Problem 696: Count Binary Substrings

Problem Information

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

Problem Description

Given a binary string

s

, return the number of non-empty substrings that have the same number of

0

's and

1

's, and all the

0

's and all the

1

's in these substrings are grouped consecutively.

Substrings that occur multiple times are counted the number of times they occur.

Example 1:

Input:

s = "00110011"

Output:

6

Explanation:

There are 6 substrings that have equal number of consecutive 1's and 0's: "0011", "01", "1100", "10", "0011", and "01". Notice that some of these substrings repeat and are counted the number of times they occur. Also, "00110011" is not a valid substring because all the 0's (and 1's) are not grouped together.

Example 2:

Input:

s = "10101"

Output:

4

Explanation:

There are 4 substrings: "10", "01", "10", "01" that have equal number of consecutive 1's and 0's.

Constraints:

1 <= s.length <= 10

5

s[i]

is either

'0'

or

'1'

.

Code Snippets

C++:

```
class Solution {
public:
int countBinarySubstrings(string s) {

}
};
```

Java:

```
class Solution {
public int countBinarySubstrings(String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countBinarySubstrings(s: string): number {

};
```

C#:

```
public class Solution {
public int CountBinarySubstrings(string s) {

}
}
```

C:

```
int countBinarySubstrings(char* s) {

}
```

Go:

```
func countBinarySubstrings(s string) int {

}
```

Kotlin:

```
class Solution {
fun countBinarySubstrings(s: String): Int {

}
}
```

Swift:

```
class Solution {
func countBinarySubstrings(_ s: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_binary_substrings(s: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @return {Integer}
def count_binary_substrings(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function countBinarySubstrings($s) {

}
}
```

Dart:

```
class Solution {
int countBinarySubstrings(String s) {

}
}
```

Scala:

```
object Solution {
def countBinarySubstrings(s: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_binary_substrings(s :: String.t) :: integer
def count_binary_substrings(s) do

end
end
```

Erlang:

```
-spec count_binary_substrings(S :: unicode:unicode_binary()) -> integer().
count_binary_substrings(S) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Binary Substrings
* 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 countBinarySubstrings(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Count Binary Substrings
* 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 countBinarySubstrings(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Binary Substrings
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 countBinarySubstrings(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Binary Substrings
* 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
* @return {number}
*/
var countBinarySubstrings = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Binary Substrings
* 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 countBinarySubstrings(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Count Binary Substrings
* 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 CountBinarySubstrings(string s) {

}
}
```

C Solution:

```
/*
* Problem: Count Binary Substrings
* 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 countBinarySubstrings(char* s) {
}
```

Go Solution:

```
// Problem: Count Binary Substrings
// 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 countBinarySubstrings(s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun countBinarySubstrings(s: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func countBinarySubstrings(_ s: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Binary Substrings
// 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_binary_substrings(s: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Integer}
def count_binary_substrings(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function countBinarySubstrings($s) {

}
}
```

Dart Solution:

```
class Solution {
int countBinarySubstrings(String s) {

}
}
```

Scala Solution:

```
object Solution {
def countBinarySubstrings(s: String): Int = {



}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_binary_substrings(s :: String.t) :: integer
def count_binary_substrings(s) do

end
end
```

Erlang Solution:

```
-spec count_binary_substrings(S :: unicode:unicode_binary()) -> integer().
count_binary_substrings(S) ->
.
```

Racket Solution:

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