Problem 2078: Two Furthest Houses With Different Colors

Problem Information

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

Problem Description

There are

n

houses evenly lined up on the street, and each house is beautifully painted. You are given a

0-indexed

integer array

colors

of length

n

, where

colors[i]

represents the color of the

i

th

house.

Return

the

maximum

distance between

two

houses with

different

colors

.

The distance between the

i

th

and

j

th

houses is

abs(i - j)

, where

abs(x)

is the

absolute value

of

x

.

Example 1:

Input:

colors = [

1

,1,1,

6

,1,1,1]

Output:

3

Explanation:

In the above image, color 1 is blue, and color 6 is red. The furthest two houses with different colors are house 0 and house 3. House 0 has color 1, and house 3 has color 6. The distance between them is abs(0 - 3) = 3. Note that houses 3 and 6 can also produce the optimal

answer.

Example 2:

Input:

colors = [

1

,8,3,8,

3

]

Output:

4

Explanation:

In the above image, color 1 is blue, color 8 is yellow, and color 3 is green. The furthest two houses with different colors are house 0 and house 4. House 0 has color 1, and house 4 has color 3. The distance between them is abs(0 - 4) = 4.

Example 3:

Input:

colors = [

0

,

1

]

Output:

1

Explanation:

The furthest two houses with different colors are house 0 and house 1. House 0 has color 0, and house 1 has color 1. The distance between them is abs(0 - 1) = 1.

Constraints:

n == colors.length

2 <= n <= 100

0 <= colors[i] <= 100

Test data are generated such that

at least

two houses have different colors.

Code Snippets

C++:

```
class Solution {
public:
int maxDistance(vector<int>& colors) {

}
};
```

Java:

```
class Solution {
public int maxDistance(int[] colors) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxDistance(colors: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxDistance(int[] colors) {
```

```
}
}
```

C:

```
int maxDistance(int* colors, int colorsSize) {

}
```

Go:

```
func maxDistance(colors []int) int {

}
```

Kotlin:

```
class Solution {
fun maxDistance(colors: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxDistance(_ colors: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_distance(colors: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} colors
# @return {Integer}
def max_distance(colors)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $colors
* @return Integer
*/
function maxDistance($colors) {

}
}
```

Dart:

```
class Solution {
int maxDistance(List<int> colors) {

}
}
```

Scala:

```
object Solution {
def maxDistance(colors: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_distance(colors :: [integer]) :: integer
def max_distance(colors) do

end
end
```

Erlang:

```
-spec max_distance(Colors :: [integer()]) -> integer().
max_distance(Colors) ->
.
```

Racket:

```
(define/contract (max-distance colors)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Two Furthest Houses With Different Colors
* Difficulty: Easy
* Tags: array, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int maxDistance(vector<int>& colors) {

}
};
```

Java Solution:

```
/**
* Problem: Two Furthest Houses With Different Colors
* Difficulty: Easy
* Tags: array, tree, greedy
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int maxDistance(int[] colors) {

}
}
```

Python3 Solution:

```
"""
Problem: Two Furthest Houses With Different Colors
Difficulty: Easy
Tags: array, tree, greedy

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def maxDistance(self, colors: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Two Furthest Houses With Different Colors
* Difficulty: Easy
```

```
* Tags: array, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[]} colors
* @return {number}
*/
var maxDistance = function(colors) {

};
```

TypeScript Solution:

```
/**
* Problem: Two Furthest Houses With Different Colors
* Difficulty: Easy
* Tags: array, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function maxDistance(colors: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Two Furthest Houses With Different Colors
* Difficulty: Easy
* Tags: array, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

public class Solution {
public int MaxDistance(int[] colors) {

}
}
```

C Solution:

```
/*
* Problem: Two Furthest Houses With Different Colors
* Difficulty: Easy
* Tags: array, tree, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int maxDistance(int* colors, int colorsSize) {

}
```

Go Solution:

```
// Problem: Two Furthest Houses With Different Colors
// Difficulty: Easy
// Tags: array, tree, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func maxDistance(colors []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxDistance(colors: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxDistance(_ colors: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Two Furthest Houses With Different Colors
// Difficulty: Easy
// Tags: array, tree, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn max_distance(colors: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} colors
# @return {Integer}
def max_distance(colors)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $colors
* @return Integer
*/
function maxDistance($colors) {

}
}
```

Dart Solution:

```
class Solution {
int maxDistance(List<int> colors) {

}
}
```

Scala Solution:

```
object Solution {
def maxDistance(colors: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_distance(colors :: [integer]) :: integer
def max_distance(colors) do

end
end
```

Erlang Solution:

```
-spec max_distance(Colors :: [integer()]) -> integer().
max_distance(Colors) ->
.
```

Racket Solution:

```
(define/contract (max-distance colors)
(-> (listof exact-integer?) exact-integer?)
)
```