题目地址（438. 找到字符串中所有字母异位词）

https://leetcode-cn.com/problems/find-all-anagrams-in-a-string/

入选理由

1. 依然是经典的滑动窗口，相信你做完这道题应该对滑动窗口的套路有了一定的理解了。 掌握知识的最笨但却好用的方法就是重复重复再重复。

标签

- 滑动窗口

难度

- 中等

题目描述

给定一个字符串 s 和一个非空字符串 p，找到 s 中所有是 p 的字母异位词的子串，返回这些子串的起始索引。

字符串只包含小写英文字母，并且字符串 s 和 p 的长度都不超过 20100。

说明：

字母异位词指字母相同，但排列不同的字符串。
不考虑答案输出的顺序。
示例 1:

输入:
s: "cbaebabacd" p: "abc"

输出:
[0, 6]

解释:
起始索引等于 0 的子串是 "cba", 它是 "abc" 的字母异位词。
起始索引等于 6 的子串是 "bac", 它是 "abc" 的字母异位词。
 示例 2:

输入:
s: "abab" p: "ab"

输出：
[0, 1, 2]

解释：
起始索引等于 0 的子串是 "ab"，它是 "ab" 的字母异位词。
起始索引等于 1 的子串是 "ba"，它是 "ab" 的字母异位词。
起始索引等于 2 的子串是 "ab"，它是 "ab" 的字母异位词。

前置知识

- Sliding Window
- 哈希表

思路

> 咳咳，暴力题解俺就不写了哈，因为和昨天基本一致

来分析一下，首先题中说找到 s 中所有是 p 的字母异位词的字串，就这句话，就包含了如下两个重要信息：

- 找到符合要求的子串长度都是 p
- 何为字母异位词？也就是我们不关心 p 这个串的顺序，只关心字母是否出现以及出现的次数，这种问题解决方案一般有两种，一种是利用排序强制顺序，另一种就是用哈希表的方法。

这么一抽象，是不是和昨天那个题很相似呢？那么问题的关键就是：

- 如何构建滑窗
- 如何更新状态，也即如何存储 p 串及更新窗口信息

针对问题 1 很容易，因为是长度固定为 p 的滑动窗口，而针对如何存储 p 串这个问题，我们可以考虑用桶来装，这个桶既可以用 26 个元素的数组（作用其实也是哈希表）也可以用哈希表

那么我们解决方案就很明朗了：

- 初始化个滑窗
- 不断移动该固定窗口，并用一个 rest 变量来记录剩余待匹配字符的个数
- 只要当前窗口符合要求，即把窗口左指针下标添加到结果集合中去。

代码

代码支持：Java，CPP，Python3

Java Code：

```java
public List<Integer> findAnagrams(String s, String p) {

    List<Integer> res = new LinkedList<>();
    if (s == null || p == null || s.length() < p.length())
        return res;

    int[] ch = new int[26];
    //统计p串字符个数
    for (char c : p.toCharArray())
        ch[c - 'a']++;
    //把窗口扩成p串的长度
    int start = 0, end = 0, rest = p.length();
    for (; end < p.length(); end++) {
        char temp = s.charAt(end);
        ch[temp - 'a']--;
        if (ch[temp - 'a'] >= 0)
            rest--;
    }

    if (rest == 0)
        res.add(0);
    //开始一步一步向右移动窗口。
    while (end < s.length()) {
        //左边的拿出来一个并更新状态
        char temp = s.charAt(start);
        if (ch[temp - 'a'] >= 0)
            rest++;
        ch[temp - 'a']++;
        start++;
        //右边的拿进来一个并更新状态
        temp = s.charAt(end);
        ch[temp - 'a']--;
        if (ch[temp - 'a'] >= 0)
            rest--;
        end++;
        // 状态合法就存到结果集合
        if (rest == 0)
            res.add(start);
    }

    return res;
}
```

CPP Code:

```
class Solution {
public:
    vector<int> findAnagrams(string s, string p) {
        vector<int> res, hash(26, 0), hashZero(26,0);
        if(s.length() < p.length()) return res;

        for(int i = 0; i < p.length(); i++){
            hash[p[i] - 'a']++;
            hash[s[i] - 'a']--;
        }
        if(hash == hashZero) res.push_back(0);

        for(int i = p.length(); i < s.length(); i++){
            hash[s[i] - 'a']--;
            hash[s[i - p.length()] - 'a']++;
            if(hash == hashZero) res.push_back(i - p.length() + 1);
        }

        return res;
    }
};
```

Python 解法具体做法稍有一点不同，没有使用 rest 变量，而是直接取的哈希表的长度。其中 哈希表的 key 是字符，value 是窗口内字符出现次数。这样当 value 为 0 时，我们移除 key，这样当哈希表容量为 0，说明我们找到了一个异位词。

Python3 Code:

```
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        target = collections.Counter(p)
        ans = []
        for i in range(len(s)):
            if i >= len(p):
                target[s[i - len(p)]] += 1
                if target[s[i - len(p)]] == 0:
                    del target[s[i - len(p)]]
            target[s[i]] -= 1
            if target[s[i]] == 0:
                del target[s[i]]
            if len(target) == 0:
                ans.append(i - len(p) + 1)
        return ans
```

你也可以将窗口封装成一个类进行操作。虽然代码会更长，但是如果你将窗口类看成黑盒，那么逻辑会很简单。

这里我提供一个 Python3 版本的**封装类解法**。

```python
class FrequencyDict:
    def __init__(self, s):
        self.d = collections.Counter()
        for char in s:
            self.increment(char)

    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.d[char] -= 1
        self._del_if_zero(char)

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


class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        ans = []

        freq = FrequencyDict(p)

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

        if freq.is_empty():
            ans.append(0)

        for i in range(len(p), len(s)):
            start, end = s[i - len(p)], s[i]
            freq.increment(start)
            freq.decrement(end)
            if freq.is_empty():
                ans.append(i - len(p) + 1)

        return ans
```

复杂度分析

令 s 的长度为 n。

- 时间复杂度：$O(n)$

- 空间复杂度：虽然我们使用了数组（或者哈希表）存储计数信息，但是大小不会超过 26，因此空间复杂度为 $O(1)$。

上一页 下一页