Problem 2672: Number of Adjacent Elements With the Same Color

Problem Information

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

Problem Description

You are given an integer

n

representing an array

colors

of length

n

where all elements are set to 0's meaning

uncolored

. You are also given a 2D integer array

queries

where

queries[i] = [index

i

, color

i

]

. For the

i

th

query

:

Set

colors[index

i

]

to

color

i

.

Count the number of adjacent pairs in

colors

which have the same color (regardless of color

i

).

Return an array

answer

of the same length as

queries

where

answer[i]

is the answer to the

i

th

query.

Example 1:

Input:

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

Output:

[0,1,1,0,2]

Explanation:

Initially array colors = [0,0,0,0], where 0 denotes uncolored elements of the array.

After the 1

st

query colors = [2,0,0,0]. The count of adjacent pairs with the same color is 0.

After the 2

nd

query colors = [2,2,0,0]. The count of adjacent pairs with the same color is 1.

After the 3

rd

query colors = [2,2,0,1]. The count of adjacent pairs with the same color is 1.

After the 4

th

query colors = [2,1,0,1]. The count of adjacent pairs with the same color is 0.

After the 5

th

query colors = [2,1,1,1]. The count of adjacent pairs with the same color is 2.

Example 2:

Input:

n = 1, queries = [[0,100000]]

Output:

[0]

Explanation:

After the 1

st

query colors = [100000]. The count of adjacent pairs with the same color is 0.

Constraints:

1 <= n <= 10

5

1 <= queries.length <= 10

5

queries[i].length == 2

0 <= index

i

<= n - 1

1 <= color

i

<= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<int> colorTheArray(int n, vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] colorTheArray(int n, int[][] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function colorTheArray(n: number, queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] ColorTheArray(int n, int[][] queries) {

}
}
```

C:

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

}
```

Go:

```
func colorTheArray(n int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun colorTheArray(n: Int, queries: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func colorTheArray(_ n: Int, _ queries: [[Int]]) -> [Int] {
}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} queries
# @return {Integer[]}
def color_the_array(n, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $queries
* @return Integer[]
*/
function colorTheArray($n, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> colorTheArray(int n, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def colorTheArray(n: Int, queries: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec color_the_array(n :: integer, queries :: [[integer]]) :: [integer]
def color_the_array(n, queries) do

end
end
```

Erlang:

```
-spec color_the_array(N :: integer(), Queries :: [[integer()]]) ->
[integer()].
color_the_array(N, Queries) ->
.
```

Racket:

```
(define/contract (color-the-array n queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Adjacent Elements With the Same Color
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

class Solution {
public:
vector<int> colorTheArray(int n, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Adjacent Elements With the Same Color
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[] colorTheArray(int n, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Adjacent Elements With the Same Color
Difficulty: Medium
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Adjacent Elements With the Same Color
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @param {number[][]} queries
* @return {number[]}
*/
var colorTheArray = function(n, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Adjacent Elements With the Same Color
* Difficulty: Medium
* Tags: array
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function colorTheArray(n: number, queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Number of Adjacent Elements With the Same Color
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[] ColorTheArray(int n, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Number of Adjacent Elements With the Same Color
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

```
*/
int* colorTheArray(int n, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Number of Adjacent Elements With the Same Color
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func colorTheArray(n int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun colorTheArray(n: Int, queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Adjacent Elements With the Same Color
// Difficulty: Medium
// Tags: array
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn color_the_array(n: i32, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} queries
# @return {Integer[]}
def color_the_array(n, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $queries
* @return Integer[]
*/
function colorTheArray($n, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> colorTheArray(int n, List<List<int>> queries) {

}
}
```

Scala Solution:

```scala
object Solution {
def colorTheArray(n: Int, queries: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```elixir
defmodule Solution do
@spec color_the_array(n :: integer, queries :: [[integer]]) :: [integer]
def color_the_array(n, queries) do

end
end
```

Erlang Solution:

```erlang
-spec color_the_array(N :: integer(), Queries :: [[integer()]]) ->
[integer()].
color_the_array(N, Queries) ->
.
```

Racket Solution:

```racket
(define/contract (color-the-array n queries)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?))
)
```