Problem 2489: Number of Substrings With Fixed Ratio

Problem Information

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

Problem Description

You are given a binary string

s

, and two integers

num1

and

num2

.

num1

and

num2

are coprime numbers.

A

ratio substring

is a substring of s where the ratio between the number of

0

's and the number of

1

's in the substring is exactly

num1 : num2

.

For example, if

num1 = 2

and

num2 = 3

, then

"01011"

and

"1110000111"

are ratio substrings, while

"11000"

is not.

Return

the number of

non-empty

ratio substrings of

s

.

Note

that:

A

substring

is a contiguous sequence of characters within a string.

Two values

x

and

y

are

coprime

if

gcd(x, y) == 1

where

gcd(x, y)

is the greatest common divisor of

x

and

y

.

Example 1:

Input:

s = "0110011", num1 = 1, num2 = 2

Output:

4

Explanation:

There exist 4 non-empty ratio substrings. - The substring s[0..2]: "

011

0011". It contains one 0 and two 1's. The ratio is 1 : 2. - The substring s[1..4]: "0

110

011". It contains one 0 and two 1's. The ratio is 1 : 2. - The substring s[4..6]: "0110

011

". It contains one 0 and two 1's. The ratio is 1 : 2. - The substring s[1..6]: "0

110011

". It contains two 0's and four 1's. The ratio is 2 : 4 == 1 : 2. It can be shown that there are no more ratio substrings.

Example 2:

Input:

s = "10101", num1 = 3, num2 = 1

Output:

0

Explanation:

There is no ratio substrings of s. We return 0.

Constraints:

1 <= s.length <= 10

5

1 <= num1, num2 <= s.length

num1

and

num2

are coprime integers.

Code Snippets

C++:

```
class Solution {
public:
long long fixedRatio(string s, int num1, int num2) {

}
};
```

Java:

```
class Solution {
public long fixedRatio(String s, int num1, int num2) {

}
}
```

Python3:

```
class Solution:
def fixedRatio(self, s: str, num1: int, num2: int) -> int:
```

Python:

```
class Solution(object):
def fixedRatio(self, s, num1, num2):
"""
:type s: str
:type num1: int
:type num2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} num1
* @param {number} num2
* @return {number}
*/
var fixedRatio = function(s, num1, num2) {

};
```

TypeScript:

```
function fixedRatio(s: string, num1: number, num2: number): number {

};
```

C#:

```
public class Solution {
public long FixedRatio(string s, int num1, int num2) {

}
}
```

C:

```
long long fixedRatio(char* s, int num1, int num2) {

}
```

Go:

```
func fixedRatio(s string, num1 int, num2 int) int64 {

}
```

Kotlin:

```
class Solution {
fun fixedRatio(s: String, num1: Int, num2: Int): Long {

}
}
```

Swift:

```
class Solution {
func fixedRatio(_ s: String, _ num1: Int, _ num2: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn fixed_ratio(s: String, num1: i32, num2: i32) -> i64 {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} num1
# @param {Integer} num2
# @return {Integer}
def fixed_ratio(s, num1, num2)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $num1
* @param Integer $num2
* @return Integer
*/
function fixedRatio($s, $num1, $num2) {

}
}
```

Dart:

```
class Solution {
int fixedRatio(String s, int num1, int num2) {

}
}
```

Scala:

```
object Solution {
def fixedRatio(s: String, num1: Int, num2: Int): Long = {
}
}
```

Elixir:

```
defmodule Solution do
@spec fixed_ratio(s :: String.t, num1 :: integer, num2 :: integer) :: integer
def fixed_ratio(s, num1, num2) do

end
end
```

Erlang:

```
-spec fixed_ratio(S :: unicode:unicode_binary(), Num1 :: integer(), Num2 ::
integer()) -> integer().
fixed_ratio(S, Num1, Num2) ->
.
```

Racket:

```
(define/contract (fixed-ratio s num1 num2)
(-> string? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Substrings With Fixed Ratio
* Difficulty: Medium
* Tags: array, string, tree, math, 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:
long long fixedRatio(string s, int num1, int num2) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Substrings With Fixed Ratio
* Difficulty: Medium
* Tags: array, string, tree, math, 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 long fixedRatio(String s, int num1, int num2) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Substrings With Fixed Ratio
Difficulty: Medium
Tags: array, string, tree, math, 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 fixedRatio(self, s: str, num1: int, num2: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def fixedRatio(self, s, num1, num2):
"""
:type s: str
:type num1: int
:type num2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Substrings With Fixed Ratio
* Difficulty: Medium
* Tags: array, string, tree, math, 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} s
* @param {number} num1
* @param {number} num2
* @return {number}
*/
var fixedRatio = function(s, num1, num2) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Substrings With Fixed Ratio
* Difficulty: Medium
* Tags: array, string, tree, math, 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 fixedRatio(s: string, num1: number, num2: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Substrings With Fixed Ratio
* Difficulty: Medium
* Tags: array, string, tree, math, 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 long FixedRatio(string s, int num1, int num2) {

}
}
```

C Solution:

```
/*
* Problem: Number of Substrings With Fixed Ratio
* Difficulty: Medium
* Tags: array, string, tree, math, 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
*/

long long fixedRatio(char* s, int num1, int num2) {

}
```

Go Solution:

```
// Problem: Number of Substrings With Fixed Ratio
// Difficulty: Medium
// Tags: array, string, tree, math, 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 fixedRatio(s string, num1 int, num2 int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun fixedRatio(s: String, num1: Int, num2: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func fixedRatio(_ s: String, _ num1: Int, _ num2: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Substrings With Fixed Ratio
// Difficulty: Medium
// Tags: array, string, tree, math, 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 fixed_ratio(s: String, num1: i32, num2: i32) -> i64 {
}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} num1
# @param {Integer} num2
# @return {Integer}
def fixed_ratio(s, num1, num2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $num1
* @param Integer $num2
* @return Integer
*/
function fixedRatio($s, $num1, $num2) {

}
}
```

Dart Solution:

```
class Solution {
int fixedRatio(String s, int num1, int num2) {

}
}
```

Scala Solution:

```
object Solution {
def fixedRatio(s: String, num1: Int, num2: Int): Long = {
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec fixed_ratio(s :: String.t, num1 :: integer, num2 :: integer) :: integer
def fixed_ratio(s, num1, num2) do

end
end
```

Erlang Solution:

```
-spec fixed_ratio(S :: unicode:unicode_binary(), Num1 :: integer(), Num2 ::
integer()) -> integer().
fixed_ratio(S, Num1, Num2) ->
.
```

Racket Solution:

```
(define/contract (fixed-ratio s num1 num2)
(-> string? exact-integer? exact-integer? exact-integer?)
)
```