Problem 2953: Count Complete Substrings

Problem Information

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

Problem Description

You are given a string

word

and an integer

k

.

A substring

s

of

word

is

complete

if:

Each character in

s

occurs

exactly

k

times.

The difference between two adjacent characters is

at most

2

. That is, for any two adjacent characters

c1

and

c2

in

s

, the absolute difference in their positions in the alphabet is

at most

2

.

Return

the number of

complete

substrings of

word

.

A

substring

is a

non-empty

contiguous sequence of characters in a string.

Example 1:

Input:

word = "igigee", k = 2

Output:

3

Explanation:

The complete substrings where each character appears exactly twice and the difference between adjacent characters is at most 2 are:

igig

ee, igig

ee

,

igigee

.

Example 2:

Input:

word = "aaabbbccc", k = 3

Output:

6

Explanation:

The complete substrings where each character appears exactly three times and the difference between adjacent characters is at most 2 are:

aaa

bbbccc, aaa

bbb

ccc, aaabbb

ccc

,

aaabbb

ccc, aaa

bbbccc

,

aaabbbccc

.

Constraints:

1 <= word.length <= 10

5

word

consists only of lowercase English letters.

1 <= k <= word.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Complete Substrings
* Difficulty: Hard
* 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:
int countCompleteSubstrings(string word, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Count Complete Substrings
* Difficulty: Hard
* 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 int countCompleteSubstrings(String word, int k) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Count Complete Substrings
Difficulty: Hard
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 countCompleteSubstrings(self, word: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Complete Substrings
* Difficulty: Hard
* 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
* @param {number} k
* @return {number}
*/
var countCompleteSubstrings = function(word, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Complete Substrings
* Difficulty: Hard
* 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 countCompleteSubstrings(word: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Complete Substrings
* Difficulty: Hard
* 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 int CountCompleteSubstrings(string word, int k) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Count Complete Substrings
* Difficulty: Hard
* 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
*/

int countCompleteSubstrings(char* word, int k) {

}
```

Go Solution:

```
// Problem: Count Complete Substrings
// Difficulty: Hard
// 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 countCompleteSubstrings(word string, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Complete Substrings
// Difficulty: Hard
// 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 count_complete_substrings(word: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @param Integer $k
* @return Integer
*/
```

```
function countCompleteSubstrings($word, $k) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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

```
)
```