Problem 3728: Stable Subarrays With Equal Boundary and Interior Sum

Problem Information

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

Problem Description

You are given an integer array

capacity

.

A

subarray

capacity[l..r]

is considered

stable

if:

Its length is

at least

3.

The

first

and

last

elements are each equal to the

sum

of all elements

strictly between

them (i.e.,

capacity[l] = capacity[r] = capacity[l + 1] + capacity[l + 2] + ... + capacity[r - 1]

).

Return an integer denoting the number of

stable subarrays

.

Example 1:

Input:

capacity = [9,3,3,3,9]

Output:

2

Explanation:

[9,3,3,3,9]

is stable because the first and last elements are both 9, and the sum of the elements strictly between them is

3 + 3 + 3 = 9

.

[3,3,3]

is stable because the first and last elements are both 3, and the sum of the elements strictly between them is 3.

Example 2:

Input:

capacity = [1,2,3,4,5]

Output:

0

Explanation:

No subarray of length at least 3 has equal first and last elements, so the answer is 0.

Example 3:

Input:

capacity = [-4,4,0,0,-8,-4]

Output:

1

Explanation:

[-4,4,0,0,-8,-4]

is stable because the first and last elements are both -4, and the sum of the elements strictly between them is

4 + 0 + 0 + (-8) = -4

Constraints:

3 <= capacity.length <= 10

5

-10

9

<= capacity[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
long long countStableSubarrays(vector<int>& capacity) {

}
};
```

Java:

```
class Solution {
public long countStableSubarrays(int[] capacity) {

}
}
```

Python3:

```
class Solution:
def countStableSubarrays(self, capacity: List[int]) -> int:
```

Python:

```
class Solution(object):
def countStableSubarrays(self, capacity):
"""
:type capacity: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} capacity
* @return {number}
*/
var countStableSubarrays = function(capacity) {

};
```

TypeScript:

```
function countStableSubarrays(capacity: number[]): number {

};
```

C#:

```
public class Solution {
public long CountStableSubarrays(int[] capacity) {

}
}
```

C:

```
long long countStableSubarrays(int* capacity, int capacitySize) {

}
```

Go:

```
func countStableSubarrays(capacity []int) int64 {

}
```

Kotlin:

```
class Solution {
fun countStableSubarrays(capacity: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func countStableSubarrays(_ capacity: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_stable_subarrays(capacity: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} capacity
# @return {Integer}
def count_stable_subarrays(capacity)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[] $capacity
* @return Integer
*/
function countStableSubarrays($capacity) {

}
}
```

Dart:

```
class Solution {
int countStableSubarrays(List<int> capacity) {

}
}
```

Scala:

```
object Solution {
def countStableSubarrays(capacity: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_stable_subarrays(capacity :: [integer]) :: integer
def count_stable_subarrays(capacity) do

end
end
```

Erlang:

```
-spec count_stable_subarrays(Capacity :: [integer()]) -> integer().
count_stable_subarrays(Capacity) ->
.
```

Racket:

```
(define/contract (count-stable-subarrays capacity)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Stable Subarrays With Equal Boundary and Interior 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:
long long countStableSubarrays(vector<int>& capacity) {

}
};
```

Java Solution:

```
/**
* Problem: Stable Subarrays With Equal Boundary and Interior 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 long countStableSubarrays(int[] capacity) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Stable Subarrays With Equal Boundary and Interior 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 countStableSubarrays(self, capacity: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countStableSubarrays(self, capacity):
"""
:type capacity: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Stable Subarrays With Equal Boundary and Interior 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[]} capacity
* @return {number}
*/
var countStableSubarrays = function(capacity) {

};
```

TypeScript Solution:

```
/**
* Problem: Stable Subarrays With Equal Boundary and Interior 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 countStableSubarrays(capacity: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Stable Subarrays With Equal Boundary and Interior 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 long CountStableSubarrays(int[] capacity) {

}
}
```

C Solution:

```
/*
* Problem: Stable Subarrays With Equal Boundary and Interior 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
*/

long long countStableSubarrays(int* capacity, int capacitySize) {

}
```

Go Solution:

```
// Problem: Stable Subarrays With Equal Boundary and Interior 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 countStableSubarrays(capacity []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun countStableSubarrays(capacity: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func countStableSubarrays(_ capacity: [Int]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Stable Subarrays With Equal Boundary and Interior 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 count_stable_subarrays(capacity: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} capacity
# @return {Integer}
def count_stable_subarrays(capacity)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $capacity
* @return Integer
*/
function countStableSubarrays($capacity) {

}
}
```

Dart Solution:

```
class Solution {
int countStableSubarrays(List<int> capacity) {

}
}
```

Scala Solution:

```
object Solution {
def countStableSubarrays(capacity: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_stable_subarrays(capacity :: [integer]) :: integer
def count_stable_subarrays(capacity) do

end
end
```

Erlang Solution:

```
-spec count_stable_subarrays(Capacity :: [integer()]) -> integer().
count_stable_subarrays(Capacity) ->
.
```

Racket Solution:

```
(define/contract (count-stable-subarrays capacity)
(-> (listof exact-integer?) exact-integer?)
)
```