Problem 3672: Sum of Weighted Modes in Subarrays

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

.

For every

subarray

of length

k

:

The

mode

is defined as the element with the

highest frequency

. If there are multiple choices for a mode, the

smallest

such element is taken.

The

weight

is defined as

mode * frequency(mode)

.

Return the

sum

of the weights of all

subarrays

of length

k

.

Note:

A

subarray

is a contiguous

non-empty

sequence of elements within an array.

The

frequency

of an element

x

is the number of times it occurs in the array.

Example 1:

Input:

nums = [1,2,2,3], k = 3

Output:

8

Explanation:

Subarrays of length

k = 3

are:

Subarray

Frequencies

Mode

Mode

Frequency

Weight

[1, 2, 2]

1: 1, 2: 2

2

2

2 × 2 = 4

[2, 2, 3]

2: 2, 3: 1

2

2

2 × 2 = 4

Thus, the sum of weights is

4 + 4 = 8

.

Example 2:

Input:

nums = [1,2,1,2], k = 2

Output:

3

Explanation:

Subarrays of length

k = 2

are:

Subarray

Frequencies

Mode

Mode

Frequency

Weight

[1, 2]

1: 1, 2: 1

1

1

1 × 1 = 1

[2, 1]

2: 1, 1: 1

1

1

1 × 1 = 1

[1, 2]

1: 1, 2: 1

1

1

1 × 1 = 1

Thus, the sum of weights is

1 + 1 + 1 = 3

.

Example 3:

Input:

nums = [4,3,4,3], k = 3

Output:

14

Explanation:

Subarrays of length

k = 3

are:

Subarray

Frequencies

Mode

Mode

Frequency

Weight

[4, 3, 4]

4: 2, 3: 1

4

2

2 × 4 = 8

[3, 4, 3]

3: 2, 4: 1

3

2

2 × 3 = 6

Thus, the sum of weights is

8 + 6 = 14

.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

1 <= k <= nums.length

Code Snippets

C++:

```
class Solution {
public:
long long modeWeight(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public long modeWeight(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

```
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function modeWeight(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long ModeWeight(int[] nums, int k) {

}
}
```

C:

```
long long modeWeight(int* nums, int numsSize, int k) {

}
```

Go:

```
func modeWeight(nums []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun modeWeight(nums: IntArray, k: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn mode_weight(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def mode_weight(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function modeWeight($nums, $k) {

}
```

```
}
```

Dart:

```
class Solution {
int modeWeight(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def modeWeight(nums: Array[Int], k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec mode_weight(nums :: [integer], k :: integer) :: integer
def mode_weight(nums, k) do

end
end
```

Erlang:

```
-spec mode_weight(Nums :: [integer()], K :: integer()) -> integer().
mode_weight(Nums, K) ->
.
```

Racket:

```
(define/contract (mode-weight nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Weighted Modes in Subarrays
* 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 modeWeight(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Weighted Modes in Subarrays
* 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 modeWeight(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Weighted Modes in Subarrays
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 modeWeight(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Weighted Modes in Subarrays
* 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} k
* @return {number}
*/
var modeWeight = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Weighted Modes in Subarrays
* 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 modeWeight(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Weighted Modes in Subarrays
* 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 ModeWeight(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Weighted Modes in Subarrays
* 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 modeWeight(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Sum of Weighted Modes in Subarrays
// 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 modeWeight(nums []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun modeWeight(nums: IntArray, k: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Weighted Modes in Subarrays
// 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 mode_weight(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def mode_weight(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function modeWeight($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int modeWeight(List<int> nums, int k) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def modeWeight(nums: Array[Int], k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec mode_weight(nums :: [integer], k :: integer) :: integer
def mode_weight(nums, k) do

end
end
```

Erlang Solution:

```
-spec mode_weight(Nums :: [integer()], K :: integer()) -> integer().
mode_weight(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (mode-weight nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```