Problem 3049: Earliest Second to Mark Indices II

Problem Information

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

Problem Description

You are given two

1-indexed

integer arrays,

nums

and,

changeIndices

, having lengths

n

and

m

, respectively.

Initially, all indices in

nums

are unmarked. Your task is to mark

all

indices in

nums

.

In each second,

s

, in order from

1

to

m

(

inclusive

), you can perform

one

of the following operations:

Choose an index

i

in the range

[1, n]

and

decrement

nums[i]

by

1

.

Set

nums[changeIndices[s]]

to any

non-negative

value.

Choose an index

i

in the range

[1, n]

, where

nums[i]

is

equal

to

0

, and

mark

index

i

.

Do nothing.

Return

an integer denoting the

earliest second

in the range

[1, m]

when

all

indices in

nums

can be marked by choosing operations optimally, or

-1

if it is impossible.

Example 1:

Input:

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

Output:

6

Explanation:

In this example, we have 7 seconds. The following operations can be performed to mark all indices: Second 1: Set nums[changeIndices[1]] to 0. nums becomes [0,2,3]. Second 2: Set nums[changeIndices[2]] to 0. nums becomes [0,2,0]. Second 3: Set nums[changeIndices[3]] to 0. nums becomes [0,0,0]. Second 4: Mark index 1, since nums[1] is equal to 0. Second 5: Mark index 2, since nums[2] is equal to 0. Second 6: Mark index 3, since nums[3] is equal to 0. Now all indices have been marked. It can be shown that it is not possible to mark all indices earlier than the 6th second. Hence, the answer is 6.

Example 2:

Input:

nums = [0,0,1,2], changeIndices = [1,2,1,2,1,2,1,2]

Output:

7

Explanation:

In this example, we have 8 seconds. The following operations can be performed to mark all indices: Second 1: Mark index 1, since nums[1] is equal to 0. Second 2: Mark index 2, since nums[2] is equal to 0. Second 3: Decrement index 4 by one. nums becomes [0,0,1,1]. Second 4: Decrement index 4 by one. nums becomes [0,0,1,0]. Second 5: Decrement index 3 by one. nums becomes [0,0,0,0]. Second 6: Mark index 3, since nums[3] is equal to 0. Second 7: Mark index 4, since nums[4] is equal to 0. Now all indices have been marked. It can be shown

that it is not possible to mark all indices earlier than the 7th second. Hence, the answer is 7.

Example 3:

Input:

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

Output:

-1

Explanation:

In this example, it can be shown that it is impossible to mark all indices, as we don't have enough seconds. Hence, the answer is -1.

Constraints:

1 <= n == nums.length <= 5000

0 <= nums[i] <= 10

9

1 <= m == changeIndices.length <= 5000

1 <= changeIndices[i] <= n

Code Snippets

C++:

```
class Solution {
public:
int earliestSecondToMarkIndices(vector<int>& nums, vector<int>&
changeIndices) {

}
```

```
};
```

Java:

```
class Solution {
public int earliestSecondToMarkIndices(int[] nums, int[] changeIndices) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function earliestSecondToMarkIndices(nums: number[], changeIndices:
number[]): number {
```

```
};
```

C#:

```
public class Solution {
public int EarliestSecondToMarkIndices(int[] nums, int[] changeIndices) {

}
}
```

C:

```
int earliestSecondToMarkIndices(int* nums, int numsSize, int* changeIndices,
int changeIndicesSize) {

}
```

Go:

```
func earliestSecondToMarkIndices(nums []int, changeIndices []int) int {

}
```

Kotlin:

```
class Solution {
fun earliestSecondToMarkIndices(nums: IntArray, changeIndices: IntArray): Int
{

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} change_indices
# @return {Integer}
def earliest_second_to_mark_indices(nums, change_indices)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $changeIndices
* @return Integer
*/
function earliestSecondToMarkIndices($nums, $changeIndices) {

}
}
```

Dart:

```
class Solution {
int earliestSecondToMarkIndices(List<int> nums, List<int> changeIndices) {

}
}
```

Scala:

```
object Solution {
def earliestSecondToMarkIndices(nums: Array[Int], changeIndices: Array[Int]):
Int = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec earliest_second_to_mark_indices(nums :: [integer], change_indices ::
[integer]) :: integer
def earliest_second_to_mark_indices(nums, change_indices) do

end
end
```

Erlang:

```
-spec earliest_second_to_mark_indices(Nums :: [integer()], ChangeIndices ::
[integer()]) -> integer().
earliest_second_to_mark_indices(Nums, ChangeIndices) ->
.
```

Racket:

```
(define/contract (earliest-second-to-mark-indices nums changeIndices)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Earliest Second to Mark Indices II
* Difficulty: Hard
* Tags: array, greedy, search, queue, heap
*
* 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 earliestSecondToMarkIndices(vector<int>& nums, vector<int>&
changeIndices) {

}
};
```

Java Solution:

```
/**
* Problem: Earliest Second to Mark Indices II
* Difficulty: Hard
* Tags: array, greedy, search, queue, heap
*
* 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 earliestSecondToMarkIndices(int[] nums, int[] changeIndices) {

}
}
```

Python3 Solution:

```
"""
Problem: Earliest Second to Mark Indices II
Difficulty: Hard
Tags: array, greedy, search, queue, heap

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 earliestSecondToMarkIndices(self, nums: List[int], changeIndices:
```

```
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Earliest Second to Mark Indices II
* Difficulty: Hard
* Tags: array, greedy, search, queue, heap
*
* 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[]} changeIndices
* @return {number}
*/
var earliestSecondToMarkIndices = function(nums, changeIndices) {

};
```

TypeScript Solution:

```
/**
* Problem: Earliest Second to Mark Indices II
* Difficulty: Hard
* Tags: array, greedy, search, queue, heap
*
```

```
* 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 earliestSecondToMarkIndices(nums: number[], changeIndices:
number[]): number {

};
```

C# Solution:

```
/*
* Problem: Earliest Second to Mark Indices II
* Difficulty: Hard
* Tags: array, greedy, search, queue, heap
*
* 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 EarliestSecondToMarkIndices(int[] nums, int[] changeIndices) {

}
}
```

C Solution:

```
/*
* Problem: Earliest Second to Mark Indices II
* Difficulty: Hard
* Tags: array, greedy, search, queue, heap
*
* 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 earliestSecondToMarkIndices(int* nums, int numsSize, int* changeIndices,
```

```
int changeIndicesSize) {

}
```

Go Solution:

```
// Problem: Earliest Second to Mark Indices II
// Difficulty: Hard
// Tags: array, greedy, search, queue, heap
//
// 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 earliestSecondToMarkIndices(nums []int, changeIndices []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun earliestSecondToMarkIndices(nums: IntArray, changeIndices: IntArray): Int
{

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Earliest Second to Mark Indices II
// Difficulty: Hard
// Tags: array, greedy, search, queue, heap
//
```

```
// 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 earliest_second_to_mark_indices(nums: Vec<i32>, change_indices:
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} change_indices
# @return {Integer}
def earliest_second_to_mark_indices(nums, change_indices)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $changeIndices
* @return Integer
*/
function earliestSecondToMarkIndices($nums, $changeIndices) {

}
}
```

Dart Solution:

```
class Solution {
int earliestSecondToMarkIndices(List<int> nums, List<int> changeIndices) {

}
}
```

Scala Solution:

```
object Solution {
def earliestSecondToMarkIndices(nums: Array[Int], changeIndices: Array[Int]):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec earliest_second_to_mark_indices(nums :: [integer], change_indices ::
[integer]) :: integer
def earliest_second_to_mark_indices(nums, change_indices) do

end
end
```

Erlang Solution:

```
-spec earliest_second_to_mark_indices(Nums :: [integer()], ChangeIndices ::
[integer()]) -> integer().
earliest_second_to_mark_indices(Nums, ChangeIndices) ->
.
```

Racket Solution:

```
(define/contract (earliest-second-to-mark-indices nums changeIndices)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```