Problem 3525: Find X Value of Array II

Problem Information

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

Problem Description

You are given an array of

positive

integers

nums

and a

positive

integer

k

. You are also given a 2D array

queries

, where

queries[i] = [index

i

, value

i

, start

i

, x

i

]

.

You are allowed to perform an operation

once

on

nums

, where you can remove any

suffix

from

nums

such that

nums

remains

non-empty

.

The

x-value

of

nums

for a given

x

is defined as the number of ways to perform this operation so that the

product

of the remaining elements leaves a

remainder

of

x

modulo

k

.

For each query in

queries

you need to determine the

x-value

of

nums

for

x

i

after performing the following actions:

Update

nums[index

i

]

to

value

i

. Only this step persists for the rest of the queries.

Remove

the prefix

nums[0..(start

i

- 1)]

(where

nums[0..(-1)]

will be used to represent the

empty

prefix).

Return an array

result

of size

queries.length

where

result[i]

is the answer for the

i

th

query.

A

prefix

of an array is a

subarray

that starts from the beginning of the array and extends to any point within it.

A

suffix

of an array is a

subarray

that starts at any point within the array and extends to the end of the array.

Note

that the prefix and suffix to be chosen for the operation can be

empty

.

Note

that x-value has a

different

definition in this version.

Example 1:

Input:

nums = [1,2,3,4,5], k = 3, queries = [[2,2,0,2],[3,3,3,0],[0,1,0,1]]

Output:

[2,2,2]

Explanation:

For query 0,

nums

becomes

[1, 2, 2, 4, 5]

, and the empty prefix

must

be removed. The possible operations are:

Remove the suffix

[2, 4, 5]

.

nums

becomes

[1, 2]

.

Remove the empty suffix.

nums

becomes

[1, 2, 2, 4, 5]

with a product 80, which gives remainder 2 when divided by 3.

For query 1,

nums

becomes

[1, 2, 2, 3, 5]

, and the prefix

[1, 2, 2]

must

be removed. The possible operations are:

Remove the empty suffix.

nums

becomes

[3, 5]

.

Remove the suffix

[5]

.

nums

becomes

[3]

.

For query 2,

nums

becomes

[1, 2, 2, 3, 5]

, and the empty prefix

must

be removed. The possible operations are:

Remove the suffix

[2, 2, 3, 5]

.

nums

becomes

[1]

.

Remove the suffix

[3, 5]

.

nums

becomes

[1, 2, 2]

.

Example 2:

Input:

nums = [1,2,4,8,16,32], k = 4, queries = [[0,2,0,2],[0,2,0,1]]

Output:

[1,0]

Explanation:

For query 0,

nums

becomes

[2, 2, 4, 8, 16, 32]

. The only possible operation is:

Remove the suffix

[2, 4, 8, 16, 32]

.

For query 1,

nums

becomes

[2, 2, 4, 8, 16, 32]

. There is no possible way to perform the operation.

Example 3:

Input:

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

Output:

[5]

Constraints:

1 <= nums[i] <= 10

9

1 <= nums.length <= 10

5

1 <= k <= 5

1 <= queries.length <= 2 * 10

4

queries[i] == [index

i

, value

i

, start

i

, x

i

]

0 <= index

i

<= nums.length - 1

1 <= value

i

<= 10

9

0 <= start

i

<= nums.length - 1

0 <= x

i

<= k - 1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

```
func resultArray(nums []int, k int, queries [][]int) []int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (result-array nums k queries)
(-> (listof exact-integer?) exact-integer? (listof (listof exact-integer?))
(listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find X Value of Array II
* Difficulty: Hard
* Tags: array, tree, math
*
* 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:
vector<int> resultArray(vector<int>& nums, int k, vector<vector<int>>&
queries) {

}
};
```

Java Solution:

```
/**
* Problem: Find X Value of Array II
* Difficulty: Hard
* Tags: array, tree, math
*
* 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[] resultArray(int[] nums, int k, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Find X Value of Array II
Difficulty: Hard
```

```
Tags: array, tree, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find X Value of Array II
* Difficulty: Hard
* Tags: array, tree, math
*
* 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[]} nums
* @param {number} k
* @param {number[][]} queries
* @return {number[]}
```

```
*/
var resultArray = function(nums, k, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Find X Value of Array II
* Difficulty: Hard
* Tags: array, tree, math
*
* 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 resultArray(nums: number[], k: number, queries: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Find X Value of Array II
* Difficulty: Hard
* Tags: array, tree, math
*
* 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[] ResultArray(int[] nums, int k, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Find X Value of Array II
* Difficulty: Hard
* Tags: array, tree, math
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* resultArray(int* nums, int numsSize, int k, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find X Value of Array II
// Difficulty: Hard
// Tags: array, tree, math
//
// 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 resultArray(nums []int, k int, queries [][]int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find X Value of Array II
// Difficulty: Hard
// Tags: array, tree, math
//
// 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 result_array(nums: Vec<i32>, k: i32, queries: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (result-array nums k queries)
(-> (listof exact-integer?) exact-integer? (listof (listof exact-integer?))
(listof exact-integer?))
)
```