Problem 2913: Subarrays Distinct Element Sum of Squares I

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

.

The

distinct count

of a subarray of

nums

is defined as:

Let

nums[i..j]

be a subarray of

nums

consisting of all the indices from

i

to

j

such that

0 <= i <= j < nums.length

. Then the number of distinct values in

nums[i..j]

is called the distinct count of

nums[i..j]

.

Return

the sum of the

squares

of

distinct counts

of all subarrays of

nums

.

A subarray is a contiguous

non-empty

sequence of elements within an array.

Example 1:

Input:

nums = [1,2,1]

Output:

15

Explanation:

Six possible subarrays are: [1]: 1 distinct value [2]: 1 distinct value [1]: 1 distinct value [1,2]: 2 distinct values [2,1]: 2 distinct values [1,2,1]: 2 distinct values The sum of the squares of the distinct counts in all subarrays is equal to 1

2

+ 1

2

+ 1

2

+ 2

2

+ 2

2

+ 2

2

= 15.

Example 2:

Input:

nums = [1,1]

Output:

3

Explanation:

Three possible subarrays are: [1]: 1 distinct value [1]: 1 distinct value [1,1]: 1 distinct value
The sum of the squares of the distinct counts in all subarrays is equal to 1

2

+ 1

2

+ 1

2

= 3.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int sumCounts(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int sumCounts(List<Integer> nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
```

```
var sumCounts = function(nums) {

};
```

TypeScript:

```
function sumCounts(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int SumCounts(IList<int> nums) {

}
}
```

C:

```
int sumCounts(int* nums, int numsSize) {

}
```

Go:

```
func sumCounts(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun sumCounts(nums: List<Int>): Int {

}
}
```

Swift:

```
class Solution {
func sumCounts(_ nums: [Int]) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn sum_counts(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def sum_counts(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function sumCounts($nums) {

}
}
```

Dart:

```
class Solution {
int sumCounts(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def sumCounts(nums: List[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sum_counts(nums :: [integer]) :: integer
def sum_counts(nums) do

end
end
```

Erlang:

```
-spec sum_counts(Nums :: [integer()]) -> integer().
sum_counts(Nums) ->
.
```

Racket:

```
(define/contract (sum-counts nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Subarrays Distinct Element Sum of Squares I
* Difficulty: Easy
* 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 sumCounts(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Subarrays Distinct Element Sum of Squares I
* Difficulty: Easy
* 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 sumCounts(List<Integer> nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Subarrays Distinct Element Sum of Squares I
Difficulty: Easy
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 sumCounts(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Subarrays Distinct Element Sum of Squares I
* Difficulty: Easy
* 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
* @return {number}
*/
var sumCounts = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Subarrays Distinct Element Sum of Squares I
* Difficulty: Easy
* 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 sumCounts(nums: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Subarrays Distinct Element Sum of Squares I
* Difficulty: Easy
* 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 SumCounts(IList<int> nums) {

}
}
```

C Solution:

```
/*
* Problem: Subarrays Distinct Element Sum of Squares I
* Difficulty: Easy
* 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 sumCounts(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Subarrays Distinct Element Sum of Squares I
// Difficulty: Easy
```

```
// 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 sumCounts(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun sumCounts(nums: List<Int>): Int {

}
}
```

Swift Solution:

```
class Solution {
func sumCounts(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Subarrays Distinct Element Sum of Squares I
// Difficulty: Easy
// 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 sum_counts(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def sum_counts(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function sumCounts($nums) {

}
}
```

Dart Solution:

```
class Solution {
int sumCounts(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def sumCounts(nums: List[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sum_counts(nums :: [integer]) :: integer
def sum_counts(nums) do
```

```
end
end
```

Erlang Solution:

```
-spec sum_counts(Nums :: [integer()]) -> integer().
sum_counts(Nums) ->
.
```

Racket Solution:

```
(define/contract (sum-counts nums)
(-> (listof exact-integer?) exact-integer?)
)
```