Problem 992: Subarrays with K Different Integers

Problem Information

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

Problem Description

Given an integer array

nums

and an integer

k

, return

the number of

good subarrays

of

nums

.

A

good array

is an array where the number of different integers in that array is exactly

k

.

For example,

[1,2,3,1,2]

has

3

different integers:

1

,

2

, and

3

.

A

subarray

is a

contiguous

part of an array.

Example 1:

Input:

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

Output:

7

Explanation:

Subarrays formed with exactly 2 different integers: [1,2], [2,1], [1,2], [2,3], [1,2,1], [2,1,2], [1,2,1,2]

Example 2:

Input:

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

Output:

3

Explanation:

Subarrays formed with exactly 3 different integers: [1,2,1,3], [2,1,3], [1,3,4].

Constraints:

1 <= nums.length <= 2 * 10

4

1 <= nums[i], k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int subarraysWithKDistinct(int* nums, int numsSize, int k) {

}
```

Go:

```
func subarraysWithKDistinct(nums []int, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (subarrays-with-k-distinct nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Subarrays with K Different Integers
* Difficulty: Hard
* 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 subarraysWithKDistinct(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Subarrays with K Different Integers
* Difficulty: Hard
* 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 subarraysWithKDistinct(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Subarrays with K Different Integers
Difficulty: Hard
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 subarraysWithKDistinct(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Subarrays with K Different Integers
* Difficulty: Hard
* 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 subarraysWithKDistinct = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Subarrays with K Different Integers
* Difficulty: Hard
* 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 subarraysWithKDistinct(nums: number[], k: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Subarrays with K Different Integers
* Difficulty: Hard
* 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 SubarraysWithKDistinct(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Subarrays with K Different Integers
* Difficulty: Hard
* 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 subarraysWithKDistinct(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Subarrays with K Different Integers
// Difficulty: Hard
```

```
// 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 subarraysWithKDistinct(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Subarrays with K Different Integers
// Difficulty: Hard
// 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 subarrays_with_k_distinct(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (subarrays-with-k-distinct nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```