04.01.2025

Problem: Unique Length-3 Palindromic Subsequences

Problem Statement:
You are given a string `s`. Return the number of unique palindromic subsequences of length 3 that are a subsequence of `s`.

- A palindrome reads the same forward and backward.
- A subsequence is obtained by deleting some characters without changing the order of the remaining characters.

Example 1:
Input: `s = "aabca"`
Output: `3`
Explanation:
The unique palindromic subsequences of length 3 are: `"aba"`, `"aaa"`, `"aca"`.

Example 2:
Input: `s = "adc"`
Output: `0`
Explanation: No palindromic subsequences of length 3.

Example 3:
Input: `s = "bbcbaba"`
Output: `4`
Explanation: The unique palindromic subsequences of length 3 are: `"bbb"`, `"bcb"`, `"bab"`, `"aba"`.

Approach (Greedy + Hash Set)

Key Idea:

- A palindrome of length 3 has the form `"x_y_x"`.
- To count all such unique palindromic subsequences:
  a. Identify the **first** and **last** occurrence of each character in the string.
  b. Count the unique characters between these two indices (they form the middle part of the palindrome).

Steps:

1. **Initialize Arrays:**
   a. Use two arrays `first` and `last` of size 26 (for all lowercase letters) to store the first and last index of each character.
2. **Populate First and Last Occurrences:**
   a. Traverse the string and update the first and last occurrences for each character.

3. **Count Unique Palindromic Subsequences:**
    a. For each character (`a` to `z`):
        i. If `first[i] < last[i]`:
            1. Find the unique characters between `first[i]` and `last[i]` using a set.
            2. Add the count of unique characters to the answer.
4. **Return the Final Count.**

Code:

```
class Solution {
public:
    int countPalindromicSubsequence(string s) {
        int ans = 0;

        // To store first and last occurrence of each character
        vector<int> first(26, s.length());
        vector<int> last(26, -1); // Initialize with -1 for characters not present

        // Populate first and last occurrence
        for (int i = 0; i < s.length(); ++i) {
            int index = s[i] - 'a';
            first[index] = min(first[index], i);
            last[index] = i;
        }

        // Count unique palindromic subsequences
        for (int i = 0; i < 26; ++i) {
            if (first[i] < last[i]) {
                // Create a set to hold unique characters between first and last
occurrence
                unordered_set<char> uniqueChars(s.begin() + first[i] + 1,
s.begin() + last[i]);
                ans += uniqueChars.size();  // Count the unique characters
            }
        }

        return ans;
    }
};
```

Explanation:

For `s = "aabca"`:

- First and last occurrence of `'a'`: `first[a] = 0`, `last[a] = 4`
- Characters between index `0` and `4`: `b, c`

- Unique palindromic subsequences formed: `"aba"`, `"aca"`, `"aaa"`.

Complexity Analysis:

- **Time Complexity:** `O(n + 26 * n)` → The loop iterates through the string once, and for each character, it processes the substring using a hash set.
- **Space Complexity:** `O(26)` → The `first` and `last` arrays hold data for 26 characters, and the hash set also can store at most 26 unique characters.

Edge Cases Handled:

- If the string length is less than 3 → Return `0`.
- If no palindrome of length 3 can be formed → Return `0`.

Key Takeaways:

- This solution leverages **prefix and suffix analysis** to reduce the complexity.
- Using **hash sets** helps efficiently count unique elements.
- Great problem to practice **greedy approaches** and **set operations**.