Problem 3434: Maximum Frequency After Subarray Operation

Problem Information

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

Problem Description

You are given an array

nums

of length

n

. You are also given an integer

k

.

You perform the following operation on

nums

once

:

Select a

subarray

nums[i..j]

where

0 <= i <= j <= n - 1

.

Select an integer

x

and add

x

to

all

the elements in

nums[i..j]

.

Find the

maximum

frequency of the value

k

after the operation.

Example 1:

Input:

nums = [1,2,3,4,5,6], k = 1

Output:

2

Explanation:

After adding -5 to

nums[2..5]

, 1 has a frequency of 2 in

[1, 2, -2, -1, 0, 1]

.

Example 2:

Input:

nums = [10,2,3,4,5,5,4,3,2,2], k = 10

Output:

4

Explanation:

After adding 8 to

nums[1..9]

, 10 has a frequency of 4 in

[10, 10, 11, 12, 13, 13, 12, 11, 10, 10]

.

Constraints:

1 <= n == nums.length <= 10

5

1 <= nums[i] <= 50

1 <= k <= 50

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def maxFrequency(self, nums, k):
```

```
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

```
}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Frequency After Subarray Operation
* Difficulty: Medium
* Tags: array, dp, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Maximum Frequency After Subarray Operation
* Difficulty: Medium
* Tags: array, dp, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Frequency After Subarray Operation
```

```
Difficulty: Medium
Tags: array, dp, greedy, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxFrequency(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Frequency After Subarray Operation
* Difficulty: Medium
* Tags: array, dp, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

```
};
```

TypeScript Solution:

```
/**
* Problem: Maximum Frequency After Subarray Operation
* Difficulty: Medium
* Tags: array, dp, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxFrequency(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Frequency After Subarray Operation
* Difficulty: Medium
* Tags: array, dp, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Maximum Frequency After Subarray Operation
```

```
* Difficulty: Medium
* Tags: array, dp, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxFrequency(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Frequency After Subarray Operation
// Difficulty: Medium
// Tags: array, dp, greedy, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxFrequency(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Frequency After Subarray Operation
// Difficulty: Medium
// Tags: array, dp, greedy, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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