Problem 2261: K Divisible Elements Subarrays

Problem Information

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

Problem Description

Given an integer array

nums

and two integers

k

and

p

, return

the number of

distinct subarrays,

which have

at most

k

elements

that are

divisible by

p

.

Two arrays

nums1

and

nums2

are said to be

distinct

if:

They are of

different

lengths, or

There exists

at least

one index

i

where

nums1[i] != nums2[i]

.

A

subarray

is defined as a

non-empty

contiguous sequence of elements in an array.

Example 1:

Input:

nums = [

2

,3,3,

2

,

2

], k = 2, p = 2

Output:

11

Explanation:

The elements at indices 0, 3, and 4 are divisible by p = 2. The 11 distinct subarrays which have at most k = 2 elements divisible by 2 are: [2], [2,3], [2,3,3], [2,3,3,2], [3], [3,3], [3,3,2], [3,3,2,2], [3,2], [3,2,2], and [2,2]. Note that the subarrays [2] and [3] occur more than once in nums, but they should each be counted only once. The subarray [2,3,3,2,2] should not be counted because it has 3 elements that are divisible by 2.

Example 2:

Input:

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

Output:

10

Explanation:

All element of nums are divisible by p = 1. Also, every subarray of nums will have at most 4 elements that are divisible by 1. Since all subarrays are distinct, the total number of subarrays satisfying all the constraints is 10.

Constraints:

1 <= nums.length <= 200

1 <= nums[i], p <= 200

1 <= k <= nums.length

Follow up:

Can you solve this problem in O(n

2

) time complexity?

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int countDistinct(int[] nums, int k, int p) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

}

TypeScript:

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

};
```

C#:

```
public class Solution {
public int CountDistinct(int[] nums, int k, int p) {

}
}
```

C:

```
int countDistinct(int* nums, int numsSize, int k, int p) {

}
```

Go:

```
func countDistinct(nums []int, k int, p int) int {

}
```

Kotlin:

```
class Solution {
fun countDistinct(nums: IntArray, k: Int, p: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def countDistinct(nums: Array[Int], k: Int, p: Int): Int = {
}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-distinct nums k p)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: K Divisible Elements 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:
int countDistinct(vector<int>& nums, int k, int p) {

}
};
```

Java Solution:

```
/**
* Problem: K Divisible Elements 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 int countDistinct(int[] nums, int k, int p) {

}
}
```

Python3 Solution:

```
"""
Problem: K Divisible Elements 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 countDistinct(self, nums: List[int], k: int, p: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: K Divisible Elements 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
* @param {number} p
* @return {number}
*/
var countDistinct = function(nums, k, p) {

};
```

TypeScript Solution:

```
/**
* Problem: K Divisible Elements 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 countDistinct(nums: number[], k: number, p: number): number {

};
```

C# Solution:

```
/*
* Problem: K Divisible Elements 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 int CountDistinct(int[] nums, int k, int p) {

}
}
```

C Solution:

```
/*
* Problem: K Divisible Elements 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
*/

int countDistinct(int* nums, int numsSize, int k, int p) {
}
```

Go Solution:

```
// Problem: K Divisible Elements 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 countDistinct(nums []int, k int, p int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countDistinct(nums: IntArray, k: Int, p: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: K Divisible Elements 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_distinct(nums: Vec<i32>, k: i32, p: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-distinct nums k p)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```