Problem 3258: Count Substrings That Satisfy K-Constraint I

Problem Information

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

Problem Description

You are given a

binary

string

s

and an integer

k

.

A

binary string

satisfies the

k-constraint

if

either

of the following conditions holds:

The number of

0

's in the string is at most

k

.

The number of

1

's in the string is at most

k

.

Return an integer denoting the number of

substrings

of

s

that satisfy the

k-constraint

.

Example 1:

Input:

s = "10101", k = 1

Output:

12

Explanation:

Every substring of

s

except the substrings

"1010"

,

"10101"

, and

"0101"

satisfies the k-constraint.

Example 2:

Input:

s = "1010101", k = 2

Output:

25

Explanation:

Every substring of

s

except the substrings with a length greater than 5 satisfies the k-constraint.

Example 3:

Input:

s = "11111", k = 1

Output:

15

Explanation:

All substrings of

s

satisfy the k-constraint.

Constraints:

1 <= s.length <= 50

1 <= k <= s.length

s[i]

is either

'0'

or

'1'

.

Code Snippets

C++:

```
class Solution {
public:
int countKConstraintSubstrings(string s, int k) {

}
};
```

Java:

```
class Solution {
public int countKConstraintSubstrings(String s, int k) {

}
}
```

Python3:

```
class Solution:
def countKConstraintSubstrings(self, s: str, k: int) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countKConstraintSubstrings(s: string, k: number): number {

};
```

C#:

```
public class Solution {
public int CountKConstraintSubstrings(string s, int k) {

}
}
```

C:

```
int countKConstraintSubstrings(char* s, int k) {

}
```

Go:

```
func countKConstraintSubstrings(s string, k int) int {

}
```

Kotlin:

```
class Solution {
fun countKConstraintSubstrings(s: String, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func countKConstraintSubstrings(_ s: String, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_k_constraint_substrings(s: String, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def count_k_constraint_substrings(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function countKConstraintSubstrings($s, $k) {

}
}
```

Dart:

```
class Solution {
int countKConstraintSubstrings(String s, int k) {
```

```
}
}
```

Scala:

```
object Solution {
def countKConstraintSubstrings(s: String, k: Int): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec count_k_constraint_substrings(S :: unicode:unicode_binary(), K ::
integer()) -> integer().
count_k_constraint_substrings(S, K) ->
.
```

Racket:

```
(define/contract (count-k-constraint-substrings s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Substrings That Satisfy K-Constraint I
```

```
* Difficulty: Easy
* Tags: array, string, tree
*
* 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 countKConstraintSubstrings(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Count Substrings That Satisfy K-Constraint I
* Difficulty: Easy
* Tags: array, string, tree
*
* 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 countKConstraintSubstrings(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Substrings That Satisfy K-Constraint I
Difficulty: Easy
Tags: array, string, tree

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 countKConstraintSubstrings(self, s: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Substrings That Satisfy K-Constraint I
* Difficulty: Easy
* Tags: array, string, tree
*
* 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} s
* @param {number} k
* @return {number}
*/
var countKConstraintSubstrings = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Substrings That Satisfy K-Constraint I
* Difficulty: Easy
* Tags: array, string, tree
*
* 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 countKConstraintSubstrings(s: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Substrings That Satisfy K-Constraint I
* Difficulty: Easy
* Tags: array, string, tree
*
* 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 CountKConstraintSubstrings(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Count Substrings That Satisfy K-Constraint I
* Difficulty: Easy
* Tags: array, string, tree
*
* 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 countKConstraintSubstrings(char* s, int k) {

}
```

Go Solution:

```
// Problem: Count Substrings That Satisfy K-Constraint I
// Difficulty: Easy
// Tags: array, string, tree
//
// 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 countKConstraintSubstrings(s string, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countKConstraintSubstrings(s: String, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countKConstraintSubstrings(_ s: String, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Substrings That Satisfy K-Constraint I
// Difficulty: Easy
// Tags: array, string, tree
```

```
//
// 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_k_constraint_substrings(s: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def count_k_constraint_substrings(s, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function countKConstraintSubstrings($s, $k) {

}
}
```

Dart Solution:

```
class Solution {
int countKConstraintSubstrings(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def countKConstraintSubstrings(s: String, k: Int): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec count_k_constraint_substrings(S :: unicode:unicode_binary(), K ::
integer()) -> integer().
count_k_constraint_substrings(S, K) ->
.
```

Racket Solution:

```
(define/contract (count-k-constraint-substrings s k)
(-> string? exact-integer? exact-integer?)
)
```