Problem 1182: Shortest Distance to Target Color

Problem Information

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

Problem Description

You are given an array

colors

, in which there are three colors:

1

,

2

and

3

.

You are also given some queries. Each query consists of two integers

i

and

c

, return the shortest distance between the given index

i

and the target color

c

. If there is no solution return

-1

.

Example 1:

Input:

colors = [1,1,2,1,3,2,2,3,3], queries = [[1,3],[2,2],[6,1]]

Output:

[3,0,3]

Explanation:

The nearest 3 from index 1 is at index 4 (3 steps away). The nearest 2 from index 2 is at index 2 itself (0 steps away). The nearest 1 from index 6 is at index 3 (3 steps away).

Example 2:

Input:

colors = [1,2], queries = [[0,3]]

Output:

[-1]

Explanation:

There is no 3 in the array.

Constraints:

1 <= colors.length <= 5*10^4

1 <= colors[i] <= 3

1 <= queries.length <= 5*10^4

queries[i].length == 2

0 <= queries[i][0] < colors.length

1 <= queries[i][1] <= 3

Code Snippets

C++:

```
class Solution {
public:
vector<int> shortestDistanceColor(vector<int>& colors, vector<vector<int>>&
queries) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function shortestDistanceColor(colors: number[], queries: number[][]):
number[] {

};
```

C#:

```
public class Solution {
public IList<int> ShortestDistanceColor(int[] colors, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* shortestDistanceColor(int* colors, int colorsSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func shortestDistanceColor(colors []int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun shortestDistanceColor(colors: IntArray, queries: Array<IntArray>):
List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} colors
# @param {Integer[][]} queries
# @return {Integer[]}
def shortest_distance_color(colors, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $colors
* @param Integer[][] $queries
* @return Integer[]
*/
function shortestDistanceColor($colors, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> shortestDistanceColor(List<int> colors, List<List<int>> queries) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec shortest_distance_color(colors :: [integer], queries :: [[integer]]) ::
[integer]
```

```
def shortest_distance_color(colors, queries) do

end
end
```

Erlang:

```
-spec shortest_distance_color(Colors :: [integer()], Queries ::
[[integer()]]) -> [integer()].
shortest_distance_color(Colors, Queries) ->
.
```

Racket:

```
(define/contract (shortest-distance-color colors queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Distance to Target Color
* Difficulty: Medium
* Tags: array, dp, search
*
* 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:
vector<int> shortestDistanceColor(vector<int>& colors, vector<vector<int>>&
queries) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Distance to Target Color
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 List<Integer> shortestDistanceColor(int[] colors, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Distance to Target Color
Difficulty: Medium
Tags: array, dp, search

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

Python Solution:

```
class Solution(object):
def shortestDistanceColor(self, colors, queries):
"""
:type colors: List[int]
```

```
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Shortest Distance to Target Color
* Difficulty: Medium
* Tags: array, dp, search
*
* 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[]} colors
* @param {number[][]} queries
* @return {number[]}
*/
var shortestDistanceColor = function(colors, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Distance to Target Color
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 shortestDistanceColor(colors: number[], queries: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Shortest Distance to Target Color
* Difficulty: Medium
* Tags: array, dp, search
*
* 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 IList<int> ShortestDistanceColor(int[] colors, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Distance to Target Color
* Difficulty: Medium
* Tags: array, dp, search
*
* 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
*/

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

}
```

Go Solution:

```
// Problem: Shortest Distance to Target Color
// Difficulty: Medium
```

```
// Tags: array, dp, search
//
// 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 shortestDistanceColor(colors []int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun shortestDistanceColor(colors: IntArray, queries: Array<IntArray>):
List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest Distance to Target Color
// Difficulty: Medium
// Tags: array, dp, search
//
// 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 shortest_distance_color(colors: Vec<i32>, queries: Vec<Vec<i32>>) ->
Vec<i32> {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer[]} colors
# @param {Integer[][]} queries
# @return {Integer[]}
def shortest_distance_color(colors, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $colors
* @param Integer[][] $queries
* @return Integer[]
*/
function shortestDistanceColor($colors, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> shortestDistanceColor(List<int> colors, List<List<int>> queries) {

}
}
```

Scala Solution:

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

}
```

```
}
```

Elixir Solution:

```
defmodule Solution do
@spec shortest_distance_color(colors :: [integer], queries :: [[integer]]) ::
[integer]
def shortest_distance_color(colors, queries) do

end
end
```

Erlang Solution:

```
-spec shortest_distance_color(Colors :: [integer()], Queries ::
[[integer()]]) -> [integer()].
shortest_distance_color(Colors, Queries) ->
.
```

Racket Solution:

```
(define/contract (shortest-distance-color colors queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```