Take K Of Each Character From Left And Right

solved by	Senan
Platform	LeetCode
difficulty	Medium
# Serial	2516
tags	Sliding Window
language	C++
solved on	@20/11/2024
link	https://leetcode.com/problems/take-k-of-each-character-from-left-and-right/submissions/1457784267/
Completion	☑

Intuition

To satisfy the condition where every character appears at least `k` times, we need to determine the minimum number of characters to take from the string `s`. The problem becomes one of finding the largest valid substring in the string `s` where all characters appear at least `k` times and calculating the complement of its length.

Approach

1. **Count Initial Frequencies**: First, count the frequencies of each character (`a`, `b`, `c`) in the string `s`. If any character has a frequency less than `k`, it is impossible to meet the requirement, and we return `1`.
2. **Sliding Window**: Use a two-pointer technique to traverse the string:
   - Maintain a window `[l, r]` where the substring satisfies the frequency condition.
   - For each character added to the window (`r`), decrement its count.
   - If the count of any character in the window falls below `k`, adjust the left pointer (`l`) by incrementing its count back until the condition is restored.
3. **Minimize Characters to Remove**: The solution involves calculating the smallest length of the complement of the valid substring `[l, r]` to minimize the number of characters to remove. Update the result (`ans`) during each step.

Complexity

Time Complexity:

- **O(n)**: We traverse the string once with the sliding window approach, where each character is processed at most twice (once when added to the window and once when removed).

Space Complexity:

- **O(1)**: We use a fixed-size vector `count` of size 3 for tracking frequencies of the three characters.

Code

```
class Solution {
 public:
  int takeCharacters(string s, int k) {
    int n = s.size();
    int ans = n;

    vector<int> count(3, 0);

    // Count frequencies of 'a', 'b', 'c'
    for (char c : s) count[c - 'a']++;
    if (count[0] < k || count[1] < k || count[2] < k) return -1;

    // Sliding window approach
    for (int l = 0, r = 0; r < n; r++) {
      count[s[r] - 'a']--;
      while (count[s[r] - 'a'] < k) count[s[l++] - 'a']++;
      ans = min(ans, n - (r - l + 1));
    }

    return ans;
  }
};
```