1957. Delete Characters to Make Fancy String

Easy String

Leetcode Link

Problem Description

In this problem, we are dealing with the concept of a "fancy string". A string is considered fancy if no three consecutive characters are identical. The goal is to transform any given string `s` into a fancy string by removing the minimum number of characters necessary.

For example, if the input string is `s = "aaabaaaa"`, a possible fancy string could be `s = "aabaa"` after deleting the right characters. The task is to return the modified string that fulfills the condition of being a fancy string.

Intuition

To approach this problem, we need to scan through the input string and check for any occurrence of three identical consecutive characters. Whenever we find such a sequence, we don't include the third character in our result.

A straightforward way to do this is to iterate over each character in the input string while maintaining a separate list to build the fancy string. For each character, we compare it with the last two characters in the list. If all three are the same, we skip adding the current character to the list; otherwise, we add it.

This process ensures that at no point will there be three identical consecutive characters in our resultant list. In the end, we join the characters in the list to form the final fancy string and return it. This solution has the advantage of examining each character only once, giving us a time-efficient approach to making the string fancy.

Solution Approach

The solution to the problem makes use of a simple iteration pattern over the characters of the string, coupled with a condition check to enforce the "fancy" constraint.

To implement this, we use a Python list, `ans`, which is a dynamic array in this context. We iterate through each character `c` in the input string `s`, and check if this character is about to form a triplet with the two previously added characters (if any).

Here's a step-by-step breakdown of the solution:

1. **Initialization**: An empty list `ans` is created which will store the characters that will eventually make up the fancy string.
2. **Iteration**: We iterate over each character `c` in the given string `s`.
3. **Condition Check**: Before adding the character `c` to the list `ans`, we perform a check:
   - If the length of `ans` is greater than 1 (which means there are at least two characters already in `ans`),
   - And the last character (`ans[-1]`) and the second to last character (`ans[-2]`) in `ans` are both equal to `c`,
   - Then this character `c` would create a sequence of three identical consecutive characters. In this case, we simply skip adding `c` to `ans`.
4. **Character Addition**: If the check fails (meaning adding `c` won't lead to three consecutive identical characters), we add `c` to the list `ans`.
5. **Result Formation**: Once we've finished iterating over the string, we use `''.join(ans)` to concatenate all the elements in the list `ans` into a string, which is the fancy string that we return as the result.

By using a for loop and a conditional statement, we are able to enforce the consecutive character constraint efficiently. The usage of an array (`List` in Python) to build our result allows for easy appending and checking of the last two characters. This approach has a linear runtime complexity because each character in the input string is processed exactly once.

Example Walkthrough

Let's consider a smaller example to illustrate the solution approach using the input string `s = "xxxyyyz"`. Following the solution approach step by step, we will transform `s` into a fancy string.

1. **Initialization**: We create an empty list `ans` which will collect the characters for the fancy string.
2. **Iteration**:
   - We begin by iterating over each character in `s`. The first character is `'x'`.
3. **Condition Check**:
   - Since `ans` is empty, we do not need to check for triplets. There are no previous characters to compare with.
4. **Character Addition**:
   - We add `'x'` to `ans`, resulting in `ans = ['x']`.
5. **Next Iteration**:
   - The next character is `'x'`. We check `ans` and find there is only one `'x'`, so we can safely add another.
   - `ans = ['x', 'x']`.
6. **Next Iteration**:
   - We encounter `'x'` again. We check the last two characters of `ans` and find that adding another `'x'` would create three in a row, which is not allowed for a fancy string.
   - We skip adding this `'x'` and move on to the next character.
7. **Next Iteration**:
   - The next character is `'y'`. There's no issue in adding `'y'` to `ans` as it wouldn't create three identical consecutive characters.
   - `ans = ['x', 'x', 'y']`.
8. **Continued Iteration**:
   - We continue this process for the remaining characters `'y'`,`'y'`, and `'z'`. For both occurrences of `'y'` after the first, we skip adding them since they would create a triplet with the previous two `'y'`s. Finally, `'z'` can be added without any problems.
   - `ans = ['x', 'x', 'y', 'z']`.
9. **Result Formation**:
   - At the end of the iteration, we join the characters in `ans` to form the final string.
   - The fancy string is `'xxyz'`.

In this example, by following the solution approach, we removed two characters `'y'` and one character `'x'` to transform `"xxxyyyz"` into a fancy string `"xxyz"`. This process minimizes the number of deletions needed and ensures that no three consecutive characters in the result are identical.

Python Solution

```python
class Solution:
    def makeFancyString(self, s: str) -> str:
        # Initialize an empty list to store the modified characters of the string
        result = []

        # Iterate over each character in the input string
        for char in s:
            # Check if the last two characters in 'result' are the same as the current character
            if len(result) > 1 and result[-1] == result[-2] == char:
                # If true, skip adding the current character to avoid three consecutive identical characters
                continue
            # Otherwise, append the current character to the result list
            result.append(char)

        # Join all characters in the result list to form the modified string
        return ''.join(result)

```

Java Solution

```java
class Solution {
    // Method to transform the input string into a fancy string
    // A fancy string is a string where no three consecutive characters are equal
    public String makeFancyString(String s) {
        // StringBuilder is initialized to build the fancy string efficiently
        StringBuilder fancyString = new StringBuilder();

        // Iterate over each character in the input string
        for (char currentChar : s.toCharArray()) {
            // Calculate the length of the current fancyString
            int currentLength = fancyString.length();

            // Check if the last two characters in fancyString are the same as the current character
            if (currentLength > 1 &&
                fancyString.charAt(currentLength - 1) == currentChar &&
                fancyString.charAt(currentLength - 2) == currentChar) {
                // If true, continue to the next iteration to avoid adding a third consecutive character
                continue;
            }

            // Append the current character to fancyString
            fancyString.append(currentChar);
        }

        // Convert the StringBuilder to a String and return it
        return fancyString.toString();
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Method that converts a string into a 'fancy string'.
    // A 'fancy string' is a string where no three consecutive characters are equal.
    string makeFancyString(string s) {
        string result; // Initialize an empty string to store the 'fancy string'.

        // Iterate through each character in the input string.
        for (char& currentChar : s) {
            int currentLength = result.size(); // Get the current length of the 'fancy string'.

            // Check if the last two characters in the 'fancy string' are the same as the current character.
            if (currentLength > 1 && result[currentLength - 1] == currentChar && result[currentLength - 2] == currentChar) {
                // If the last two characters are the same as the current character, skip adding this character.
                continue;
            }

            // Append the current character to the 'fancy string' if no three consecutive characters are the same.
            result.push_back(currentChar);
        }

        // Return the 'fancy string'.
        return result;
    }
};

```

Typescript Solution

```typescript
// Function that converts a string into a 'fancy string'.
// A 'fancy string' is a string where no three consecutive characters are equal.
function makeFancyString(s: string): string {
    let result = ''; // Initialize an empty string to store the 'fancy string'.

    // Iterate through each character in the input string.
    for (let i = 0; i < s.length; i++) {
        const currentChar = s[i]; // Get the current character.
        const currentLength = result.length; // Get the current length of the 'fancy string'.

        // Check if the last two characters in the 'fancy string' are the same as the current character.
        if (currentLength > 1 && result[currentLength - 1] === currentChar && result[currentLength - 2] === currentChar) {
            // If the last two characters are the same as the current character, skip adding this character.
            continue;
        }

        // Append the current character to the 'fancy string' if no three consecutive characters are the same.
        result += currentChar;
    }

    // Return the 'fancy string'.
    return result;
}

// Example usage:
// const fancyString = makeFancyString("aaabccc");
// console.log(fancyString); // Output should be "aabcc"

```

Time and Space Complexity

Time Complexity

The given code loops over each character in the input string `s` only once. The condition inside the loop checks the last two characters in the `ans` list to determine if the current character `c` should be added to `ans`.

Since each character in `s` is processed once and each operation within the loop, including comparison and list appending, is O(1), the overall time complexity of the algorithm is O(n), where `n` is the length of the input string `s`.

Space Complexity

The space complexity of the code is mostly determined by the space used to store the `ans` list. In the worst case, when no characters are the same, `ans` will contain all the characters of `s`. Therefore, the space complexity is O(n) as well, where `n` is the length of the input string `s`.