Problem 3488: Closest Equal Element Queries

Problem Information

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

Problem Description

You are given a

circular

array

nums

and an array

queries

.

For each query

i

, you have to find the following:

The

minimum

distance between the element at index

queries[i]

and

any

other index

j

in the

circular

array, where

nums[j] == nums[queries[i]]

. If no such index exists, the answer for that query should be -1.

Return an array

answer

of the

same

size as

queries

, where

answer[i]

represents the result for query

i

.

Example 1:

Input:

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

Output:

[2,-1,3]

Explanation:

Query 0: The element at

queries[0] = 0

is

nums[0] = 1

. The nearest index with the same value is 2, and the distance between them is 2.

Query 1: The element at

queries[1] = 3

is

nums[3] = 4

. No other index contains 4, so the result is -1.

Query 2: The element at

queries[2] = 5

is

nums[5] = 3

. The nearest index with the same value is 1, and the distance between them is 3 (following the circular path:

5 -> 6 -> 0 -> 1

).

Example 2:

Input:

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

Output:

[-1,-1,-1,-1]

Explanation:

Each value in

nums

is unique, so no index shares the same value as the queried element. This results in -1 for all queries.

Constraints:

1 <= queries.length <= nums.length <= 10

5

1 <= nums[i] <= 10

6

0 <= queries[i] < nums.length

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> solveQueries(int[] nums, int[] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number[]} queries
```

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<int> SolveQueries(int[] nums, int[] queries) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun solveQueries(nums: IntArray, queries: IntArray): List<Int> {

}
```

```
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Closest Equal Element Queries
* Difficulty: Medium
* 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> solveQueries(vector<int>& nums, vector<int>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Closest Equal Element Queries
* Difficulty: Medium
* 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 List<Integer> solveQueries(int[] nums, int[] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Closest Equal Element Queries
Difficulty: Medium
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 solveQueries(self, nums: List[int], queries: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Closest Equal Element Queries
* Difficulty: Medium
* 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 solveQueries = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Closest Equal Element Queries
* Difficulty: Medium
* 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 solveQueries(nums: number[], queries: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Closest Equal Element Queries
* Difficulty: Medium
* 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 IList<int> SolveQueries(int[] nums, int[] queries) {

}
}
```

C Solution:

```
/*
* Problem: Closest Equal Element Queries
* Difficulty: Medium
* 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* solveQueries(int* nums, int numsSize, int* queries, int queriesSize,
int* returnSize) {

}
```

Go Solution:

```
// Problem: Closest Equal Element Queries
// Difficulty: Medium
// 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 solveQueries(nums []int, queries []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun solveQueries(nums: IntArray, queries: IntArray): List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Closest Equal Element Queries
// Difficulty: Medium
// 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 solve_queries(nums: Vec<i32>, queries: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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