Problem 3330: Find the Original Typed String I

Problem Information

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

Problem Description

Alice is attempting to type a specific string on her computer. However, she tends to be clumsy and

may

press a key for too long, resulting in a character being typed

multiple

times.

Although Alice tried to focus on her typing, she is aware that she may still have done this

at most

once

.

You are given a string

word

, which represents the

final

output displayed on Alice's screen.

Return the total number of

possible

original strings that Alice

might

have intended to type.

Example 1:

Input:

word = "abbcccc"

Output:

5

Explanation:

The possible strings are:

"abbcccc"

,

"abbccc"

,

"abbcc"

,

"abbc"

, and

"abcccc"

.

Example 2:

Input:

word = "abcd"

Output:

1

Explanation:

The only possible string is

"abcd"

.

Example 3:

Input:

word = "aaaa"

Output:

4

Constraints:

1 <= word.length <= 100

word

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int possibleStringCount(string word) {

}
};
```

Java:

```
class Solution {
public int possibleStringCount(String word) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int PossibleStringCount(string word) {

}
}
```

C:

```
int possibleStringCount(char* word) {

}
```

Go:

```
func possibleStringCount(word string) int {

}
```

Kotlin:

```
class Solution {
fun possibleStringCount(word: String): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn possible_string_count(word: String) -> i32 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int possibleStringCount(String word) {

}
}
```

Scala:

```
object Solution {
def possibleStringCount(word: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec possible_string_count(word :: String.t) :: integer
def possible_string_count(word) do

end
end
```

Erlang:

```
-spec possible_string_count(Word :: unicode:unicode_binary()) -> integer().
possible_string_count(Word) ->
.
```

Racket:

```
(define/contract (possible-string-count word)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Original Typed String I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int possibleStringCount(string word) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Original Typed String I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int possibleStringCount(String word) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Original Typed String I
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Original Typed String I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Find the Original Typed String I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

function possibleStringCount(word: string): number {

};
```

C# Solution:

```
/*
* Problem: Find the Original Typed String I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int PossibleStringCount(string word) {

}
}
```

C Solution:

```
/*
* Problem: Find the Original Typed String I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int possibleStringCount(char* word) {

}
```

Go Solution:

```
// Problem: Find the Original Typed String I
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func possibleStringCount(word string) int {

}
```

Kotlin Solution:

```
class Solution {
fun possibleStringCount(word: String): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Original Typed String I
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn possible_string_count(word: String) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int possibleStringCount(String word) {

}
}
```

Scala Solution:

```
object Solution {
def possibleStringCount(word: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec possible_string_count(word :: String.t) :: integer
def possible_string_count(word) do

end
end
```

Erlang Solution:

```
-spec possible_string_count(Word :: unicode:unicode_binary()) -> integer().
possible_string_count(Word) ->
.
```

Racket Solution:

```
(define/contract (possible-string-count word)
(-> string? exact-integer?)
)
```