Find All Anagrams in a String

Given a string **s** and a **non-empty** string **p**, find all the start indices of **p**'s anagrams in **s**.

Strings consists of lowercase English letters only and the length of both strings**s** and **p** will not be larger than 20,100.

The order of output does not matter.

Example 1:

```
Input:
s: "cbaebabacd" p: "abc"

Output:
[0, 6]

Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc".
The substring with start index = 6 is "bac", which is an anagram of "abc".
```

Example 2:

```
Input:
s: "abab" p: "ab"

Output:
[0, 1, 2]

Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab".
The substring with start index = 1 is "ba", which is an anagram of "ab".
The substring with start index = 2 is "ab", which is an anagram of "ab".
```

Solution 1

Same idea from a fantastic sliding window template, please refer: https://discuss.leetcode.com/topic/30941/here-is-a-10-line-template-that-can-solve-most-substring-problems

Time Complexity will be O(n) because the "start" and "end" points will only move from left to right once.

```
public List<Integer> findAnagrams(String s, String p) {
    List<Integer> list = new ArrayList<>();
    if (s == null || s.length() == 0 || p == null || p.length() == 0) return list;
    int[] hash = new int[256]; //character hash
    //record each character in p to hash
    for (char c : p.toCharArray()) {
        hash[c]++;
    }
    //two points, initialize count to p's length
    int left = 0, right = 0, count = p.length();
    while (right < s.length()) {
        //move right everytime, if the character exists in p's hash, decrease the
count
        //current hash value >= 1 means the character is existing in p
        if (hash[s.charAt(right++)]-- >= 1) count--;

        //when the count is down to 0, means we found the right anagram
        //then add window's left to result list
        if (count == 0) list.add(left);

        //if we find the window's size equals to p, then we have to move left (nar
row the window) to find the new match window
        //++ to reset the hash because we kicked out the left
        //only increase the count if the character is in p
        //the count >= 0 indicate it was original in the hash, cuz it won't go bel
ow 0
        if (right - left == p.length() && hash[s.charAt(left++)]++ >= 0) count++;
    }
    return list;
}
```

written by NathanNi original link here

Solution 2

```
public List<Integer> findAnagrams(String s, String p) {
    int[] chars = new int[26];
    List<Integer> result = new ArrayList<>();

    if (s == null || p == null || s.length() < p.length())
        return result;
    for (char c : p.toCharArray())
        chars[c-'a']++;

    int start = 0, end = 0, count = p.length();
    // Go over the string
    while (end < s.length()) {
        // If the char at start appeared in p, we increase count
        if (end - start == p.length() && chars[s.charAt(start++)-'a']++ >= 0)
            count++;
        // If the char at end appeared in p (since it's not -1 after decreasing),
we decrease count
        if (--chars[s.charAt(end++)-'a'] >= 0)
            count--;
        if (count == 0)
            result.add(start);
    }

    return result;
}
```

written by dahui original link here

Solution 3

```
    private List<Integer> findAnagrams(String s, String p) {
        List<Integer> res = new ArrayList<>();
        int[] win = new int[256];
        int[] pFixWin = new int[256];

        if (s.length() == 0 || p.length() == 0 || p.length() > s.length()) {
            return res;
        }

        /// pre - load the pWin
        for (int i = 0; i < p.length(); i++) {
            pFixWin[p.charAt(i)]++;
        }

        // pre-load the moving  window
        for (int i = 0; i < p.length(); i++) {
            win[s.charAt(i)]++;
        }

        for (int i = 0; i < s.length(); i++) {
            // for each position check if the numbers of each letter
            // coincide in the window and the fixed window
            if (isPContainedInS(pFixWin, win)) {
                res.add(i);
            }

            // evict from the window the char we just passed in S
            // add from the S window the next character
            if ((i + p.length()) < s.length()) {
                win[s.charAt(i)]--;
                win[s.charAt(i + p.length())]++;
            } else {
                break;
            }

        }
        return res;

    }

    private boolean isPContainedInS(int[] pFixWin, int[] win) {
        for (int i = 0; i < pFixWin.length; i++) {
            if (pFixWin[i] != win[i]) {
                return false;
            }
        }
        return true;
    }
```

written by federico original link here