984. String Without AAA or BBB

Medium Greedy String

Leetcode Link

Problem Description

Given two integers `a` and `b`, we are asked to construct a string `s` of length `a + b` which contains exactly `a` 'a' letters and `b` 'b' letters. However, the string `s` must not contain the substrings 'aaa' or 'bbb', which means we are not allowed to have three consecutive 'a's or 'b's. The task is to return any such string that satisfies all the conditions.

Intuition

The core idea behind the solution is to create the string by appending letters in a pattern that prevents three consecutive identical characters from being formed. This is accomplished by deciding on the order of 'a's and 'b's to be added to the string based on their counts. If we have more 'a's than 'b's, it means we should avoid adding too many 'a's in a row to prevent forming 'aaa'. Similarly, if there are more 'b's, we should avoid adding too many 'b's in a row.

To achieve this, we use a greedy approach, where we sequentially build the final string by appending the characters in small blocks in the required sequence:

- When `a > b`, we insert a block 'aab'. This uses up more 'a's than 'b's but keeps a safe distance from the forbidden 'aaa' sequence.
- When `b > a`, we do the opposite, inserting a block of 'bba', using up more 'b's while avoiding the forbidden 'bbb'.
- When `a == b`, this means we have an equal number of 'a' and 'b' left, and to maintain the balance we add 'ab' (or 'ba', both are valid) as it does not further skew the balance between 'a' and 'b'.

Once one of the counts reaches zero and the other still has characters left, we can safely append the remaining characters one by one. Since the other character's count is at zero, there is no risk of forming a forbidden sequence. The remaining 'a' or 'b' characters will either be appended at the end of the string as a series of unbroken 'a's or 'b's (but not more than two), or distributed through the already appended blocks, ensuring no forbidden sequences can form.

The join operation is used at the end to convert the list of strings into a single string as the output. Each block appended to the list ensures that the conditions are maintained, and the join simply merges them into the required output format.

Solution Approach

The solution to the string problem uses a greedy algorithm, which is a method for making a sequence of choices in a local optimum manner with the hope that this will lead to a global optimum. The algorithm closely follows the intuition to alternate characters in a way that avoids having three consecutive 'a's or 'b's. It uses a loop to construct the string piece by piece, appending "blocks" of characters based on the count of 'a's and 'b's remaining.

The data structure used in the implementation is primarily a list, which in Python can be efficiently appended to, and it also provides the ability to later convert it to a string with the `join()` method. The algorithm works as follows:

1. Initialize an empty list `ans` that will contain blocks of the final string.
2. Enter a `while` loop that continues as long as both `a` and `b` have remaining characters (i.e., `a > 0` and `b > 0`).
3. Within the while loop, the following checks are made:
   - If `a > b`, it means there are more 'a's to be placed than 'b's. So, append 'aab' to the list and decrement `a` by 2 and `b` by 1 to reflect the characters used.
   - If `a < b`, there are more 'b's left, hence append 'bba' to the list, and decrement `a` by 1 and `b` by 2.
   - If `a == b`, there is a balance, so append 'ab' to the list, and decrement both `a` and `b` by 1.
4. Once the while loop exits, it will either be because `a` or `b` is 0. If there are any 'a's left (i.e., `a` is not 0), append the remaining 'a's as a block to the list. Since 'b' is 0, there's no risk of forming 'aaa'.
5. Similarly, if there are any 'b's left, append them as a single block (`'b' * b`). There will be no 'a' characters left to pair with, preventing 'bbb' from forming.
6. Finally, the `join()` method is used to convert the list of strings into a single string result.

The algorithm ensures that at each step, no block that could lead to either 'aaa' or 'bbb' being formed is ever added, yielding a string that fits the constraints of the problem. There's no need to backtrack or check the entire string at any point since the construction of the string is done with local decisions that respect the global constraints.

Example Walkthrough

Let's say we have `a = 4` and `b = 3`. Our goal is to construct a string of length `a + b = 7` that contains exactly 4 'a's and 3 'b's without having 'aaa' or 'bbb'.

Now, let's walk through the steps:

1. Initialize the list `ans` as empty. This will hold the pieces of our final string.
2. Enter the while loop since both `a` (4) and `b` (3) are greater than 0.
3. Since `a > b`, we append the block 'aab' to `ans`, yielding `ans = ['aab']`. We then decrement `a` by 2 (now `a = 2`) and `b` by 1 (now `b = 2`).
4. Next iteration of the loop, `a` (2) and `b` (2) are now equal. We add 'ab' to `ans`, resulting in `ans = ['aab', 'ab']`. We decrement both `a` and `b` by 1, so now `a = 1` and `b = 1`.
5. In the next iteration, `a` and `b` are still equal, so we add another `ab`. Now `ans = ['aab', 'ab', 'ab']` and both `a` and `b` are decremented by 1, leaving both `a = 0` and `b = 0`.
6. Since `a` is 0, we don't need to add any more 'a's and since `b` is 0 as well, we don't need to add any more 'b's. The loop ends here.
7. Lastly, we use the `join()` method on `ans` to get the final string. Joining `['aab', 'ab', 'ab']` we get the final string `"aabbabb"`.

The resultant string `"aabbabb"` is of length 7, contains exactly 4 'a's and 3 'b's, and avoids the sequences 'aaa' and 'bbb', hence meeting all the criteria requested.

Python Solution

```python
class Solution:
    def str_without_3a3b(self, a: int, b: int) -> str:
        # Start with an empty list to store the result
        result = []

        # Continue until either 'a' or 'b' runs out
        while a and b:
            # If 'a's are more, add 'aab' to result
            if a > b:
                result.append('aab')
                a -= 2  # Use -= operator for better readability
                b -= 1
            # If 'b's are more, add 'bba' to result
            elif a < b:
                result.append('bba')
                a -= 1
                b -= 2
            # If 'a' and 'b' are equal, add 'ab' to result
            else:
                result.append('ab')
                a -= 1
                b -= 1

        # If 'a's are left, append them to result as they will not violate the condition
        if a:
            result.append('a' * a)  # Multiple 'a's by remaining count

        # If 'b's are left, append them to result as they will not violate the condition
        if b:
            result.append('b' * b)  # Multiple 'b's by remaining count

        # Combine the elements of the list into a single string and return
        return ''.join(result)

```

Java Solution

```java
class Solution {

    // Method to create a string that follows the condition of not having more than two consecutive 'a' or 'b'
    public String strWithout3a3b(int aCount, int bCount) {
        // StringBuilder to construct the result string efficiently
        StringBuilder result = new StringBuilder();

        // Loop until one of aCount or bCount becomes 0
        while (aCount > 0 && bCount > 0) {
            // If there are more 'a's left, append "aab" to the result
            if (aCount > bCount) {
                result.append("aab");
                aCount -= 2; // Used two 'a's
                bCount -= 1; // Used one 'b'
            }
            // If there are more 'b's left, append "bba" to the result
            else if (aCount < bCount) {
                result.append("bba");
                aCount -= 1; // Used one 'a'
                bCount -= 2; // Used two 'b's
            }
            // If the number of 'a' and 'b' is equal, append "ab" to the result
            else {
                result.append("ab");
                aCount -= 1; // Used one 'a'
                bCount -= 1; // Used one 'b'
            }
        }

        // If any 'a's are left, append all remaining 'a's at the end.
        // Repeat 'a' the number of times that are left
        while (aCount > 0) {
            result.append("a");
            aCount--;
        }

        // If any 'b's are left, append all remaining 'b's at the end.
        // Repeat 'b' the number of times that are left
        while (bCount > 0) {
            result.append("b");
            bCount--;
        }

        // Convert the StringBuilder to a string and return
        return result.toString();
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    // Generates a string where "a" and "b" are not repeated more than twice consecutively
    string strWithout3a3b(int countA, int countB) {
        string result;

        // Loop as long as both a and b have at least one remaining
        while (countA > 0 && countB > 0) {
            if (countA > countB) {
                // If we have more 'a's, append "aab"
                result += "aab";
                countA -= 2;
                countB -= 1;
            } else if (countA < countB) {
                // If we have more 'b's, append "bba"
                result += "bba";
                countA -= 1;
                countB -= 2;
            } else {
                // If the counts are equal, append "ab"
                result += "ab";
                --countA;
                --countB;
            }
        }

        // If 'a's are left, append all remaining 'a's
        if (countA > 0) {
            result += string(countA, 'a');
        }
        // If 'b's are left, append all remaining 'b's
        if (countB > 0) {
            result += string(countB, 'b');
        }

        // Return the final string
        return result;
    }
};

```

Typescript Solution

```typescript
// Generates a string where "a" and "b" are not repeated more than twice consecutively
function strWithout3a3b(countA: number, countB: number): string {
    let result: string = '';

    // Loop as long as both a and b have at least one remaining
    while (countA > 0 && countB > 0) {
        if (countA > countB) {
            // If we have more 'a's, append "aab"
            result += 'aab';
            countA -= 2;
            countB -= 1;
        } else if (countA < countB) {
            // If we have more 'b's, append "bba"
            result += 'bba';
            countA -= 1;
            countB -= 2;
        } else {
            // If the counts are equal, append "ab"
            result += 'ab';
            countA--;
            countB--;
        }
    }

    // If 'a's are left, append all remaining 'a's
    if (countA > 0) {
        result += 'a'.repeat(countA);
    }

    // If 'b's are left, append all remaining 'b's
    if (countB > 0) {
        result += 'b'.repeat(countB);
    }

    // Return the final string
    return result;
}

```

Time and Space Complexity

The given Python code generates a string where 'a's and 'b's are not in groups of three consecutively by iterating through the counts of 'a's and 'b's.

Time Complexity

The time complexity of the code primarily depends on the number of iterations of the while-loop, which holds true while both `a` and `b` are positive. Within each iteration, the conditionals will execute in constant time, and either `a` or `b`, or both are reduced by at least one.

In the worst-case scenario, `a` and `b` are equal, and each iteration reduces them by 1. As such, the loop runs a maximum of `max(a, b)` times. After the while-loop, appending the remaining 'a's or 'b's happens in linear time relative to the remaining amount of `a` or `b`. Therefore, the worst-case time complexity is $O(max(a, b))$.

Space Complexity

The space complexity is primarily determined by the output space required for the answer string. As `a` and `b` decrease, elements are continuously appended to the `ans` list. This list can contain at most `a + b` entries (when the final counts of `a` and `b` are appended as single characters). No additional significant space is used, so the space complexity is also $O(a + b)$, equivalent to the length of the generated string.