Problem 3636: Threshold Majority Queries

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

and an array

queries

, where

queries[i] = [l

i

, r

i

, threshold

i

]

.

Return an array of integers

ans

where

ans[i]

is equal to the element in the subarray

nums[l

i

...r

i

]

that appears

at least

threshold

i

times, selecting the element with the

highest

frequency (choosing the

smallest

in case of a tie), or -1 if no such element

exists

.

Example 1:

Input:

nums = [1,1,2,2,1,1], queries = [[0,5,4],[0,3,3],[2,3,2]]

Output:

[1,-1,2]

Explanation:

Query

Sub-array

Threshold

Frequency table

Answer

[0, 5, 4]

[1, 1, 2, 2, 1, 1]

4

$1 \rightarrow 4$, $2 \rightarrow 2$

1

[0, 3, 3]

[1, 1, 2, 2]

3

$1 \rightarrow 2$, $2 \rightarrow 2$

-1

[2, 3, 2]

[2, 2]

2

$2 \rightarrow 2$

2

Example 2:

Input:

nums = [3,2,3,2,3,2,3], queries = [[0,6,4],[1,5,2],[2,4,1],[3,3,1]]

Output:

[3,2,3,2]

Explanation:

Query

Sub-array

Threshold

Frequency table

Answer

[0, 6, 4]

[3, 2, 3, 2, 3, 2, 3]

4

$3 \rightarrow 4$, $2 \rightarrow 3$

3

[1, 5, 2]

[2, 3, 2, 3, 2]

2

$2 \rightarrow 3$, $3 \rightarrow 2$

2

[2, 4, 1]

[3, 2, 3]

1

$3 \rightarrow 2$, $2 \rightarrow 1$

3

[3, 3, 1]

[2]

1

2 → 1

2

Constraints:

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

4

1 <= nums[i] <= 10

9

1 <= queries.length <= 5 * 10

4

queries[i] = [l

i

, r

i

, threshold

i

]

0 <= l

i

<= r

i

< n

1 <= threshold

i

<= r

i

- l

i

+ 1

Code Snippets

C++:

```
class Solution {
public:
vector<int> subarrayMajority(vector<int>& nums, vector<vector<int>>& queries)
{

}
};
```

Java:

```
class Solution {
public int[] subarrayMajority(int[] nums, int[][] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function subarrayMajority(nums: number[], queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] SubarrayMajority(int[] nums, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
```

```
*/
int* subarrayMajority(int* nums, int numsSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func subarrayMajority(nums []int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun subarrayMajority(nums: IntArray, queries: Array<IntArray>): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
```

```
def subarray_majority(nums, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function subarrayMajority($nums, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> subarrayMajority(List<int> nums, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def subarrayMajority(nums: Array[Int], queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec subarray_majority(nums :: [integer], queries :: [[integer]]) ::
[integer]
def subarray_majority(nums, queries) do
```

```
end
end
```

Erlang:

```
-spec subarray_majority(Nums :: [integer()], Queries :: [[integer()]]) ->
[integer()].
subarray_majority(Nums, Queries) ->
.
```

Racket:

```
(define/contract (subarray-majority nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Threshold Majority Queries
* Difficulty: Hard
* Tags: array, hash, search
*
* 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:
vector<int> subarrayMajority(vector<int>& nums, vector<vector<int>>& queries)
{

}
};
```

Java Solution:

```
/**
* Problem: Threshold Majority Queries
* Difficulty: Hard
* Tags: array, hash, search
*
* 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[] subarrayMajority(int[] nums, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Threshold Majority Queries
Difficulty: Hard
Tags: array, hash, search

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 subarrayMajority(self, nums: List[int], queries: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Threshold Majority Queries
* Difficulty: Hard
* Tags: array, hash, search
*
* 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[][]} queries
* @return {number[]}
*/
var subarrayMajority = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Threshold Majority Queries
* Difficulty: Hard
* Tags: array, hash, search
*
* 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 subarrayMajority(nums: number[], queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Threshold Majority Queries
* Difficulty: Hard
* Tags: array, hash, search
*
* 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[] SubarrayMajority(int[] nums, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Threshold Majority Queries
* Difficulty: Hard
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* subarrayMajority(int* nums, int numsSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Threshold Majority Queries
// Difficulty: Hard
// Tags: array, hash, search
//
```

```
// 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 subarrayMajority(nums []int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun subarrayMajority(nums: IntArray, queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Threshold Majority Queries
// Difficulty: Hard
// Tags: array, hash, search
//
// 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 subarray_majority(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<i32>
{

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def subarray_majority(nums, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function subarrayMajority($nums, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> subarrayMajority(List<int> nums, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def subarrayMajority(nums: Array[Int], queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec subarray_majority(nums :: [integer], queries :: [[integer]]) ::
[integer]
def subarray_majority(nums, queries) do

end
end
```

Erlang Solution:

```
-spec subarray_majority(Nums :: [integer()], Queries :: [[integer()]]) ->
[integer()].
subarray_majority(Nums, Queries) ->
.
```

Racket Solution:

```
(define/contract (subarray-majority nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```