This takes O(k *s* w) time and space, where k is the length of a word.