1576. Replace All 's to Avoid Consecutive Repeating Characters

Easy String

Problem Description

The problem asks us to take a string that contains only lowercase English letters and the '?' character. Our goal is to replace every '?' in the string with a lowercase English letter such that no two consecutive characters in the final string are identical. Note that the provided string does not have consecutive repeating characters other than '?'. The challenge is to do this replacement in such a way that we never end up with two of the same characters next to each other. Additionally, we are not allowed to change any characters other than '?'. The requirement is to create a valid string that adheres to these constraints, and we can return any valid answer since there may be multiple solutions.

Intuition

To solve this problem, we need to generate a string with no two identical consecutive characters by replacing '?' with lowercase English letters. The intuition behind the approach is relatively straightforward.

1. We iterate through the string to find the '?' characters that need replacement.
2. For each '?' character found, we attempt to replace it with a character from a set of possible options (in this case, 'a', 'b', or 'c').
3. We choose a replacement character that is different from the characters immediately before and after the current '?' to ensure there are no consecutive repeating characters.
4. This is managed by checking the adjacent characters of each '?' position before deciding which character to assign to it.
5. Since the string is guaranteed not to have consecutive repeating characters apart from '?', and we're using three different characters for replacement, there is always at least one character that will not form a repeating sequence.

Solution Approach

To implement the solution, we convert the input string `s` into a list, making it mutable so we can easily replace '?' characters in place. Then, we go through the following steps:

1. Loop through each character in the list by its index. We only need to take action when we encounter a '?' character.
2. When a '?' is found, we must find a replacement character that doesn't match the character before or after the current position. Since we're only dealing with lowercase English letters, we use a small set of options 'a', 'b', and 'c' to find a suitable replacement. This small subset is sufficient because we're guaranteed there are no consecutive repeating characters except for '?'.
3. For each candidate replacement character `c`, we perform a check:
   - If the '?' is not the first character in the list (i.e., `i > 0`), we need to ensure the previous character `s[i - 1]` is not the same as our candidate `c`.
   - If the '?' is not the last character in the list (i.e., `i + 1 < n`), we need to make sure the next character `s[i + 1]` is not the same as our candidate `c`.
4. If both conditions are satisfied (meaning `c` does not match neighboring characters), we set `s[i]` to `c` and break out of the inner loop, as our replacement is done for this position.
5. After handling all '?' characters, we join the list back into a string with `"".join(s)` and return the result.

The key data structure used in the solution is a list, which allows us to replace characters in the string easily. The algorithm iterates through the characters of the string once, making the time complexity O(n), where n is the length of the string. We employ a simple replacement strategy that checks adjacent characters to ensure we meet the problem's conditions.

Here is the Python code for the implementation:

```
class Solution:
    def modifyString(self, s: str) -> str:
        s = list(s)
        n = len(s)
        for i in range(n):
            if s[i] == "?":
                for c in "abc":
                    if (i and s[i - 1] == c) or (i + 1 < n and s[i + 1] == c):
                        continue
                    s[i] = c
                    break
        return "".join(s)
```

This solution efficiently ensures that the resulting string will have no consecutive repeating characters, utilizing minimal checks and avoiding unnecessary complexity.

Example Walkthrough

Let's walk through a small example to illustrate the solution approach using the given Python code.

Suppose we have the input string `s = "ab?ac?c"`. We are tasked to replace the '?' characters such that no two consecutive characters are identical.

Following the steps of the solution:

1. We first convert `s` to a list: `['a', 'b', '?', 'a', 'c', '?']`.
2. Then we loop through each character:
   - For the first two characters 'a' and 'b', no action is taken since they are not '?'.
   - At index 2 and 3, we find two '?' characters that need to be replaced.
3. For the first '?', at index 2, we need to choose a character that is not 'b' (preceding character) and not 'a' (the next character which is currently '?', but we assume it might turn into 'a' since 'a' is the character following the next '?'). The candidate replacement character can be 'c'.
4. We now replace the first '?', and our string list becomes `['a', 'b', 'c', '?', 'a', 'c', '?']`.
5. For the second '?', at index 3, we need a character that is not 'c' (preceding character) and not 'a' (following character). The candidate replacement character can be 'b'.
6. We replace the second '?', and our list now looks like `['a', 'b', 'c', 'b', 'a', 'c', '?']`.
7. At the last '?', index 6, we choose a character that is not 'c' (preceding character). We can pick either 'a' or 'b' (since there is no following character), let's pick 'a'.
8. After replacing the last '?', our final list is `['a', 'b', 'c', 'b', 'a', 'c', 'a']`.
9. We join this list to form the final string, which is `"abcba" + "ca"` = `"abcbaca"`.

The output string is valid as there are no two consecutive, identical characters. This example demonstrates one of the many possible valid strings that can be formed following the outlined approach.

Solution Implementation

Python

```
class Solution:
    def modifyString(self, s: str) -> str:
        # Convert the input string into a list to modify characters
        char_list = list(s)
        n = len(char_list)

        # Loop through each character in the list
        for i in range(n):
            # Check for '?' placeholders needing replacement
            if char_list[i] == "?":
                # Try replacing with characters 'a', 'b', or 'c'
                for c in "abc":
                    # Ensure the replacement doesn't match neighboring characters
                    if (i > 0 and char_list[i - 1] == c) or (i + 1 < n and char_list[i + 1] == c):
                        continue
                    # Once a valid character is found, replace '?' and move to the next character
                    char_list[i] = c
                    break

        # Join the list back into a string and return
        return "".join(char_list)
```

Java

```
class Solution {
    public String modifyString(String s) {
        // Convert the string to a char array to modify characters in place.
        char[] charArray = s.toCharArray();
        int length = charArray.length;

        // Iterate over each character in the char array.
        for (int i = 0; i < length; ++i) {
            // Check if the current character is a question mark.
            if (charArray[i] == '?') {
                for (char c = 'a'; c <= 'c'; ++c) {
                    // Check if the same character is present on the left or right.
                    // Make sure not to go out of bounds by checking the index.
                    if ((i > 0 && charArray[i - 1] == c) || (i + 1 < length && charArray[i + 1] == c)) {
                        // If the same character is on either side, continue to the next character.
                        continue;
                    }
                    // Assign the character that doesn't match its neighbors.
                    charArray[i] = c;
                    // Once we have found a suitable character, break the inner loop.
                    break;
                }
            }
        }
        // Convert the char array back to a string and return it.
        return String.valueOf(charArray);
    }
}
```

C++

```
class Solution {
public:
    // Function to modify the string by replacing '?' characters
    string modifyString(string s) {
        int n = s.size(); // Get the size of the string

        // Iterate over the characters of the string
        for (int i = 0; i < n; ++i) {
            // Check if the current character is '?'
            if (s[i] == '?') {
                // Iterate over the choices of 'a', 'b', and 'c'
                for (char c = 'a'; c <= 'c'; ++c) {
                    // Check if choosing 'c' would violate the requirement (no same adjacent characters)
                    if ((i > 0 && s[i - 1] == c) || (i + 1 < n && s[i + 1] == c)) {
                        continue; // Skip this letter as it matches an adjacent character
                    }
                    // Found a valid replacement for '?', so set it and break out of the inner loop
                    s[i] = c;
                    break;
                }
            }
        }
        // Return the modified string
        return s;
    }
};
```

TypeScript

```
function modifyString(s: string): string {
    // Create an array of characters from the input string.
    const characters = s.split('');
    // Determine the length of the string.
    const length = s.length;

    // Iterate through each character in the array.
    for (let i = 0; i < length; ++i) {
        // Check if the current character is a question mark.
        if (characters[i] === '?') {
            // Loop through the letters 'a', 'b', and 'c'.
            for (const letter of 'abc') {
                // If the previous character or the next character is the same as `letter`, skip to the next letter.
                if (i > 0 && characters[i - 1] === letter) || (i + 1 < length && characters[i + 1] === letter)) {
                    continue;
                }
                // Replace the question mark with the current `letter` and exit the inner loop.
                characters[i] = letter;
                break;
            }
        }
    }
    // Join the array of characters back into a string and return it.
    return characters.join('');
}
```

```
class Solution:
    def modifyString(self, s: str) -> str:
        # Convert the input string into a list to modify characters
        char_list = list(s)
        n = len(char_list)

        # Loop through each character in the list
        for i in range(n):
            # Check for '?' placeholders needing replacement
            if char_list[i] == "?":
                # Try replacing with characters 'a', 'b', or 'c'
                for c in "abc":
                    # Ensure the replacement doesn't match neighboring characters
                    if (i > 0 and char_list[i - 1] == c) or (i + 1 < n and char_list[i + 1] == c):
                        continue
                    # Once a valid character is found, replace '?' and move to the next character
                    char_list[i] = c
                    break

        # Join the list back into a string and return
        return "".join(char_list)
```

Time and Space Complexity

Time Complexity

The given code loops through each character of the input string `s` exactly once, with a fixed number of operations per loop iteration (checking and possibly replacing a character). For each character that is `"?"`, it attempts a maximum of three substitution checks – each check involves comparing against the previous and the next character in the string (if any).

Since the checking and substitution are constant-time operations and independent of the size of the string, we can determine that the inner loop has a constant time complexity of `O(1)` for each character. The outer loop runs `n` times, where `n` is the length of `s`.

Therefore, the time complexity is `O(n)`, where `n` is the length of the string, because we perform a constant amount of work for each character in the string.

Space Complexity

The space complexity of the code is also dependent on the length of the input string `s`. This is because the string is converted to a list of characters to allow in-place modifications, which takes `O(n)` space, where `n` is the length of the input string. Apart from this, only a constant amount of additional space is used for variables `i`, `c`, and `n`.

Hence, the space complexity is `O(n)`.