# 题目

给你一个字符串 s 和一个整数 k ，请你找出 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。

如果不存在这样的子字符串，则返回 0。

示例 1：

输入：s = "aaabb", k = 3

输出：3

解释：最长子串为 "aaa" ，其中 'a' 重复了 3 次。

示例 2：

输入：s = "ababbc", k = 2

输出：5

解释：最长子串为 "ababb" ，其中 'a' 重复了 2 次， 'b' 重复了 3 次。

提示：

1 <= s.length <= 104

s 仅由小写英文字母组成

1 <= k <= 105

# 分析

要解决“找到字符串中每个字符出现次数都不少于k的最长子串”问题，核心思路是分治递归：通过排除出现次数不足k的字符，将原问题拆解为更小的子问题，递归求解子问题的最长有效子串长度。

解题思路：

1、核心观察：

若某字符在整个字符串中的出现次数小于k，则任何包含该字符的子串都不可能是有效子串（因该字符的出现次数无法满足≥k）。因此，可将该字符作为“分隔符”，将字符串分割为多个子串，递归求解每个子串的最长有效长度。

2、分治步骤：

- 统计字符频次：计算当前字符串中每个字符的出现次数。

- 寻找分隔符：找出所有出现次数 < k的字符（分隔符）。

- 递归分割：若不存在分隔符，当前字符串整体有效，返回其长度；否则，按分隔符分割字符串为多个子串，递归计算每个子串的最长有效长度，取最大值。

3、终止条件：

当子串长度小于k时，不可能存在有效子串，返回0。

代码：

class Solution {

public:

int longestSubstring(string s, int k) {

// 递归终止条件：子串长度小于k，不可能有效

if (s.size() < k) {

return 0;

}

// 统计当前字符串中每个字符的出现次数（仅小写字母）

vector<int> count(26, 0);

for (char c : s) {

count[c - 'a']++;

}

// 寻找出现次数 < k 的字符（分隔符）

char split = 0;

for (int i = 0; i < 26; ++i) {

if (count[i] > 0 && count[i] < k) {

split = 'a' + i;

break;

}

}

// 若没有分隔符，当前字符串整体有效

if (split == 0) {

return s.size();

}

// 按分隔符分割字符串，递归处理每个子串

int max\_len = 0;

size\_t pos = 0;

size\_t next\_pos;

while ((next\_pos = s.find(split, pos)) != string::npos) {

// 递归计算[pos, next\_pos)子串的最长有效长度

string sub = s.substr(pos, next\_pos - pos);

max\_len = max(max\_len, longestSubstring(sub, k));

pos = next\_pos + 1; // 跳过分隔符

}

// 处理最后一个子串（从pos到字符串末尾）

string last\_sub = s.substr(pos);

max\_len = max(max\_len, longestSubstring(last\_sub, k));

return max\_len;

}

};

代码解释：

1、频次统计：用大小为26的数组count记录每个小写字母的出现次数，时间复杂度O(n)（n为当前字符串长度）。

2、分隔符判断：遍历频次数组，找到第一个出现次数 < k的字符作为分隔符。若不存在，当前字符串整体有效，直接返回长度。

3、递归分割：使用find函数按分隔符分割字符串为多个子串，对每个子串递归调用longestSubstring，取返回值的最大值作为当前问题的解。

4、终止条件处理：当子串长度小于k时，直接返回0，避免无效计算。

复杂度分析

- 时间复杂度：O(26n)，其中n是字符串长度。每次递归至少排除一个字符（作为分隔符），最多递归26次（因小写字母共26个），每次递归的字符串处理（统计频次、分割）时间为O(n)，故总时间为O(26n)，即O(n)。

- 空间复杂度：O(26 + n)。递归栈深度最多26层（常数级），每层递归需存储子串（最坏情况下为O(n)），故空间复杂度为O(n)。