Expressive Words

Sometimes people repeat letters to represent extra feeling, such as "hello" -> "heeellooo", "hi" -> "hiiii". Here, we have groups, of adjacent letters that are all the same character, and adjacent characters to the group are different. A group is extended if that group is length 3 or more, so "e" and "o" would be extended in the first example, and "i" would be extended in the second example. As another example, the groups of "abbcccaaaa" would be "a", "bb", "ccc", and "aaaa"; and "ccc" and "aaaa" are the extended groups of that string.

For some given string S, a query word is *stretchy* if it can be made to be equal to S by extending some groups. Formally, we are allowed to repeatedly choose a group (as defined above) of characters `c`, and add some number of the same character `c` to it so that the length of the group is 3 or more. Note that we cannot extend a group of size one like "h" to a group of size two like "hh" - all extensions must leave the group extended - ie., at least 3 characters long.

Given a list of query words, return the number of words that are stretchy.

Example:
Input:
```
S = "heeellooo"
words = ["hello", "hi", "helo"]
```
Output: 1
Explanation:
```
We can extend "e" and "o" in the word "hello" to get "heeellooo".
We can't extend "helo" to get "heeellooo" because the group "ll" is not extended.
```

Notes:

- `0 <= len(S) <= 100`.
- `0 <= len(words) <= 100`.
- `0 <= len(words[i]) <= 100`.
- `S` and all words in `words` consist only of lowercase letters

Solution 1

Idea from @mzchen

```
    public int expressiveWords(String S, String[] words) {
        int count = 0;
        for (String w : words) {
            int i, j; // S & w's pointers.
            for (i = 0, j = 0; i < S.length(); ++i) {
                if (j < w.length() && S.charAt(i) == w.charAt(j)) { // matches, w p
ointer j 1 step forward to move together with i.
                    ++j;
                }else if (i > 0 && S.charAt(i - 1) == S.charAt(i) && i + 1 < S.leng
th() && S.charAt(i) == S.charAt(i + 1)) { // previous, current & next are same.
                    ++i; // S pointer 1 step forward, attempt to traverse the repea
ted chars.
                }else if (!(i > 1 && S.charAt(i) == S.charAt(i - 1) && S.charAt(i)
== S.charAt(i - 2))) { // current & previous 2 are not same.
                    break; // No match.
                }
            }
            if (i == S.length() && j == w.length()) { ++count; } // both pointers re
ach ends.
        }
        return count;
    }
```

written by rock original link here

Solution 2

For each word to be tested, scan from left to right and check if whenever a mismatched character is reached in `S` three identical characters also present.

```
int expressiveWords(string S, vector<string>& words) {
    int res = 0;
    for (auto &w : words)
        if (w.size() <= S.size()) {
            int i, j;
            for (i = 0, j = 0; j < S.size(); j++) {
                if (w[i] == S[j]) // w[i] references to a null char when i reaches w.size()
                    i++;
                else if (j > 0 && S[j] == S[j - 1] && j + 1 < S.size() && S[j] == S[j + 1]) // last, this and next
                    j++;
                else if (!(j > 1 && S[j] == S[j - 1] && S[j] == S[j - 2])) // this and last 2 chars
                    break;
            }
            if (i == w.size() && j == S.size()) // both pointers reach the end
                res++;
        }
    return res;
}
```

written by mzchen original link here

Solution 3

```
    def expressiveWords(self, S, words):
        p = re.compile("((.)\\2*)")
        matches = [(m[1], len(m[0])) for m in re.findall(p, S)]
        res = 0
        for word in words:
            m = [(m[1], len(m[0])) for m in re.findall(p, word)]
            if len(matches) != len(m): break
            in_res = True
            for pair in zip(matches, m):
                (first, first_len), (sec, sec_len) = pair
                if first != sec or (first_len != sec_len and (first_len < 3 or sec_
len > first_len)):
                    in_res = False
                    break
            res += 1 if in_res else 0
        return res
```

written by johnyrufus16 original link here