题目地址(1456. 定长子串中元音的最大数目)

https://leetcode-cn.com/problems/maximum-number-of-vowels-in-a-substring-of-given-length

入选理由

1. 滑动窗口第一道题，难度不算大，而且是非常常规的题目。

标签

- 滑动窗口

难度

- 中等

题目描述

给你字符串 s 和整数 k 。

请返回字符串 s 中长度为 k 的单个子字符串中可能包含的最大元音字母数。

英文中的 元音字母 为（a, e, i, o, u）。

示例 1:

输入: s = "abciiidef", k = 3
输出: 3
解释: 子字符串 "iii" 包含 3 个元音字母。
示例 2:

输入: s = "aeiou", k = 2
输出: 2
解释: 任意长度为 2 的子字符串都包含 2 个元音字母。
示例 3:

输入: s = "leetcode", k = 3
输出: 2
解释: "lee"、"eet" 和 "ode" 都包含 2 个元音字母。
示例 4:

输入：s = "rhythms", k = 4
输出：0
解释：字符串 s 中不含任何元音字母。
示例 5：

输入：s = "tryhard", k = 4
输出：1

提示：

1 <= s.length <= 10^5
s 由小写英文字母组成
1 <= k <= s.length

前置知识

- 滑动窗口[1]
- 哈希表

思路

拿这个题作为本专题第一道滑动窗口练手题再合适不过了，题目直观清晰。

- 元音字母有五个，为了避免我们总要写 if 啊 switch 啊这种，我们可以用个哈希表来存着方便后续查找是否存在。
- 题目要求我们找出所有 k 长度子串中可能包含的最大元音字母数，那我们遍历一边所有长度为 k 的子串不就知道啦 → 暴力，上代码

```
public int maxVowels(String s, int k) {

    if (s == null || s.length() < k)
        return 0;

    int res = 0;
    Set<Character> set = new HashSet<>(){{
        add('a');add('e');add('i');add('o');add('u');
    }};

    for (int i = 0; i < s.length() - k + 1; i++) {

        String sub = s.substring(i, i + k);
        int count = 0;

        for (int j = 0; j < sub.length(); j++)
            if (set.contains(sub.charAt(j)))
                count++;
```

```
            res = Math.max(res, count);
        }

        return res;
    }
```

很直观，但是提交会发现 TLE 了，我们也不难发现复杂度为$O((N - K + 1) * K)$，有什么优化方法呢？其实也容易想到：

- 利用前缀和，只不过我们前缀和数组元素 i 存的是子串 0..i 的元音字母个数，这样再遍历一遍前缀和数组就可以求出结果 → 前缀和方案（有兴趣可以自行实现）
- 其实我们完全没有必要去构建这个前缀和数组，我们维护一个窗口大小为 k 的滑窗即可，每移动一次可以归纳为：
  - 窗口左端弹出一个字符（删除步）
  - 若删除了元音则计数器-1（更新步）
  - 窗口右端加进来一个字符（添加步）
  - 若添加的字符是元音则计数器+1（更新步）
- 这样就得到了 → 滑动窗口解决方案

> 前面的部分也提到了前缀和， 你能试着总结一下和前缀和相关的考点有哪些么？

代码

代码支持：Python, Java, JS, CPP

Python Code:

```
class Solution:
    def maxVowels(self, s: str, k: int) -> int:
        res = 0
        temp = 0
        vowels = set(['a','e','i','o','u'])
        for i in range(k):
            res += s[i] in vowels
        if res==k: return k
        temp = res
        for i in range(k,len(s)):
            temp += (s[i] in vowels) - (s[i-k] in vowels)
            res = max(temp,res)
            if res ==k: return k
        return res
```

Java Code：

```java
public int maxVowels(String s, int k) {

    if (s == null || s.length() < k)
        return 0;

    int res = 0;
    Set<Character> set = new HashSet<>(){{
        add('a');add('e');add('i');add('o');add('u');
    }};

    // init
    for (int i = 0; i < k; i++)
        if (set.contains(s.charAt(i)))
            res++;

    int cur = res;
    for (int i = 1; i < s.length() - k + 1; i++) {

        if (set.contains(s.charAt(i - 1)))
            cur--;
        if (set.contains(s.charAt(i + k - 1)))
            cur++;

        res = Math.max(res, cur);
    }

    return res;
}
```

JS Code:

```javascript
var maxVowels = function (s, k) {
  const dict = new Set(["a", "e", "i", "o", "u"]);
  let ret = 0;
  for (let i = 0; i < k; i++) {
    if (dict.has(s[i])) ret++;
  }

  let temp = ret;
  for (let i = k, j = 0; i < s.length; i++, j++) {
    if (dict.has(s[i])) temp++;
    if (dict.has(s[j])) temp--;

    ret = Math.max(temp, ret);
  }
```

```
  return ret;
};
```

CPP Code:

```
class Solution {
public:
    int maxVowels(string s, int k) {
      unordered_set<char> vowels = {'a', 'e', 'i', 'o', 'u'};
      int cnt = 0, ans = 0;
      int n = s.size();
      for (int i = 0; i < n; i++) {
        if (i >= k) cnt -= vowels.count(s[i - k]);
        cnt += vowels.count(s[i]);
        ans = max(ans, cnt);
      }
      return ans;
    }
};
```

复杂度分析

- 时间复杂度：$O(n)$，n 为字串长度
- 空间复杂度：$O(1)$

参考资料

[1] 滑动窗口: *https://github.com/azl397985856/leetcode/blob/master/thinkings/slide-window.md*