438. Find All Anagrams in a String

Medium 3775 194 Add to List Share

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".
```

While add right, update window counter.
When window.length == s.length, shrink window.
While add left, reduce window counter.
Before shrink window, update the final result.

```cpp
class Solution {
public:
    vector<int> findAnagrams(string s, string p) {
        unordered_map<char, int> need, window;
        for (char c : p) {
            need[c]++;
        }

        int left = 0, right = 0;
        int valid = 0;

        vector<int> res;

        while(right < s.size()) {
            // c is the char adding to window
            char c = s[right];
            // move right side of window
            right++;
            // update window counter and valid
            if (need.count(c)) {
                window[c]++;
                if (window[c] == need[c]) {
                    valid++;
                }
            }

            // cout << "window: [" << left << "," << right << ")" << endl;

            // when window's length is larger than s's length, shrink window
            while(right - left == p.size()) {
                // update the final result
                if (valid == need.size()) {
                    res.push_back(left);
                }
                // d is the char removing from window
                char d = s[left];
                // move left size of window;
                left++;
                // update window counter and valid
                if (need.count(d)) {
                    if (window[d] == need[d]) {
                        valid--;
                    }
                    window[d]--;
                }
            }
        }
        return res;
    }
};
```