Problem 3741: Minimum Distance Between Three Equal Elements II

Problem Information

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

Problem Description

You are given an integer array

nums

.

A tuple

(i, j, k)

of 3

distinct

indices is

good

if

nums[i] == nums[j] == nums[k]

.

The

distance

of a

good

tuple is

abs(i - j) + abs(j - k) + abs(k - i)

, where

abs(x)

denotes the

absolute value

of

x

.

Return an integer denoting the

minimum

possible

distance

of a

good

tuple. If no

good

tuples exist, return

-1

.

Example 1:

Input:

nums = [1,2,1,1,3]

Output:

6

Explanation:

The minimum distance is achieved by the good tuple

(0, 2, 3)

.

(0, 2, 3)

is a good tuple because

nums[0] == nums[2] == nums[3] == 1

. Its distance is

abs(0 - 2) + abs(2 - 3) + abs(3 - 0) = 2 + 1 + 3 = 6

.

Example 2:

Input:

nums = [1,1,2,3,2,1,2]

Output:

8

Explanation:

The minimum distance is achieved by the good tuple

(2, 4, 6)

.

(2, 4, 6)

is a good tuple because

nums[2] == nums[4] == nums[6] == 2

. Its distance is

abs(2 - 4) + abs(4 - 6) + abs(6 - 2) = 2 + 2 + 4 = 8

.

Example 3:

Input:

nums = [1]

Output:

-1

Explanation:

There are no good tuples. Therefore, the answer is -1.

Constraints:

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

5

1 <= nums[i] <= n

Code Snippets

C++:

```
class Solution {
public:
int minimumDistance(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int minimumDistance(int[] nums) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def minimumDistance(self, nums):
```

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

JavaScript:

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

};
```

TypeScript:

```
function minimumDistance(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MinimumDistance(int[] nums) {

}
}
```

C:

```
int minimumDistance(int* nums, int numsSize) {

}
```

Go:

```
func minimumDistance(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun minimumDistance(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minimumDistance(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_distance(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def minimum_distance(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function minimumDistance($nums) {

}
}
```

Dart:

```
class Solution {
int minimumDistance(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def minimumDistance(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_distance(nums :: [integer]) :: integer
def minimum_distance(nums) do

end
end
```

Erlang:

```
-spec minimum_distance(Nums :: [integer()]) -> integer().
minimum_distance(Nums) ->
.
```

Racket:

```
(define/contract (minimum-distance nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Distance Between Three Equal Elements II
* Difficulty: Medium
* Tags: array, hash
*
* 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 minimumDistance(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Distance Between Three Equal Elements II
* Difficulty: Medium
* Tags: array, hash
*
* 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 minimumDistance(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Distance Between Three Equal Elements II
Difficulty: Medium
Tags: array, hash
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Distance Between Three Equal Elements II
* Difficulty: Medium
* Tags: array, hash
*
* 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
* @return {number}
*/
var minimumDistance = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Distance Between Three Equal Elements II
* Difficulty: Medium
* Tags: array, hash
*
* 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 minimumDistance(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Distance Between Three Equal Elements II
* Difficulty: Medium
* Tags: array, hash
*
* 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 MinimumDistance(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Distance Between Three Equal Elements II
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

int minimumDistance(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Distance Between Three Equal Elements II
// Difficulty: Medium
// Tags: array, hash
//
// 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 minimumDistance(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumDistance(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumDistance(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Distance Between Three Equal Elements II
// Difficulty: Medium
// Tags: array, hash
```

```
//
// 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 minimum_distance(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def minimum_distance(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function minimumDistance($nums) {

}
}
```

Dart Solution:

```
class Solution {
int minimumDistance(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def minimumDistance(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_distance(nums :: [integer]) :: integer
def minimum_distance(nums) do

end
end
```

Erlang Solution:

```
-spec minimum_distance(Nums :: [integer()]) -> integer().
minimum_distance(Nums) ->
.
```

Racket Solution:

```
(define/contract (minimum-distance nums)
(-> (listof exact-integer?) exact-integer?)
)
```