0395. 至少有 K 个重复字符的最长子串

ITCharge 大约 3 分钟

- 标签：哈希表、字符串、分治、滑动窗口
- 难度：中等

题目链接

- 0395. 至少有 K 个重复字符的最长子串 - 力扣

题目大意

给定一个字符串 `s` 和一个整数 `k` 。

要求：找出 `s` 中的最长子串， 要求该子串中的每一字符出现次数都不少于 `k` 。返回这一子串的长度。

注意： `s` 仅由小写英文字母构成。

解题思路

这道题看起来很像是常规滑动窗口套路的问题，但是用普通滑动窗口思路无法解决问题。

如果窗口需要保证「各种字符出现次数都大于等于 `k` 」这一性质。那么当向右移动 `right` ，扩大窗口时，如果 `s[right]` 是第一次出现的元素，窗口内的字符种类数量必然会增加，此时缩小 `s[left]` 也不一定满足窗口内「各种字符出现次数都大于等于 `k` 」这一性质。那么我们就无法通过这种方式来进行滑动窗口。

但是我们可以通过固定字符种类数的方式进行滑动窗口。因为给定字符串 `s` 仅有小写字母构成，则最长子串中的字符种类数目，最少为 `1` 种，最多为 `26` 种。我们通过枚举最长子串中可能出现的字符种类数目，从而固定窗口中出现的字符种类数目 `i` （`1 <= i <= 26`） ，再进行滑动数组。窗口内需要保证出现的字符种类数目等于 `i` 。向右移动 `right` ，扩大窗口时，记录窗口内各种类字符数量。当窗口内出现字符数量大于 `i` 时，则不断右移 `right` ，保证窗口内出现字符种类等于 `i` 。同时，记录窗口内出现次数小于 `k` 的字符数量，当窗口中出现次数小于 `k` 的字符数量为 `0` 时，就可以记录答案，并维护答案最大值了。

整个算法的具体步骤如下：

- 使用 `ans` 记录满足要求的最长子串长度。
- 枚举最长子串中的字符种类数目 `i` ，最小为 `1` 种，最大为 `26` 种。对于给定字符种类数目 `i` ：
  - 使用两个指针 `left` 、 `right` 指向滑动窗口的左右边界。
  - 使用 `window_count` 变量来统计窗口内字符种类数目，保证窗口中的字符种类数目 `window_count` 不多于 `i` 。
  - 使用 `letter_map` 哈希表记录窗口中各个字符出现的数目。使用 `less_k_count` 记录窗口内出现次数小于 `k` 次的字符数量。
  - 向右移动 `right` ，将最右侧字符 `s[right]` 加入当前窗口，用 `letter_map` 记录该字符个数。
  - 如果该字符第一次出现，即 `letter_map[s[right]] == 1` ，则窗口内字符种类数目 + 1，即 `window_count += 1` 。同时窗口内小于 `k` 次的字符数量 + 1（等到 `letter_map[s[right]] >= k` 时再减去），即 `less_k_count += 1` 。
  - 如果该字符已经出现过 `k` 次，即 `letter_map[s[right]] == k` ，则窗口内小于 `k` 次的字符数量 -1，即 `less_k_count -= 1` 。
  - 当窗口内字符种类数目 `window_count` 大于给定字符种类数目 `i` 时，即 `window_count > i` ，则不断右移 `left` ，缩小滑动窗口长度，直到 `window_count == i` 。
  - 如果此时窗口内字符种类数目 `ow_count` 等于给定字符种类 `i` 并且小于 `k` 次的字符数量为 `0` ，即 `window_count == i and less_k_count == 0` 时，维护更新答案为 `ans = max(right - left + 1, ans)` 。
- 最后输出答案 `ans` 。

代码

```py
class Solution:
    def longestSubstring(self, s: str, k: int) -> int:
        ans = 0
        for i in range(1, 27):
            left, right = 0, 0
            window_count = 0
            less_k_count = 0
            letter_map = dict()
            while right < len(s):
                if s[right] in letter_map:
                    letter_map[s[right]] += 1
```

```
            else:
                letter_map[s[right]] = 1

            if letter_map[s[right]] == 1:
                window_count += 1
                less_k_count += 1
            if letter_map[s[right]] == k:
                less_k_count -= 1

            while window_count > i:
                letter_map[s[left]] -= 1
                if letter_map[s[left]] == 0:
                    window_count -= 1
                    less_k_count -= 1
                if letter_map[s[left]] == k - 1:
                    less_k_count += 1
                left += 1

            if window_count == i and less_k_count == 0:
                ans = max(right - left + 1, ans)
            right += 1
    return ans
```