2489. Number of Substrings With Fixed Ratio

**Prefix Sum** 

Math String

**Problem Description** 

Hash Table

## The problem provides us with a binary string s and two integers num1 and num2 that are coprime (meaning their greatest

Medium

the string) where the ratio of the number of 0 s to the number of 1 s is exactly num1: num2. To illustrate, if num1 = 2 and num2 = 3, a valid ratio substring might be "01011" since there are two 0s and three 1s. The problem requires us to count all such substrings in the given binary string s.

common divisor is 1). We are tasked with finding the number of non-empty substrings (contiguous sequence of characters within

Intuition

The solution leverages the fact that the difference between num1 times the count of 1s and num2 times the count of 0s in a

once ({0: 1}).

Here's how we arrive at the solution step by step: 1. Initialize two counters, no and n1, to count the number of os and 1s encountered in the string as we iterate. 2. Use a counter dictionary, cnt, to keep track of how many times each difference (key) has occurred, starting with a difference of 0 occurring

substring will be the same for all substrings that have the num1 : num2 ratio. To keep track of these differences, a counter is used while iterating through the string.

4. Calculate the current difference x = n1 \* num1 - n0 \* num2. This difference will be the same for all substrings that satisfy the ratio condition. 5. Increment the answer by the count of how many times we've encountered this difference previously, because each occurrence indicates a

3. Iterate through the string, updating no and n1 each time we encounter a o or 1, respectively.

- potential starting point for a valid substring ending at the current position. 6. Update the counter for the current difference, indicating that we have another potential starting point for future substrings. 7. At the end of the string, ans contains the total count of ratio substrings.

making it very efficient.

Solution Approach

For implementing the solution to count the non-empty ratio substrings, a combination of prefix sums, mathematical reasoning,

Initialize two variables no and n1 to count the occurrences of os and 1s, respectively, as we iterate through the binary string

This approach effectively reduces the problem to a single pass through the string with a constant-time check for each character,

and a hash map to efficiently count differences is used. Here is the detailed breakdown of the algorithm:

## Initialize a variable ans to store the count of valid ratio substrings. This will be our final answer. Create a Counter dictionary cnt to keep track of the observed differences. Start with a difference of 0 that has occurred

once. This relates to the empty substring before we start. Iterate through each character c in the string s.

For each character in the string, calculate the difference x which is given by the formula x = n1 \* num1 - n0 \* num2. This will give us a unique value for a valid ratio between num1 : num2 at each position in the string.

from the initialization of cnt.

**Example Walkthrough** 

exactly 2:1.

When the character is '0', increment n0.

When the character is '1', increment n1.

value associated with this difference is the number of times a substring has ended at the current point in the string with a valid ratio.

The value x represents the cumulative difference at any point in the string. Look up this difference in the cnt dictionary. The

Add the value from cnt[x] to ans. If the difference x has not been encountered before, it contributes 0 to ans, as seen

Finally, increment the count for the difference x in cnt. This step records that a new potential starting point for valid ratio

substrings has been found. After the end of the loop, the ans variable holds the total number of valid ratio substrings found in the binary string s.

We use a hash map (Counter) for fast lookups and updates of the differences, which allows the solution to run with a time

complexity of O(n), where n is the length of the binary string. The combination of prefix sums (here, the cumulative counts of 0 s

and 1s) and the hash map to record the frequencies of differences encountered so far is a powerful pattern that enables us to efficiently solve this problem.

Let's illustrate the solution approach with a small example. Suppose we have a binary string s = "010101" and our coprime

numbers num1 = 2 and num2 = 1. This means we want to count substrings where the number of 0s to the number of 1s is

Follow along with the following steps:

3. Create a Counter dict cnt = {0: 1} to keep track of the observed differences, starting with a 0 difference. 4. Start iterating through each character in s. Read first character: c = '0', increment n0 to 1. • Calculate difference: x = n1 \* 2 - n0 \* 1 = 0 \* 2 - 1 \* 1 = -1.  $\circ$  Add cnt[x] to ans: ans = ans + cnt.get(x, 0) = 0 (since -1 is not in cnt, we assume 0).

 $\circ$  Next c = '1'; n1 = 2; x = 2 \* 2 - 2 \* 1 = 2; ans = ans + cnt.get(x, 0) = 2; cnt[2] = cnt.get(2, 0) + 1 = 1.  $\circ$  Next c = '0'; n0 = 3; x = 2 \* 2 - 3 \* 1 = 1; ans = ans + cnt[1] = 2; cnt[1] = cnt[1] + 1 = 2.

Solution Implementation

from collections import Counter

for char in string:

count zeros = count\_ones = 0

count zeros += char == '0'

count\_ones += char == '1'

ans += counter[difference]

counter[difference] += 1

long count0 = 0, count1 = 0;

for (char c : s.toCharArray()) {

count0 += c == '0' ? 1 : 0;

count1 += c == '1' ? 1 : 0;

**Python** 

class Solution:

ans = 0

6. Continue with the rest of the string:

5. Move to the next character and repeat steps.

 $\circ$  Calculate x = 1 \* 2 - 1 \* 1 = 1.

o Update cnt: cnt[1] = cnt.get(1, 0) + 1 = 1.

 $\circ$  c = '1', increment n1 to 1.

• Last c = '1'; n1 = 3; x = 3 \* 2 - 3 \* 1 = 3; ans = ans + cnt.get(x, 0) = 2 (since 3 is not in cnt, we assume 0); cnt[3] = 1cnt.get(3, 0) + 1 = 1.

def fixedRatio(self, string: str, num zeros: int, num ones: int) -> int:

# Increment the count of '0's and '1's based on the current character

difference = count\_ones \* num\_zeros - count\_zeros \* num\_ones

# Accumulate the number of occurrences of the current difference

# Increment the count for the current difference in the counter

// Initialize the answer, which will store the number of valid substrings.

# Initialize counters for '0's and '1's and result variable ans

# Iterate over each character in the input string

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

// Initialize the count of '0's and '1's seen so far.

// Iterate over each character in the input string.

// Return the total count of valid substrings.

// Define a type alias 'll' for 'long long' for convenient usage.

// Increment count0 if the current character is '0'.

// Increment count1 if the current character is '1'.

long currentDifference = count1 \* num1 - count0 \* num2;

answer += countMap.getOrDefault(currentDifference, 0L);

// Update the count of the current difference in the map.

// Determine the current difference based on the fixed ratio.

// Increment answer by the count of this difference seen so far.

countMap.put(currentDifference, countMap.getOrDefault(currentDifference, 0L) + 1);

1. Initialize two variables n0 = 0 and n1 = 0 for counting 0 s and 1 s as we go along.

○ Update cnt with the new difference: cnt[-1] = cnt.get(-1, 0) + 1 = 1.

 $\circ$  ans = ans + cnt.get(x, 0) = 0 (since 1 is not in cnt, we assume 0).

2. Initialize ans = 0 which will hold the final count of valid ratio substrings.

The two valid substrings are "01" from positions 1 to 2 and "0101" from positions 1 to 4. Each time we found the required

• Next c = '0'; n0 = 2; x = 1 \* 2 - 2 \* 1 = 0; ans = ans + cnt[0] = 1; cnt[0] = cnt[0] + 1 = 2.

7. After iterating through all characters, ans holds the total count of valid ratio substrings. In this case, ans = 2.

substrings with ratios of 0 s to 1 s of 2:1 without needing to check every possible substring explicitly.

difference, it indicated that a valid substring ended at the current character, thus we added the count from cnt.

# Initialize a Counter to keep track of the differences counter = Counter({0: 1})

# Calculate the difference between the counts of '1's and '0's multiplied by respective input factors

// Function to calculate the number of substrings with a fixed ratio between the number of '0's and '1's.

By keeping track of the differences and using them to map to potential substring start points, we've efficiently counted the

# Return the final accumulated result return ans

```
long answer = 0;
// HashMap to store the counts of differences computed.
Map<Long, Long> countMap = new HashMap<>();
// Initially put a difference of '0' with a count of '1' into the map.
countMap.put(0L, 1L);
```

C++

#include <string>

class Solution {

#include <unordered\_map>

using ll = long long;

Java

import java.util.HashMap;

return answer;

import java.util.Map;

class Solution {

```
public:
    // This function calculates the number of substrings where
    // the ratio of the number of '1's to the number of '0's is num1 : num2.
    long long fixedRatio(const string& s, int num1, int num2) {
        // Initialize counters for '0's and '1's, and for the answer.
        ll count0 = 0, count1 = 0;
        Il answer = 0;
        // Create a hash map to store the frequency of each ratio difference.
        unordered map<ll, ll> frequencyCounter:
        // Initialize the ratio difference of 0 with a count of 1.
        frequencyCounter[0] = 1;
        // Iterate through each character in the string.
        for (const char& c : s) {
            // Increment the count for '0's or '1's based on the current character.
            count0 += (c == '0');
            count1 += (c == '1');
            // Calculate the current difference in the scaled counts of '1's and '0's.
            ll difference = count1 * num1 - count0 * num2;
            // Increment the answer by the number of times this difference has been seen.
            answer += frequencyCounter[difference];
            // Record the current difference by incrementing its count.
            ++frequencyCounter[difference];
        // Return the total count of valid substrings.
        return answer;
TypeScript
// Importing the necessary module to use the Map data structure.
import { Map } from "typescript-collections";
// Define an alias 'long' for the 'number' type for long integer simulation.
type long = number;
// Function to calculate the number of substrings where
// the ratio of the number of '1s' to the number of '0s' is num1 : num2.
function fixedRatio(s: string, num1: number, num2: number): long {
  // Initialize counters for '0s' and '1s', as well as the answer.
  let countZeroes: long = 0, countOnes: long = 0;
  let answer: long = 0;
  // Initialize a map to store the frequency of each ratio difference.
  let frequencyCounter: Map<long, long> = new Map<long, long>();
  // Initialize the ratio difference of 0 with a count of 1.
  frequencyCounter.setValue(0, 1);
  // Iterate over each character in the string.
  for (const c of s) {
    // Increment the count for '0s' or '1s' based on the current character.
    if (c === '0') {
```

## # Increment the count of '0's and '1's based on the current character count zeros += char == '0' count\_ones += char == '1' # Calculate the difference between the counts of '1's and '0's multiplied by respective input factors

return ans

**Time Complexity** 

Time and Space Complexity

for char in string:

countZeroes++;

// Calculate the current difference in the scaled counts of '1s' and '0s'.

let existingFrequency = frequencyCounter.getValue(difference) || 0;

// Return the total count of substrings that fulfill the ratio condition.

def fixedRatio(self, string: str, num zeros: int, num ones: int) -> int:

difference = count\_ones \* num\_zeros - count\_zeros \* num\_ones

# Accumulate the number of occurrences of the current difference

# Increment the count for the current difference in the counter

# Initialize counters for '0's and '1's and result variable ans

# Initialize a Counter to keep track of the differences

# Iterate over each character in the input string

// console.log(`Number of valid substrings: \${fixedRatio(sampleString, ratioNum1, ratioNum2)}`);

// Record the current difference by incrementing its frequency.

frequencyCounter.setValue(difference, existingFrequency + 1);

// Uncomment the following code to test the functionality

let difference = countOnes \* num1 - countZeroes \* num2;

// increment the answer by this amount.

answer += existingFrequency;

// let sampleString: string = "0110101";

count zeros = count\_ones = 0

ans += counter[difference]

counter[difference] += 1

# Return the final accumulated result

counter = Counter({0: 1})

// let ratioNum1: number = 2:

// let ratioNum2: number = 1;

from collections import Counter

// Get the number of times this difference has been seen.

countOnes++;

} else {

return answer;

class Solution:

ans = 0

The given code primarily consists of a single loop that iterates over all characters in the input string s. Inside this loop, the operations performed are constant time operations, including comparison, addition, and dictionary access or update.

• The dictionary operations (cnt[x] and cnt[x] += 1): usually take 0(1) time, thanks to the hash table implementation of Python dictionaries. Since these 0(1) operations are performed once for each of the n characters in the input string, the overall time complexity is

• no and n1: these are just two integer counters, which use 0(1) space.

O(n), where n is the length of the string s.

• The additions (n0 += ..., n1 += ...): also take 0(1) time each.

• The comparison ( c == '0', c == '1'): takes 0(1) time each.

- **Space Complexity** The space complexity comes from the variables n0, n1, and the Counter dictionary cnt.
- cnt: at worst, it will contain a distinct count for every prefix sum difference encountered. In the worst case, each prefix difference is unique, leading to n entries.

Therefore, the worst-case space complexity is O(n), where n is the length of the string s.