13.01.2025

Problem: Minimum Length of String After Operations

Problem Statement: You are given a string `s`, and you can perform the following operation any number of times:

- Choose an index `i` such that there is at least one character to the left of index `i` that is equal to `s[i]` and at least one character to the right that is also equal to `s[i]`.
- Remove the closest character to the left of index `i` that is equal to `s[i]`, and remove the closest character to the right of index `i` that is equal to `s[i]`.

You need to return the minimum length of the final string that can be achieved by performing these operations.

Key Insights:

1. **String Reduction Process:**
   a. The process described is essentially about removing characters from both the left and right side of an index if there are duplicate characters at that index. This operation can be repeated as long as it's possible to find duplicates around a character.
2. **Character Frequencies:**
   a. The goal is to minimize the string length by removing duplicate characters that meet the conditions.
   b. A frequency count for each character helps determine how many characters can be ignored when reducing the string.
3. **Optimal Removal Strategy:**
   a. For each character, if it appears more than twice, you can remove its excess characters. Specifically:
      i. If a character appears 3 times, remove 2 characters (since the remaining character can’t be reduced further).
      ii. This operation can be generalized: remove pairs of duplicates whenever possible.
4. **String Length Reduction:**
   a. The key is to track how many characters can be "ignored" or "removed" based on their frequency. The final length of the string is simply the initial length minus the number of characters that can be removed.

Solution Approach:

1. **Frequency Array:**

   a. Use a frequency array `charFreq` of size 26 to store the count of each character (since the string consists of lowercase English letters only).

2. **Track Ignored Characters:**
   a. Use an integer `ignoreChars` to keep track of the number of characters that can be removed. For each character that appears 3 or more times, reduce its count by 2 and add 2 to `ignoreChars`.

3. **Return the Final Length:**
   a. The final length of the string is calculated as the initial length minus the number of ignored characters.

Code Implementation:

```
class Solution {
public:
    int minimumLength(string s) {
        // to store the frequency of each character.
        vector<int> charFreq(26, 0);
        // to maintain the count of ignorable characters (those that can be
removed).
        int ignoreChars = 0;

        // Traverse the string to calculate frequency of each character
        for (int i = 0; i < s.length(); i++) {
            charFreq[s[i] - 'a'] += 1;  // Increment the count of the current
character

            // If the frequency of the character reaches 3, we can remove two
characters
            if (charFreq[s[i] - 'a'] == 3) {
                // Remove two characters (ignoring them)
                charFreq[s[i] - 'a'] -= 2;
                ignoreChars += 2;  // Add two to the count of ignored characters
            }
        }

        // Return the length of the string after removing the ignored characters
        return s.length() - ignoreChars;
    }
};
```

Explanation of the Code:

1. **Frequency Array (`charFreq`):**
   a. This array stores the frequency of each character. Each index of the array corresponds to a letter of the alphabet (e.g., index 0 for 'a', index 1 for 'b', etc.).

2. **Counting Ignored Characters (`ignoreChars`):**

   a. We increment the `ignoreChars` whenever a character appears more than twice. If a character appears 3 times, we "ignore" two of those occurrences, making them removable.

3. **Loop Through String (`s`):**
   a. For each character in the string `s`, we increment its frequency in the `charFreq` array.
   b. If a character appears 3 times, we adjust the frequency and increase the `ignoreChars` count by 2, as two of the occurrences can be ignored.

4. **Return the Result:**
   a. The result is simply the length of the string minus the ignored characters, giving the minimum length after performing the operations.

Time Complexity:

- **O(n)**, where `n` is the length of the string `s`. We loop through the string once and perform constant-time operations inside the loop (updating the frequency array and checking conditions).

Space Complexity:

- **O(1)**, since the frequency array has a fixed size of 26 (one for each letter of the alphabet), and we use a few additional variables for the count of ignored characters.

Example Walkthrough:

1. **Example 1:**
   a. Input: `s = "abaacbcbb"`
   b. Process:
      i. First character: 'a' (frequency: 1)
      ii. Second character: 'b' (frequency: 1)
      iii. Third character: 'a' (frequency: 2)
      iv. Fourth character: 'a' (frequency: 3) → Ignore two 'a's.
      v. Continue processing, ignoring duplicates when possible.
   c. Output: `5` (after ignoring two characters).

2. **Example 2:**
   a. Input: `s = "aa"`
   b. Process:
      i. Both characters are 'a' and appear only twice, so no operation is performed.
   c. Output: `2`.

Conclusion:

- This approach efficiently computes the minimum possible length of the string by removing characters that appear more than twice, resulting in an optimized solution with linear time complexity.