Problem 809: Expressive Words

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Sometimes people repeat letters to represent extra feeling. For example:

"hello" -> "heeellooo"

"hi" -> "hiiii"

In these strings like

"heeellooo"

, we have groups of adjacent letters that are all the same:

"h"

,

"eee"

,

"ll"

,

"ooo"

.

You are given a string

s

and an array of query strings

words

. A query word is

stretchy

if it can be made to be equal to

s

by any number of applications of the following extension operation: choose a group consisting of characters

c

, and add some number of characters

c

to the group so that the size of the group is

three or more

.

For example, starting with

"hello"

, we could do an extension on the group

"o"

to get

"hellooo"

, but we cannot get

"helloo"

since the group

"oo"

has a size less than three. Also, we could do another extension like

"ll" -> "llll"

to get

"hellllooo"

. If

s = "hellllooo"

, then the query word

"hello"

would be

stretchy

because of these two extension operations:

query = "hello" -> "hellooo" -> "hellllooo" = s

.

Return

the number of query strings that are

stretchy

.

Example 1:

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 size 3 or more.

Example 2:

Input:

s = "zzzzzyyyyy", words = ["zzyy","zy","zyy"]

Output:

3

Constraints:

1 <= s.length, words.length <= 100

1 <= words[i].length <= 100

s

and

words[i]

consist of lowercase letters.

Code Snippets

C++:

```
class Solution {
public:
int expressiveWords(string s, vector<string>& words) {

}
};
```

Java:

```
class Solution {
public int expressiveWords(String s, String[] words) {

}
}
```

Python3:

```
class Solution:
def expressiveWords(self, s: str, words: List[str]) -> int:
```

Python:

```
class Solution(object):
def expressiveWords(self, s, words):
"""
:type s: str
:type words: List[str]
:rtype: int
```

```
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string[]} words
* @return {number}
*/
var expressiveWords = function(s, words) {

};
```

TypeScript:

```
function expressiveWords(s: string, words: string[]): number {

};
```

C#:

```
public class Solution {
public int ExpressiveWords(string s, string[] words) {

}
}
```

C:

```
int expressiveWords(char* s, char** words, int wordsSize) {

}
```

Go:

```
func expressiveWords(s string, words []string) int {

}
```

Kotlin:

```
class Solution {
fun expressiveWords(s: String, words: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func expressiveWords(_ s: String, _ words: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn expressive_words(s: String, words: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {String[]} words
# @return {Integer}
def expressive_words(s, words)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String[] $words
* @return Integer
*/
function expressiveWords($s, $words) {

}
```

```
}
```

Dart:

```
class Solution {
int expressiveWords(String s, List<String> words) {

}
}
```

Scala:

```
object Solution {
def expressiveWords(s: String, words: Array[String]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec expressive_words(s :: String.t, words :: [String.t]) :: integer
def expressive_words(s, words) do

end
end
```

Erlang:

```
-spec expressive_words(S :: unicode:unicode_binary(), Words ::
[unicode:unicode_binary()]) -> integer().
expressive_words(S, Words) ->
.
```

Racket:

```
(define/contract (expressive-words s words)
(-> string? (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Expressive Words
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int expressiveWords(string s, vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Expressive Words
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int expressiveWords(String s, String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Expressive Words
```

```
Difficulty: Medium
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def expressiveWords(self, s: str, words: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def expressiveWords(self, s, words):
"""
:type s: str
:type words: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Expressive Words
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} s
* @param {string[]} words
* @return {number}
*/
var expressiveWords = function(s, words) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Expressive Words
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function expressiveWords(s: string, words: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Expressive Words
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int ExpressiveWords(string s, string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Expressive Words
```

```
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int expressiveWords(char* s, char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Expressive Words
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func expressiveWords(s string, words []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun expressiveWords(s: String, words: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func expressiveWords(_ s: String, _ words: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Expressive Words
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn expressive_words(s: String, words: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String[]} words
# @return {Integer}
def expressive_words(s, words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String[] $words
* @return Integer
*/
function expressiveWords($s, $words) {

}
}
```

Dart Solution:

```
class Solution {
int expressiveWords(String s, List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def expressiveWords(s: String, words: Array[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec expressive_words(s :: String.t, words :: [String.t]) :: integer
def expressive_words(s, words) do

end
end
```

Erlang Solution:

```
-spec expressive_words(S :: unicode:unicode_binary(), Words ::
[unicode:unicode_binary()]) -> integer().
expressive_words(S, Words) ->
.
```

Racket Solution:

```
(define/contract (expressive-words s words)
(-> string? (listof string?) exact-integer?)
)
```