Daily Coding Problem #111

Problem

This problem was asked by Google.

Given a word W and a string S, find all starting indices in S which are anagrams of W.

For example, given that W is "ab", and S is "abxaba", return 0, 3, and 4.

Solution

Brute force

The brute force solution here would be to go over each word-sized window in S and check if they're anagrams, like so:

```
from collections import Counter

def is_anagram(s1, s2):
    return Counter(s1) == Counter(s2)

def anagram_indices(word, s):
    result = []
    for i in range(len(s) - len(word) + 1):
        window = s[i:i + len(word)]
        if is_anagram(window, word):
            result.append(i)
    return result
```

This would take O(|W| * |S|) time. Can we make this any faster?

Count difference

Notice that moving along the window seems to mean recomputing the frequency counts of the entire window, when only a little bit of it actually updated. This insight lead us to the following strategy:

- Make a frequency dictionary of the target word
- Continuously diff against it as we go along the string
- When the dict is empty, the window and the word matches

We diff in our frequency dict by incrementing the new character in the window and removing old one.

```
class FrequencyDict:
    def __init__(self, s):
        self.d = {}
        for char in s:
```

```
            self.increment(char)

    def _create_if_not_exists(self, char):
        if char not in self.d:
            self.d[char] = 0

    def _del_if_zero(self, char):
        if self.d[char] == 0:
            del self.d[char]

    def is_empty(self):
        return not self.d

    def decrement(self, char):
        self._create_if_not_exists(char)
        self.d[char] -= 1
        self._del_if_zero(char)

    def increment(self, char):
        self._create_if_not_exists(char)
        self.d[char] += 1
        self._del_if_zero(char)


def anagram_indices(word, s):
    result = []

    freq = FrequencyDict(word)

    for char in s[:len(word)]:
        freq.decrement(char)
```

```
    if freq.is_empty():
        result.append(0)

    for i in range(len(word), len(s)):
        start_char, end_char = s[i - len(word)], s[i]
        freq.increment(start_char)
        freq.decrement(end_char)
        if freq.is_empty():
            beginning_index = i - len(word) + 1
            result.append(beginning_index)

    return result
```

This should run in O(S) time.