Problem 2190: Most Frequent Number Following Key In an Array

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

.

You are also given an integer

key

, which is present in

nums

.

For every unique integer

target

in

nums

,

count

the number of times

target

immediately follows an occurrence of

key

in

nums

. In other words, count the number of indices

i

such that:

0 <= i <= nums.length - 2

,

nums[i] == key

and,

nums[i + 1] == target

.

Return

the

target

with the

maximum

count

. The test cases will be generated such that the

target

with maximum count is unique.

Example 1:

Input:

nums = [1,100,200,1,100], key = 1

Output:

100

Explanation:

For target = 100, there are 2 occurrences at indices 1 and 4 which follow an occurrence of key. No other integers follow an occurrence of key, so we return 100.

Example 2:

Input:

nums = [2,2,2,2,3], key = 2

Output:

2

Explanation:

For target = 2, there are 3 occurrences at indices 1, 2, and 3 which follow an occurrence of key. For target = 3, there is only one occurrence at index 4 which follows an occurrence of key. target = 2 has the maximum number of occurrences following an occurrence of key, so we return 2.

Constraints:

2 <= nums.length <= 1000

1 <= nums[i] <= 1000

The test cases will be generated such that the answer is unique.

Code Snippets

C++:

```
class Solution {
public:
int mostFrequent(vector<int>& nums, int key) {

}
};
```

Java:

```
class Solution {
public int mostFrequent(int[] nums, int key) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function mostFrequent(nums: number[], key: number): number {

};
```

C#:

```
public class Solution {
public int MostFrequent(int[] nums, int key) {

}
}
```

C:

```
int mostFrequent(int* nums, int numsSize, int key) {

}
```

Go:

```
func mostFrequent(nums []int, key int) int {

}
```

Kotlin:

```
class Solution {
fun mostFrequent(nums: IntArray, key: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} key
# @return {Integer}
def most_frequent(nums, key)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
```

```
* @param Integer $key
* @return Integer
*/
function mostFrequent($nums, $key) {

}
}
```

Dart:

```
class Solution {
int mostFrequent(List<int> nums, int key) {

}
}
```

Scala:

```
object Solution {
def mostFrequent(nums: Array[Int], key: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec most_frequent(nums :: [integer], key :: integer) :: integer
def most_frequent(nums, key) do

end
end
```

Erlang:

```
-spec most_frequent(Nums :: [integer()], Key :: integer()) -> integer().
most_frequent(Nums, Key) ->
.
```

Racket:

```
(define/contract (most-frequent nums key)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Most Frequent Number Following Key In an Array
* Difficulty: Easy
* 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 mostFrequent(vector<int>& nums, int key) {

}
};
```

Java Solution:

```
/**
* Problem: Most Frequent Number Following Key In an Array
* Difficulty: Easy
* 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 mostFrequent(int[] nums, int key) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Most Frequent Number Following Key In an Array
Difficulty: Easy
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 mostFrequent(self, nums: List[int], key: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Most Frequent Number Following Key In an Array
* Difficulty: Easy
* 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} key
* @return {number}
*/
var mostFrequent = function(nums, key) {

};
```

TypeScript Solution:

```
/**
* Problem: Most Frequent Number Following Key In an Array
* Difficulty: Easy
* 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 mostFrequent(nums: number[], key: number): number {

};
```

C# Solution:

```
/*
* Problem: Most Frequent Number Following Key In an Array
* Difficulty: Easy
* 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 MostFrequent(int[] nums, int key) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Most Frequent Number Following Key In an Array
* Difficulty: Easy
* 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 mostFrequent(int* nums, int numsSize, int key) {

}
```

Go Solution:

```
// Problem: Most Frequent Number Following Key In an Array
// Difficulty: Easy
// 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 mostFrequent(nums []int, key int) int {

}
```

Kotlin Solution:

```
class Solution {
fun mostFrequent(nums: IntArray, key: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Most Frequent Number Following Key In an Array
// Difficulty: Easy
// 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 most_frequent(nums: Vec<i32>, key: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} key
# @return {Integer}
def most_frequent(nums, key)

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

```
class Solution {
int mostFrequent(List<int> nums, int key) {

}
}
```

Scala Solution:

```
object Solution {
def mostFrequent(nums: Array[Int], key: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec most_frequent(nums :: [integer], key :: integer) :: integer
def most_frequent(nums, key) do

end
end
```

Erlang Solution:

```
-spec most_frequent(Nums :: [integer()], Key :: integer()) -> integer().
most_frequent(Nums, Key) ->
.
```

Racket Solution:

```
(define/contract (most-frequent nums key)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```