3223. Minimum Length of String After Operations

You are given a string `s`.

You can perform the following process on `s` **any** number of times:

- Choose an index `i` in the string 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]`.
- Delete the **closest** character to the **left** of index `i` that is equal to `s[i]`.
- Delete the **closest** character to the **right** of index `i` that is equal to `s[i]`.

Return the **minimum** length of the final string `s` that you can achieve.

Example 1:

Input: s = "abaacbcbb"

Output: 5

Explanation:
We do the following operations:

- Choose index 2, then remove the characters at indices 0 and 3. The resulting string is `s = "bacbcbb"`.
- Choose index 3, then remove the characters at indices 0 and 5. The resulting string is `s = "acbcb"`.

Example 2:

Input: s = "aa"

Output: 2

Explanation:
We cannot perform any operations, so we return the length of the original string.

Constraints:

- `1 <= s.length <= 2 *` 10^5
- `s` consists only of lowercase English letters.

3223. Minimum Length of String After Operations

Overview

We are given a string `s`. The goal is to repeatedly perform the following operation until it is no longer possible:

1. Choose an index `i` such that:
    - There is at least one character equal to `s[i]` to its left.
    - There is at least one character equal to `s[i]` to its right.
2. Once such an index is found, the following characters are removed:
    - The closest matching character to the left of index `i`.
    - The closest matching character to the right of index `i`.

We need to find the smallest possible length of the string after applying this operation repeatedly.

Intuition

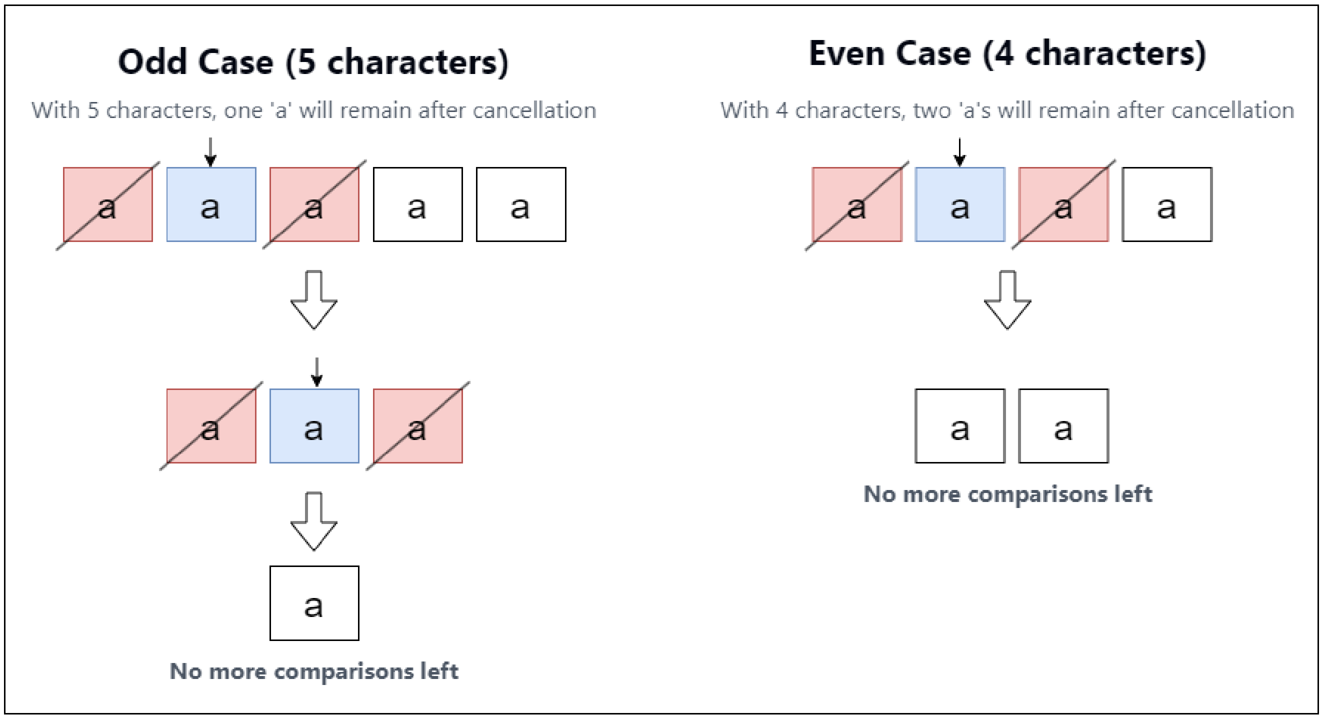
To approach this problem, we need to consider how often each character appears in the string. The goal is to figure out how many characters need to be removed to minimize the string, based on how many times each character occurs:

- If a character appears an odd number of times, we can keep exactly one instance of it, and remove the rest.
- If a character appears an even number of times, we can keep two instances of it—one on the left side and one on the right side, ensuring a valid operation.

For example, let's consider the case where we have 5 `'a'` characters. Since 5 is odd, we'll end up with exactly one `'a'`. We can remove the first and third `'a'` characters because they are closest to the second `'a'`. After that, we are left with three `'a'` characters, and we repeat the process of removing pairs. In the end, only one `'a'` remains. This is because each pair cancels out, leaving the extra character.

Now, let's look at the case with 4 `'a'` characters. Since 4 is even, we first remove the first and third `'a'` characters, which are closest to the second `'a'`. We're left with 2 `'a'` characters, but for comparisons, we need three characters: one as the reference pivot and two indices, one on the left and one on the right, to remove. So, we stop here in the even case.

The entire intuition can be summarized with the help of the image below.



3223. Minimum Length of String After Operations

```
/*
   Counting: Frequency array
   Time compelxity: O(n+26)=O(n)
   Space complexity: O(26)=O(1)
*/

class Solution {
public:
   int minimumLength(std::string s) {
      int freq[26]={0};

      for(auto& c: s) freq[c-'a']++;

      int ans=0;

      for(int i=0;i<26;++i){
         if(freq[i]>0){
            ans+=freq[i]%2?1:2;
         }
      }

      return ans;
   }
};
```