Problem 3333: Find the Original Typed String II

Problem Information

Difficulty: Hard
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.

You are given a string

word

, which represents the

final

output displayed on Alice's screen. You are also given a

positive

integer

k

.

Return the total number of

possible

original strings that Alice

might

have intended to type, if she was trying to type a string of size

at least

k

.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

word = "aabbccdd", k = 7

Output:

5

Explanation:

The possible strings are:

"aabbccdd"

,

"aabbccd"

,

"aabbcdd"

,

"aabccdd"

, and

"abbccdd"

.

Example 2:

Input:

word = "aabbccdd", k = 8

Output:

1

Explanation:

The only possible string is

"aabbccdd"

.

Example 3:

Input:

word = "aaabbb", k = 3

Output:

8

Constraints:

1 <= word.length <= 5 * 10

5

word

consists only of lowercase English letters.

1 <= k <= 2000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

```
int possibleStringCount(char* word, int k) {

}
```

Go:

```
func possibleStringCount(word string, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find the Original Typed String II
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int possibleStringCount(string word, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Original Typed String II
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int possibleStringCount(String word, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Original Typed String II
Difficulty: Hard
Tags: array, string, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Original Typed String II
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Find the Original Typed String II
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function possibleStringCount(word: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the Original Typed String II
* Difficulty: Hard
```

```
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Find the Original Typed String II
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int possibleStringCount(char* word, int k) {

}
```

Go Solution:

```
// Problem: Find the Original Typed String II
// Difficulty: Hard
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func possibleStringCount(word string, k int) int {
```

```
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Original Typed String II
// Difficulty: Hard
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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