3335. Total Characters in String After Transformations I

🔍 Problem Summary

Given:

- A string s of lowercase letters.
- An integer t — number of times we transform the string.

🔁 Transformation Rules:

1. If the character is 'z', it becomes the string "ab" → adds 2 characters.
2. Otherwise, a character becomes the next character in the alphabet:
   - 'a' → 'b'
   - 'b' → 'c'
   - ...
   - 'y' → 'z'

Goal:

After t transformations, return the length of the resulting string, modulo $10^9 + 7$.

✅ Key Observations

- You don't need to build the actual string because its length can grow exponentially.
- Just track the count of each character using a frequency array.
- Update this frequency array after each transformation.

🧠 Code Explanation

cpp

CopyEdit

```
const int mod = 1e9 + 7;


int lengthAfterTransformations(string s, int t) {
   vector<long long> freq(26, 0);  // freq[i] = count of character 'a' + i


   // Step 1: Count initial frequency of each character in s
   for (char ch : s) {
      freq[ch - 'a']++;
```

```
    }

    // Step 2: Perform t transformations
    for (int step = 0; step < t; ++step) {
        vector<long long> next(26, 0);  // temporary array for next frequencies

        // Rule: 'a' to 'y' becomes the next character
        for (int i = 0; i < 25; ++i) {
            next[i + 1] = (next[i + 1] + freq[i]) % mod;
        }

        // Rule: 'z' becomes "ab"
        next[0] = (next[0] + freq[25]) % mod;  // add to 'a'
        next[1] = (next[1] + freq[25]) % mod;  // add to 'b'

        freq = next;  // move to the next state
    }

    // Step 3: Sum up all frequencies to get final length
    long long ans = 0;
    for (int i = 0; i < 26; ++i) {
        ans = (ans + freq[i]) % mod;
    }

    return (int)ans;
}
```

🧪 Example Dry Run

Input:

s = "abcyy", t = 2

Step 0 (Initial counts):

- **a: 1, b: 1, c: 1, y: 2**

Step 1:

- **a → b**
- **b → c**
- **c → d**
- **y → z (twice)**

Now:
b:1, c:1, d:1, z:2

Step 2:

- **b → c**
- **c → d**
- **d → e**
- **z → ab (twice) → +2 a, +2 b**

Now:
a:2, b:2, c:1, d:1, e:1

Final string = "cdeabab" → length = 7

✅ Output: 7

✅ Time Complexity:

- **Each transformation is O(26) (constant time).**
- **So for t transformations: O(t)**
- **Efficient even for t = 1e5.**

🎯 Summary:

- **Track only frequencies of characters, not actual strings.**
- **Use the transformation rules to update frequencies.**
- **Sum the frequencies after t steps to get the final length.**