Problem 2958: Length of Longest Subarray With at Most K Frequency

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

.

The

frequency

of an element

x

is the number of times it occurs in an array.

An array is called

good

if the frequency of each element in this array is less than or equal

to

k

.

Return

the length of the

longest

good

subarray of

nums

.

A

subarray

is a contiguous non-empty sequence of elements within an array.

Example 1:

Input:

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

Output:

6

Explanation:

The longest possible good subarray is [1,2,3,1,2,3] since the values 1, 2, and 3 occur at most twice in this subarray. Note that the subarrays [2,3,1,2,3,1] and [3,1,2,3,1,2] are also good. It can be shown that there are no good subarrays with length more than 6.

Example 2:

Input:

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

Output:

2

Explanation:

The longest possible good subarray is [1,2] since the values 1 and 2 occur at most once in this subarray. Note that the subarray [2,1] is also good. It can be shown that there are no good subarrays with length more than 2.

Example 3:

Input:

nums = [5,5,5,5,5,5,5], k = 4

Output:

4

Explanation:

The longest possible good subarray is [5,5,5,5] since the value 5 occurs 4 times in this subarray. It can be shown that there are no good subarrays with length more than 4.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}

}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-subarray-length nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Length of Longest Subarray With at Most K Frequency
* 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 maxSubarrayLength(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Length of Longest Subarray With at Most K Frequency
* 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 maxSubarrayLength(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Length of Longest Subarray With at Most K Frequency
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 maxSubarrayLength(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Length of Longest Subarray With at Most K Frequency
* 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 maxSubarrayLength = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Length of Longest Subarray With at Most K Frequency
* 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 maxSubarrayLength(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Length of Longest Subarray With at Most K Frequency
* 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 MaxSubarrayLength(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Length of Longest Subarray With at Most K Frequency
* 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 maxSubarrayLength(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Length of Longest Subarray With at Most K Frequency
// 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 maxSubarrayLength(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Length of Longest Subarray With at Most K Frequency
// 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 max_subarray_length(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-subarray-length nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```