Problem 1915: Number of Wonderful Substrings

Problem Information

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

Problem Description

A

wonderful

string is a string where

at most one

letter appears an

odd

number of times.

For example,

"ccjjc"

and

"abab"

are wonderful, but

"ab"

is not.

Given a string

word

that consists of the first ten lowercase English letters (

'a'

through

'j'

), return

the

number of wonderful non-empty substrings

in

word

. If the same substring appears multiple times in

word

, then count

each occurrence

separately.

A

substring

is a contiguous sequence of characters in a string.

Example 1:

Input:

word = "aba"

Output:

4

Explanation:

The four wonderful substrings are underlined below: - "

a

ba" -> "a" - "a

b

a" -> "b" - "ab

a

" -> "a" - "

aba

" -> "aba"

Example 2:

Input:

word = "aabb"

Output:

9

Explanation:

The nine wonderful substrings are underlined below: - ''

a

abb" -> "a" - ''

aa

bb" -> "aa" - ''

aab

b" -> "aab" - ''

aabb

'' -> "aabb" - "a

a

bb" -> "a" - "a

abb

'' -> "abb" - "aa

b

b" -> "b" - "aa

bb

'' -> "bb" - "aab

b

" -> "b"

Example 3:

Input:

word = "he"

Output:

2

Explanation:

The two wonderful substrings are underlined below: - "

h

e" -> "h" - "h

e

" -> "e"

Constraints:

1 <= word.length <= 10

5

word

consists of lowercase English letters from

'a'

to

'j'

.

Code Snippets

C++:

```
class Solution {
public:
long long wonderfulSubstrings(string word) {

}
};
```

Java:

```
class Solution {
public long wonderfulSubstrings(String word) {

}
}
```

Python3:

```
class Solution:
def wonderfulSubstrings(self, word: str) -> int:
```

Python:

```
class Solution(object):
def wonderfulSubstrings(self, word):
"""
:type word: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} word
```

```
 * @return {number}
 */
var wonderfulSubstrings = function(word) {

};
```

TypeScript:

```
function wonderfulSubstrings(word: string): number {

};
```

C#:

```
public class Solution {
public long WonderfulSubstrings(string word) {

}
}
```

C:

```
long long wonderfulSubstrings(char * word){

}
```

Go:

```
func wonderfulSubstrings(word string) int64 {

}
```

Kotlin:

```
class Solution {
fun wonderfulSubstrings(word: String): Long {

}
}
```

Swift:

```
class Solution {
func wonderfulSubstrings(_ word: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn wonderful_substrings(word: String) -> i64 {

}
}
```

Ruby:

```
# @param {String} word
# @return {Integer}
def wonderful_substrings(word)

end
```

PHP:

```
class Solution {

/**
* @param String $word
* @return Integer
*/
function wonderfulSubstrings($word) {

}
}
```

Scala:

```
object Solution {
def wonderfulSubstrings(word: String): Long = {

}
}
```

Racket:

```
(define/contract (wonderful-substrings word)
(-> string? exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Wonderful Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
long long wonderfulSubstrings(string word) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Wonderful Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
```

```
public long wonderfulSubstrings(String word) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Wonderful Substrings
Difficulty: Medium
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def wonderfulSubstrings(self, word: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def wonderfulSubstrings(self, word):
"""
:type word: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Wonderful Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

/**
* @param {string} word
* @return {number}
*/
var wonderfulSubstrings = function(word) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Wonderful Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function wonderfulSubstrings(word: string): number {

};
```

C# Solution:

```
/*
* Problem: Number of Wonderful Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public long WonderfulSubstrings(string word) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Number of Wonderful Substrings
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

long long wonderfulSubstrings(char * word){

}
```

Go Solution:

```
// Problem: Number of Wonderful Substrings
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func wonderfulSubstrings(word string) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun wonderfulSubstrings(word: String): Long {

}
}
```

Swift Solution:

```
class Solution {
func wonderfulSubstrings(_ word: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Wonderful Substrings
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn wonderful_substrings(word: String) -> i64 {

}
}
```

Ruby Solution:

```
# @param {String} word
# @return {Integer}
def wonderful_substrings(word)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @return Integer
*/
function wonderfulSubstrings($word) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def wonderfulSubstrings(word: String): Long = {

}
}
```

Racket Solution:

```
(define/contract (wonderful-substrings word)
(-> string? exact-integer?)

)
```