Problem 930: Binary Subarrays With Sum

Problem Information

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

Problem Description

Given a binary array

nums

and an integer

goal

, return

the number of non-empty

subarrays

with a sum

goal

.

A

subarray

is a contiguous part of the array.

Example 1:

Input:

nums = [1,0,1,0,1], goal = 2

Output:

4

Explanation:

The 4 subarrays are bolded and underlined below: [

1,0,1

,0,1] [

1,0,1,0

,1] [1,

0,1,0,1

] [1,0,

1,0,1

]

Example 2:

Input:

nums = [0,0,0,0,0], goal = 0

Output:

15

Constraints:

1 <= nums.length <= 3 * 10

4

nums[i]

is either

0

or

1

.

0 <= goal <= nums.length

Code Snippets

C++:

```
class Solution {
public:
int numSubarraysWithSum(vector<int>& nums, int goal) {

}
};
```

Java:

```
class Solution {
public int numSubarraysWithSum(int[] nums, int goal) {

}
}
```

Python3:

```
class Solution:
def numSubarraysWithSum(self, nums: List[int], goal: int) -> int:
```

Python:

```
class Solution(object):
def numSubarraysWithSum(self, nums, goal):
"""
:type nums: List[int]
:type goal: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} goal
* @return {number}
*/
var numSubarraysWithSum = function(nums, goal) {

};
```

TypeScript:

```
function numSubarraysWithSum(nums: number[], goal: number): number {

};
```

C#:

```
public class Solution {
public int NumSubarraysWithSum(int[] nums, int goal) {

}
}
```

C:

```
int numSubarraysWithSum(int* nums, int numsSize, int goal) {

}
```

Go:

```
func numSubarraysWithSum(nums []int, goal int) int {

}
```

Kotlin:

```
class Solution {
fun numSubarraysWithSum(nums: IntArray, goal: Int): Int {

}
}
```

Swift:

```
class Solution {
func numSubarraysWithSum(_ nums: [Int], _ goal: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_subarrays_with_sum(nums: Vec<i32>, goal: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} goal
# @return {Integer}
def num_subarrays_with_sum(nums, goal)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $goal
* @return Integer
*/
function numSubarraysWithSum($nums, $goal) {

}
}
```

Dart:

```
class Solution {
int numSubarraysWithSum(List<int> nums, int goal) {

}
}
```

Scala:

```
object Solution {
def numSubarraysWithSum(nums: Array[Int], goal: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_subarrays_with_sum(nums :: [integer], goal :: integer) :: integer
def num_subarrays_with_sum(nums, goal) do

end
end
```

Erlang:

```
-spec num_subarrays_with_sum(Nums :: [integer()], Goal :: integer()) ->
integer().
```

```
num_subarrays_with_sum(Nums, Goal) ->
.
```

Racket:

```
(define/contract (num-subarrays-with-sum nums goal)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Binary Subarrays With Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int numSubarraysWithSum(vector<int>& nums, int goal) {

}
};
```

Java Solution:

```
/**
* Problem: Binary Subarrays With Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

class Solution {
public int numSubarraysWithSum(int[] nums, int goal) {

}
}
```

Python3 Solution:

```
"""
Problem: Binary Subarrays With Sum
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def numSubarraysWithSum(self, nums: List[int], goal: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numSubarraysWithSum(self, nums, goal):
"""
:type nums: List[int]
:type goal: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Binary Subarrays With Sum
* Difficulty: Medium
* Tags: array, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @param {number} goal
* @return {number}
*/
var numSubarraysWithSum = function(nums, goal) {

};
```

TypeScript Solution:

```
/**
* Problem: Binary Subarrays With Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function numSubarraysWithSum(nums: number[], goal: number): number {

};
```

C# Solution:

```
/*
* Problem: Binary Subarrays With Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

public class Solution {
public int NumSubarraysWithSum(int[] nums, int goal) {

}
}
```

C Solution:

```
/*
* Problem: Binary Subarrays With Sum
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int numSubarraysWithSum(int* nums, int numsSize, int goal) {

}
```

Go Solution:

```
// Problem: Binary Subarrays With Sum
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func numSubarraysWithSum(nums []int, goal int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numSubarraysWithSum(nums: IntArray, goal: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numSubarraysWithSum(_ nums: [Int], _ goal: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Binary Subarrays With Sum
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn num_subarrays_with_sum(nums: Vec<i32>, goal: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} goal
# @return {Integer}
def num_subarrays_with_sum(nums, goal)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $goal
* @return Integer
*/
function numSubarraysWithSum($nums, $goal) {

}
}
```

Dart Solution:

```
class Solution {
int numSubarraysWithSum(List<int> nums, int goal) {

}
}
```

Scala Solution:

```
object Solution {
def numSubarraysWithSum(nums: Array[Int], goal: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_subarrays_with_sum(nums :: [integer], goal :: integer) :: integer
def num_subarrays_with_sum(nums, goal) do

end
end
```

Erlang Solution:

```
-spec num_subarrays_with_sum(Nums :: [integer()], Goal :: integer()) ->
integer().
num_subarrays_with_sum(Nums, Goal) ->
```

```
.
```

Racket Solution:

```
(define/contract (num-subarrays-with-sum nums goal)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```