Documentation

1. Problem Statement

The problem requires finding the length of the longest palindrome that can be formed using a given string backwards, consisting of lowercase and/or uppercase English letters. A palindrome reads the same forwards and backwards. It is important to note that the problem is case-sensitive, meaning that "a" and "A" are treated as distinct characters. We need to return the length of the longest palindrome that can be created by rearranging the characters of the input string.

2. Intuition

A palindrome is a string that remains the same when read from left to right and right to left. To form a palindrome from a given string, we need to use pairs of identical characters that will appear on both sides of the palindrome. If a character appears an odd number of times, one occurrence can be placed at the center of the palindrome, while the others are paired. Therefore, the problem boils down to counting character frequencies and determining how many pairs we can form, with at most one character placed at the center.

3. Key Observations

First, characters that appear an even number of times can directly form pairs, contributing fully to the palindrome's length. Characters with odd frequencies can only contribute the even part (i.e., the count minus one) to the palindrome's length. For instance, if a character appears 5 times, it can contribute 4 times (forming 2 pairs), and one occurrence can be placed in the middle if no other character has been placed there yet. If at least one character has an odd frequency, we can add one to the length for the center character. The string is case-sensitive, meaning that characters like "a" and "A" are treated separately.

4. Approach

The solution involves counting the frequencies of each character in the string using a frequency counter. We then iterate through the frequency counts of each character. For each character, we add the largest even portion of the frequency to the total palindrome length. If any character has an odd frequency, we mark that we can place one character in the center. Finally, if there is any odd frequency, we add 1 to the length to account for the center character.

5. Edge Cases

Edge cases to consider include strings with only one character, where the longest palindrome is of length 1. Another case is when all characters in the string have odd frequencies; in such cases, we can only use one of those characters at the center, making the longest palindrome of length 1. Additionally, if all characters have even frequencies, the entire string can be rearranged into a palindrome, and the result will be the full length of the string. Strings that consist of all unique characters also result in the longest palindrome being just one character long.

6. Complexity Analysis

The time complexity of the solution is O(n), where n is the length of the string. This is because we only need to traverse the string once to count the frequencies of characters, and then we process each character's count in constant time (since there are only 52 possible characters, uppercase, and lowercase). The space complexity is O(1), as the number of distinct characters is limited to 52, making the space required for storing the frequencies constant.

7. Alternative Approaches

One alternative approach is to use brute force, where we could generate all possible substrings of the string, check if each one is a palindrome, and track the longest one. However, this would be computationally expensive with a time complexity of $O(n^3)$ due to generating substrings and checking each for being a palindrome. Another

approach is dynamic programming, which is used to find the longest palindromic subsequence, but it is not necessary here because the problem focuses on character frequencies rather than the arrangement of characters.

8. Test Cases

Test cases would include strings with varying numbers of characters, including strings where:

- All characters are identical.
- Some characters appear an odd number of times.
- Every character appears an even number of times.
- The string is made up of unique characters. In each case, the expected output should be the length of the longest palindrome that can be formed from the string's characters.

9. Final Thoughts

This solution is optimal for solving the problem of finding the longest palindrome from a string, with a time complexity of O(n) and a space complexity of O(1). It efficiently uses character frequency counting to determine how many characters can contribute to the palindrome's length. The solution handles edge cases and ensures that case sensitivity is respected, making it robust for a wide range of input strings.