Problem 3261: Count Substrings That Satisfy K-Constraint II

Problem Information

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

Problem Description

You are given a

binary

string

s

and an integer

k

.

You are also given a 2D integer array

queries

, where

queries[i] = [l

i

, r

i

]

.

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 array

answer

, where

answer[i]

is the number of

substrings

of

s[l

i

..r

i

]

that satisfy the

k-constraint

.

Example 1:

Input:

s = "0001111", k = 2, queries = [[0,6]]

Output:

[26]

Explanation:

For the query

[0, 6]

, all substrings of

s[0..6] = "0001111"

satisfy the k-constraint except for the substrings

s[0..5] = "000111"

and

s[0..6] = "0001111"

.

Example 2:

Input:

s = "010101", k = 1, queries = [[0,5],[1,4],[2,3]]

Output:

[15,9,3]

Explanation:

The substrings of

s

with a length greater than 3 do not satisfy the k-constraint.

Constraints:

1 <= s.length <= 10

5

s[i]

is either

'0'

or

'1'

.

1 <= k <= s.length

1 <= queries.length <= 10

5

queries[i] == [l

i

, r

i

]

0 <= l

i

<= r

i

< s.length

All queries are distinct.

Code Snippets

C++:

```
class Solution {
public:
vector<long long> countKConstraintSubstrings(string s, int k,
vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public long[] countKConstraintSubstrings(String s, int k, int[][] queries) {

}
}
```

Python3:

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

Python:

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

```
:type k: int
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long[] CountKConstraintSubstrings(string s, int k, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* countKConstraintSubstrings(char* s, int k, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func countKConstraintSubstrings(s string, k int, queries [][]int) []int64 {

}
```

Kotlin:

```
class Solution {
fun countKConstraintSubstrings(s: String, k: Int, queries: Array<IntArray>):
LongArray {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn count_k_constraint_substrings(s: String, k: i32, queries:
Vec<Vec<i32>>) -> Vec<i64> {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> countKConstraintSubstrings(String s, int k, List<List<int>>
queries) {

}
}
```

Scala:

```
object Solution {
def countKConstraintSubstrings(s: String, k: Int, queries:
Array[Array[Int]]): Array[Long] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Substrings That Satisfy K-Constraint II
* Difficulty: Hard
* Tags: array, string, tree, search
*
* 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:
vector<long long> countKConstraintSubstrings(string s, int k,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Count Substrings That Satisfy K-Constraint II
* Difficulty: Hard
```

```
* Tags: array, string, tree, search
*
* 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[] countKConstraintSubstrings(String s, int k, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Substrings That Satisfy K-Constraint II
Difficulty: Hard
Tags: array, string, tree, search

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, queries:
List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Substrings That Satisfy K-Constraint II
* Difficulty: Hard
* Tags: array, string, tree, search
*
* 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, queries:
number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Count Substrings That Satisfy K-Constraint II
* Difficulty: Hard
* Tags: array, string, tree, search
*
* 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[] CountKConstraintSubstrings(string s, int k, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Count Substrings That Satisfy K-Constraint II
* Difficulty: Hard
* Tags: array, string, tree, search
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* countKConstraintSubstrings(char* s, int k, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Count Substrings That Satisfy K-Constraint II
// Difficulty: Hard
// Tags: array, string, tree, search
//
```

```
// 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, queries [][]int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countKConstraintSubstrings(s: String, k: Int, queries: Array<IntArray>):
LongArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Substrings That Satisfy K-Constraint II
// Difficulty: Hard
// Tags: array, string, tree, search
//
// 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, queries:
Vec<Vec<i32>>) -> Vec<i64> {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<int> countKConstraintSubstrings(String s, int k, List<List<int>>
queries) {

}
}
```

Scala Solution:

```
object Solution {
def countKConstraintSubstrings(s: String, k: Int, queries:
Array[Array[Int]]): Array[Long] = {
```

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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