Problem 3356: Zero Array Transformation II

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

and a 2D array

queries

where

queries[i] = [l

i

, r

i

, val

i

]

.

Each

queries[i]

represents the following action on

nums

:

Decrement the value at each index in the range

[l

i

, r

i

]

in

nums

by

at most

val

i

.

The amount by which each value is decremented

can be chosen

independently

for each index.

A

Zero Array

is an array with all its elements equal to 0.

Return the

minimum

possible

non-negative

value of

k

, such that after processing the first

k

queries in

sequence

,

nums

becomes a

Zero Array

. If no such

k

exists, return -1.

Example 1:

Input:

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

Output:

2

Explanation:

For i = 0 (l = 0, r = 2, val = 1):

Decrement values at indices

[0, 1, 2]

by

[1, 0, 1]

respectively.

The array will become

[1, 0, 1]

.

For i = 1 (l = 0, r = 2, val = 1):

Decrement values at indices

[0, 1, 2]

by

[1, 0, 1]

respectively.

The array will become

[0, 0, 0]

, which is a Zero Array. Therefore, the minimum value of

k

is 2.

Example 2:

Input:

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

Output:

-1

Explanation:

For i = 0 (l = 1, r = 3, val = 2):

Decrement values at indices

[1, 2, 3]

by

[2, 2, 1]

respectively.

The array will become

[4, 1, 0, 0]

.

For i = 1 (l = 0, r = 2, val

= 1):

Decrement values at indices

[0, 1, 2]

by

[1, 1, 0]

respectively.

The array will become

[3, 0, 0, 0]

, which is not a Zero Array.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 5 * 10

5

1 <= queries.length <= 10

5

queries[i].length == 3

0 <= l

i

<= r

i

< nums.length

1 <= val

i

<= 5

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minZeroArray(int[] nums, int[][] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MinZeroArray(int[] nums, int[][] queries) {
```

```
}
}
```

C:

```
int minZeroArray(int* nums, int numsSize, int** queries, int queriesSize,
int* queriesColSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun minZeroArray(nums: IntArray, queries: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (min-zero-array nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Zero Array Transformation II
* Difficulty: Medium
* Tags: array, search
*
* 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 minZeroArray(vector<int>& nums, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Zero Array Transformation II
* Difficulty: Medium
* Tags: array, search
*
* 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 minZeroArray(int[] nums, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Zero Array Transformation II
Difficulty: Medium
Tags: array, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Zero Array Transformation II
* Difficulty: Medium
* Tags: array, search
*
* 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[]} nums
* @param {number[][]} queries
* @return {number}
*/
var minZeroArray = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Zero Array Transformation II
* Difficulty: Medium
* Tags: array, search
*
* 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 minZeroArray(nums: number[], queries: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Zero Array Transformation II
* Difficulty: Medium
```

```
* Tags: array, search
*
* 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 MinZeroArray(int[] nums, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Zero Array Transformation II
* Difficulty: Medium
* Tags: array, search
*
* 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
*/

int minZeroArray(int* nums, int numsSize, int** queries, int queriesSize,
int* queriesColSize) {

}
```

Go Solution:

```
// Problem: Zero Array Transformation II
// Difficulty: Medium
// Tags: array, search
//
// 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 minZeroArray(nums []int, queries [][]int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun minZeroArray(nums: IntArray, queries: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Zero Array Transformation II
// Difficulty: Medium
// Tags: array, search
//
// 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 min_zero_array(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-zero-array nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```