809. Expressive Words

Medium Array Two Pointers String Leetcode Link

Problem Description

The problem deals with determining if certain query words can be transformed into a specific target string by repeating characters within those words. This transformation follows a specific rule: a sequence of characters in the query word can be extended only if it results in a group of three or more of the same character in the target string. For example, if the target string is "heeellooo", the query word "hello" can be considered "stretchy" because it can be turned into the target string by repeating the 'e's and the 'o's to get "heeellooo".

To be more specific, if we have a group of characters in our target string, we can only extend the corresponding characters in our query word if the sequence length in the query word is either less than the target sequence and the target sequence has three or more characters, or it is exactly the same as that in the target sequence. So in our example, "hello" can be extended to "heeellooo" because the group "oo" is three or more characters, and the group "e" can also be extended because the original "e" in "hello" is less than the "ee" in "heeellooo" while "heeellooo" contains three or more "e"s.

The challenge is to determine how many words in a given list can be transformed in such a way into a given target string.

Intuition

The intuition behind solving this problem is to use two pointers: one for iterating over the characters of the target string `s` and the other for iterating over the characters of a query word `t`. The goal is to match groups of the same characters in `s` and `t` and validate whether the group in `t` can be stretched to match the group in `s` following the transformation rules defined by the problem.

Here's the step-by-step intuition for the approach:

- Iterate through each query word in the provided list and compare it against the target string.
- Use two pointers approach (let's call them `i` for the target string `s` and `j` for the query string `t`).
- Compare characters at `i` and `j`. If they are different, the word can't be stretched to match the string; hence it's not a "stretchy" word.
- If the characters match, find the length of the consecutive group of the same character in both `s` and `t` — count how many times the character is repeated consecutively.
- If the group in the target string `s` is at least three characters long or equal to the length of the group in `t`, and the group in `t` is not longer than that in `s`, the characters could be expanded to match each other.
- If a character in `s` is repeated less than three times and it's not the same count as in `t`, then it cannot be stretched to match, hence it's not a stretchy word.
- Continue this process until the end of both the target string and the query word are reached.
- If the end of both strings is reached simultaneously, that means the word is stretchy; otherwise, it isn't.
- Count and return the number of words that can be stretched to match the target string.

Solution Approach

The code provided implements the stretchy words problem using a straightforward approach that matches the intuition previously detailed. Here's how the implementation works:

1. Define a helper function `check(s, t)` that takes the target string `s` and a query string `t` and returns `True` if `t` can be stretched to `s`, or `False` otherwise. This function implements the two-pointer approach.

2. Inside `check(s, t)`, initialize two pointers, `i` for the target string `s` and `j` for the query string `t`.

3. Loop through both strings while `i < len(s)` and `j < len(t)`. For each character at `s[i]` and `t[j]`, do the following:

   - If `s[i]` and `t[j]` are different, return `False` as the current character cannot be stretched to match.

   - If they match, find the length of the group of the same character in both strings by incrementing `i` and `j` until the characters change. Let's call the lengths `c1` for `s` and `c2` for `t`.

   - Compare the lengths of these groups. If `c1 < c2`, the solution is immediately `False` because we can't shorten characters in the target string `s`. If the length `c1` is 3 or greater, there's no issue, but if `c1` is less than 3 and also not equal to `c2`, we can't stretch `t` to match `s`; thus, the solution is also `False`.

4. Once the while loop is terminated, the final check is `return i == len(s) and j == len(t)`, which ensures that both strings have been fully traversed and that they match up according to the problem rules.

5. The main function `expressiveWords(s, words)` then applies this `check(s, t)` function to each query string `t` in the list `words`.

6. Use a list comprehension to apply the `check` function to each word in `words`, aggregating the number of `True` results, which corresponds to the count of stretchy words.

7. Finally, return the sum as the answer to how many query strings are stretchy according to the target string `s`.

This solution employs the two pointers technique to facilitate the comparison between characters and groups of characters within the strings. It does not use any complex data structures, focusing instead on efficient iteration and comparison logic to solve the problem.

Example Walkthrough

Let's take the target string `s = "heeellooo"` and the query words list `words = ["hello", "hi", "helo"]` to illustrate the solution approach.

For the query word `hello`:

- Pointers start at the beginning. `i = 0` for `"heeellooo"` and `j = 0` for `"hello"`.
- The first characters `h` match, but we need to check the following characters 'e'.
- In `s`, it's followed by `"ee"` making a group of 3 'e's before the next character 'l'. In `t`, it's only one 'e'.
- Since the group of 'e's in `s` is 3 or more, the single 'e' in `t` can be stretched. So we move `i` to the next group in `s` and `j` to the next character in `t`, both at 'l'.
- The 'l's match, and they are in the correct number so we continue.
- Then we check the 'o's. In `s`, we have a group of `"ooo"`, and in `t`, we only have one 'o'.
- The 'o' in `t` can be stretched because the group in `s` has three or more 'o's.
- We've reached the end of both strings simultaneously. So "hello" is a stretchy word.

For `hi`:

- Characters 'h' match but then `i` points to character 'e' in `s` and `j` to character 'i' in `t`.
- The characters are different and hence `hi` cannot be transformed into `s`. So, "hi" is not a stretchy word.

For `helo`:

- Characters 'h' match. Then we have one 'e' in both `s` and `t`.
- We meet the condition of having less than 3 'e's in `s`, but since both have the same count, we can move on.
- Next, we have one 'l' in both `s` and `t`, so we move on.
- At the 'o's, `s` has a group of 3 'o's while `t` has only one.
- Since the group of 'o's in `s` is 3 or more, we can stretch the 'o' in `t`.
- We've reached the end of `t` but not `s`. We have extra characters in `s` (`"oo"`), so "helo" is not stretchy.

Following the implementation steps, we would use the helper function `check(s, t)` for each word in `words`, and only "hello" would return `True`. Thus, the number of stretchy words in this example is 1.

Python Solution

```python
from typing import List

class Solution:
    def expressiveWords(self, original: str, words: List[str]) -> int:
        # Function to check if a target word can be stretched to match the original word
        def can_stretch(original: str, target: str) -> bool:
            len_original, len_target = len(original), len(target)
            # If the target word is longer than the original, it can't be stretched to match it
            if len_target > len_original:
                return False

            index_original = index_target = 0

            # Iterate over the characters of both the original and target words
            while index_original < len_original and index_target < len_target:
                if original[index_original] != target[index_target]:
                    return False  # Characters do not match and can't be stretched

                # Count the consecutive occurrences of the current character in the original word
                count_original_char = index_original
                while count_original_char < len_original and original[count_original_char] == original[index_original]:
                    count_original_char += 1
                stretch_len = count_original_char - index_original

                # Move to the corresponding character in the target word
                index_original, count_target_char = count_original_char, index_target
                while count_target_char < len_target and target[count_target_char] == target[index_target]:
                    count_target_char += 1
                target_len = count_target_char - index_target
                index_target = count_target_char

                # Check the stretchability condition
                # If stretch_len < 3, then the occurrences must match exactly
                # If stretch_len >= 3, target_len should be at least 1 and not more than stretch_len
                if (stretch_len < target_len or (stretch_len < 3 and stretch_len != target_len)):
                    return False

            # Ensure we've reached the end of both words
            return index_original == len_original and index_target == len_target

        # Use a generator expression with sum to count how many words can be stretched
        return sum(can_stretch(original, target) for target in words)


# Example usage:
solution = Solution()
stretchy = solution.expressiveWords("heeellooo", ["hello", "hi", "helo"])
print(stretchy)  # This would print the number of stretchy words that match "heeellooo"

```

Java Solution

```java
class Solution {
    public int expressiveWords(String s, String[] words) {
        int expressiveWordCount = 0; // Initialize counter for expressive words

        // Check each word in the array
        for (String word : words) {
            if (isExpressive(s, word)) {
                expressiveWordCount++; // Increment count if word is expressive
            }
        }
        return expressiveWordCount; // Return the total count
    }

    // Helper method to check if a given word is expressive
    private boolean isExpressive(String s, String word) {
        int sLength = s.length(), wordLength = word.length();

        // If the length of word is greater than s, it cannot be expressive
        if (wordLength > sLength) {
            return false;
        }

        int i = 0, j = 0; // Pointers for iterating over the characters of s and word
        while (i < sLength && j < wordLength) {
            // Compare characters of s and word at the current pointers
            if (s.charAt(i) != word.charAt(j)) {
                return false; // If they don't match, the word is not expressive
            }
            // Count the number of consecutive identical chars from current pointer in s
            int runLengthS = i;
            while (runLengthS < sLength && s.charAt(runLengthS) == s.charAt(i)) {
                runLengthS++;
            }
            int sNum = runLengthS - i; // The count of consecutive characters in s
            i = runLengthS;

            // Similarly, count consecutive identical chars from current pointer in word
            int runLengthWord = j;
            while (runLengthWord < wordLength && word.charAt(runLengthWord) == word.charAt(j)) {
                runLengthWord++;
            }
            int wordNum = runLengthWord - j; // The count of consecutive characters in word
            j = runLengthWord;

            // If s has fewer chars in the run or the run is less than 3 and different, return false
            if (sNum < wordNum || (sNum < 3 && sNum != wordNum)) {
                return false;
            }
        }
        // Ensure both strings have been fully traversed
        return i == sLength && j == wordLength;
    }
}

```

C++ Solution

```cpp
class Solution {
public:
    /**
     * Counts how many words can be stretched to match the string S.
     *
     * @param S The original string.
     * @param words A list of words to compare to S.
     * @return The count of expressive words.
     */
    int expressiveWords(string S, vector<string>& words) {
        // Lambda function to check if t can be stretched to match s.
        auto isStretchy = [](const string& s, const string& t) -> bool {
            int sLength = s.size(), tLength = t.size();

            // If t is longer than s, we cannot stretch t to match s.
            if (tLength > sLength) return false;

            int i = 0, j = 0; // Pointers for s and t.

            // Iterate over both strings.
            while (i < sLength && j < tLength) {
                // If the characters at the current pointers don't match, return false.
                if (s[i] != t[j]) return false;

                // Count the number of consecutive characters in s.
                int runStartS = i;
                while (i < sLength && s[i] == s[runStartS]) ++i;
                int countS = i - runStartS;

                // Count the number of consecutive characters in t.
                int runStartT = j;
                while (j < tLength && t[j] == t[runStartT]) ++j;
                int countT = j - runStartT;

                // Check if the run in s can be stretched to match t's run or not.
                // If s's run is shorter than t's, or if s's run is not stretchy (countS < 3) and not equal to t's run, return fals
                if (countS < countT || (countS < 3 && countS != countT)) return false;
            }

            // If we've reached the end of both s and t, return true, otherwise return false.
            return i == sLength && j == tLength;
        };

        // Count the number of words that are stretchy.
        int expressiveCount = 0;
        for (const string& word : words) {
            if (isStretchy(S, word)) {
                expressiveCount++;
            }
        }

        return expressiveCount;
    }
};

```

Typescript Solution

```typescript
/**
 * Counts how many words can be stretched to match the string originalString.
 *
 * @param originalString The original string.
 * @param words A list of words to compare to originalString.
 * @return The count of expressive words.
 */
function expressiveWords(originalString: string, words: string[]): number {
    // Function to check if target can be stretched to match source string.
    let isStretchy = (source: string, target: string): boolean => {
        let sourceLength = source.length, targetLength = target.length;

        // If target is longer than source, it cannot be stretched to match source.
        if (targetLength > sourceLength) return false;

        let i = 0, j = 0; // Pointers for source and target strings.

        // Iterate over both strings
        while (i < sourceLength && j < targetLength) {
            // If characters at current pointers don't match, return false.
            if (source[i] !== target[j]) return false;

            // Count the number of consecutive characters in source.
            let runStartSource = i;
            while (i < sourceLength && source[i] === source[runStartSource]) i++;
            let countSource = i - runStartSource;

            // Count the number of consecutive characters in target.
            let runStartTarget = j;
            while (j < targetLength && target[j] === target[runStartTarget]) j++;
            let countTarget = j - runStartTarget;

            // Check if the run in source can be stretched to match target's run.
            // If source's run is shorter than target's, or if source's run is not stretchable (countSource < 3) and not equal to t
            if (countSource < countTarget || (countSource < 3 && countSource !== countTarget)) return false;
        }

        // If we've reached the end of both source and target, return true.
        return i === sourceLength && j === targetLength;
    };

    // Count the number of words that can be stretched.
    let expressiveCount = 0;
    for (const word of words) {
        if (isStretchy(originalString, word)) {
            expressiveCount++;
        }
    }

    return expressiveCount;
}

// Example usage:
const originalString = "heeellooo";
const words = ["hello", "hi", "helo"];
console.log(expressiveWords(originalString, words)); // Output should be the count of expressive words that match 'originalString'

```

Time and Space Complexity

The time complexity of the `expressiveWords` function depends mainly on two parameters: the length of the string `s` and the total number of characters in all the words in the `words` list. Let's denote the length of string `s` as `M` and the total number of characters in the `words` list as `N`.

The `check` function, which is called for each word in `words`, runs with two pointers, `i` and `j`, iterating over the characters of `s` and the current word, respectively. In the worst case, both pointers have to go through the entire length of `s` and the word. The inner while loops increase the pointer `k` to skip over repeated characters and compute the counts `c1` and `c2`, but they do not add to the asymptotic complexity since they simply advance the corresponding pointer `i` or `j`.

Therefore, for a single call, `check` has a time complexity of `O(M)`, where `M` is the length of the string `s`. Since `check` is called once for each word, with the total length being `N`, the overall time complexity for all calls to `check` is `O(N * M)`.

The space complexity of the algorithm is `O(1)` since only a fixed number of variables are used and they do not depend on the size of the inputs. The space required to hold pointers, counts, and other variables is constant and does not grow with the size of the input.