

Problem 2845: Count of Interesting Subarrays

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

integer array

nums

, an integer

modulo

, and an integer

k

Your task is to find the count of subarrays that are

interesting

A

subarray

$\text{nums}[l..r]$

is

interesting

if the following condition holds:

Let

cnt

be the number of indices

i

in the range

$[l, r]$

such that

$\text{nums}[i] \% \text{modulo} == k$

. Then,

$\text{cnt \% modulo} == k$

.

Return

an integer denoting the count of interesting subarrays.

Note:

A subarray is

a contiguous non-empty sequence of elements within an array

.

Example 1:

Input:

nums = [3,2,4], modulo = 2, k = 1

Output:

3

Explanation:

In this example the interesting subarrays are: The subarray nums[0..0] which is [3]. - There is only one index, $i = 0$, in the range $[0, 0]$ that satisfies $\text{nums}[i] \% \text{modulo} == k$. - Hence, $\text{cnt} = 1$ and $\text{cnt \% modulo} == k$. The subarray nums[0..1] which is [3,2]. - There is only one index, $i = 0$, in the range $[0, 1]$ that satisfies $\text{nums}[i] \% \text{modulo} == k$. - Hence, $\text{cnt} = 1$ and $\text{cnt \% modulo} == k$. The subarray nums[0..2] which is [3,2,4]. - There is only one index, $i = 0$, in the range $[0, 2]$ that satisfies $\text{nums}[i] \% \text{modulo} == k$. - Hence, $\text{cnt} = 1$ and $\text{cnt \% modulo} == k$. It can be shown that there are no other interesting subarrays. So, the answer is 3.

Example 2:

Input:

nums = [3,1,9,6], modulo = 3, k = 0

Output:

2

Explanation:

In this example the interesting subarrays are: The subarray nums[0..3] which is [3,1,9,6]. - There are three indices, $i = 0, 2, 3$, in the range $[0, 3]$ that satisfy $\text{nums}[i] \% \text{modulo} == k$. - Hence, $\text{cnt} = 3$ and $\text{cnt \% modulo} == k$. The subarray nums[1..1] which is [1]. - There is no

index, i, in the range [1, 1] that satisfies $\text{nums}[i] \% \text{modulo} == k$. - Hence, $\text{cnt} = 0$ and $\text{cnt} \% \text{modulo} == k$. It can be shown that there are no other interesting subarrays. So, the answer is 2.

Constraints:

$1 \leq \text{nums.length} \leq 10$

5

$1 \leq \text{nums}[i] \leq 10$

9

$1 \leq \text{modulo} \leq 10$

9

$0 \leq k < \text{modulo}$

Code Snippets

C++:

```
class Solution {
public:
    long long countInterestingSubarrays(vector<int>& nums, int modulo, int k) {
        }
    };
}
```

Java:

```
class Solution {
public long countInterestingSubarrays(List<Integer> nums, int modulo, int k)
{
}
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function countInterestingSubarrays(nums: number[], modulo: number, k:  
    number): number {  
  
};
```

C#:

```
public class Solution {  
    public long CountInterestingSubarrays(IList<int> nums, int modulo, int k) {  
        }  
    }
```

C:

```
long long countInterestingSubarrays(int* nums, int numssSize, int modulo, int k) {  
    }  
}
```

Go:

```
func countInterestingSubarrays(nums []int, modulo int, k int) int64 {  
    }  
}
```

Kotlin:

```
class Solution {  
    fun countInterestingSubarrays(nums: List<Int>, modulo: Int, k: Int): Long {  
        }  
    }  
}
```

Swift:

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

Rust:

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

Ruby:

```
# @param {Integer[]} nums  
# @param {Integer} modulo
```

```

# @param {Integer} k
# @return {Integer}
def count_interesting_subarrays(nums, modulo, k)

end

```

PHP:

```

class Solution {

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

    }
}

```

Dart:

```

class Solution {
int countInterestingSubarrays(List<int> nums, int modulo, int k) {

}
}

```

Scala:

```

object Solution {
def countInterestingSubarrays(nums: List[Int], modulo: Int, k: Int): Long = {

}
}

```

Elixir:

```

defmodule Solution do
@spec count_interesting_subarrays(nums :: [integer], modulo :: integer, k :: integer) :: integer

```

```

def count_interesting_subarrays(nums, modulo, k) do
  end
end

```

Erlang:

```

-spec count_interesting_subarrays(Nums :: [integer()], Modulo :: integer(), K
:: integer()) -> integer().
count_interesting_subarrays(Nums, Modulo, K) ->
  .

```

Racket:

```

(define/contract (count-interesting-subarrays nums modulo k)
  (-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?))

```

Solutions

C++ Solution:

```

/*
 * Problem: Count of Interesting 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 countInterestingSubarrays(vector<int>& nums, int modulo, int k) {
        }
    };

```

Java Solution:

```

/**
 * Problem: Count of Interesting 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 countInterestingSubarrays(List<Integer> nums, int modulo, int k) {
        return 0;
    }
}

```

Python3 Solution:

```

"""
Problem: Count of Interesting 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 countInterestingSubarrays(self, nums: List[int], modulo: int, k: int) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

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

```

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

```

JavaScript Solution:

```
/**
 * Problem: Count of Interesting 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} modulo
 * @param {number} k
 * @return {number}
 */
var countInterestingSubarrays = function(nums, modulo, k) {

};


```

TypeScript Solution:

```
/**
 * Problem: Count of Interesting 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 countInterestingSubarrays(nums: number[], modulo: number, k:
number): number {
```

```
};
```

C# Solution:

```
/*
 * Problem: Count of Interesting 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 CountInterestingSubarrays(IList<int> nums, int modulo, int k) {
        return 0;
    }
}
```

C Solution:

```
/*
 * Problem: Count of Interesting 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 countInterestingSubarrays(int* nums, int numsSize, int modulo, int k) {
    return 0;
}
```

Go Solution:

```
// Problem: Count of Interesting 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 countInterestingSubarrays(nums []int, modulo int, k int) int64 {
}

```

Kotlin Solution:

```

class Solution {
    fun countInterestingSubarrays(nums: List<Int>, modulo: Int, k: Int): Long {
        return 0
    }
}

```

Swift Solution:

```

class Solution {
    func countInterestingSubarrays(_ nums: [Int], _ modulo: Int, _ k: Int) -> Int {
        return 0
    }
}

```

Rust Solution:

```

// Problem: Count of Interesting 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 count_interesting_subarrays(nums: Vec<i32>, modulo: i32, k: i32) -> i64 {
}

```

```
}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} modulo
# @param {Integer} k
# @return {Integer}

def count_interesting_subarrays(nums, modulo, k)

end
```

PHP Solution:

```
class Solution {

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

    }
}
```

Dart Solution:

```
class Solution {
  int countInterestingSubarrays(List<int> nums, int modulo, int k) {
}
```

Scala Solution:

```
object Solution {
  def countInterestingSubarrays(nums: List[Int], modulo: Int, k: Int): Long = {
```

```
}
```

```
}
```

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-interesting-subarrays nums modulo k)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?))
)
```