Problem 3569: Maximize Count of Distinct Primes After Split

Problem Information

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

Problem Description

You are given an integer array

nums

having length

n

and a 2D integer array

queries

where

queries[i] = [idx, val]

.

For each query:

Update

nums[idx] = val

.

Choose an integer

k

with

1 <= k < n

to split the array into the non-empty prefix

nums[0..k-1]

and suffix

nums[k..n-1]

such that the sum of the counts of

distinct

prime

values in each part is

maximum

.

Note:

The changes made to the array in one query persist into the next query.

Return an array containing the result for each query, in the order they are given.

Example 1:

Input:

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

Output:

[3,4]

Explanation:

Initially

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

.

After 1

st

query,

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

. Split

nums

into

[2]

and

[2, 3, 1, 2]

.

[2]

consists of 1 distinct prime and

[2, 3, 1, 2]

consists of 2 distinct primes. Hence, the answer for this query is

1 + 2 = 3

.

After 2

nd

query,

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

. Split

nums

into

[2, 2, 3]

and

[3, 2]

with an answer of

2 + 2 = 4

.

The output is

[3, 4]

.

Example 2:

Input:

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

Output:

[0]

Explanation:

Initially

nums = [2, 1, 4]

.

After 1

st

query,

nums = [1, 1, 4]

. There are no prime numbers in

nums

, hence the answer for this query is 0.

The output is

[0]

.

Constraints:

2 <= n == nums.length <= 5 * 10

4

1 <= queries.length <= 5 * 10

4

1 <= nums[i] <= 10

5

0 <= queries[i][0] < nums.length

1 <= queries[i][1] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
```

```
def maximum_count(nums, queries)

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
```

```
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-count nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Count of Distinct Primes After Split
* 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> maximumCount(vector<int>& nums, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Count of Distinct Primes After Split
* 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[] maximumCount(int[] nums, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Count of Distinct Primes After Split
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 maximumCount(self, nums: List[int], queries: List[List[int]]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize Count of Distinct Primes After Split
* 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[][]} queries
* @return {number[]}
*/
var maximumCount = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Count of Distinct Primes After Split
* 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 maximumCount(nums: number[], queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximize Count of Distinct Primes After Split
* 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[] MaximumCount(int[] nums, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Count of Distinct Primes After Split
* 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* maximumCount(int* nums, int numsSize, int** queries, int queriesSize,
int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Maximize Count of Distinct Primes After Split
// 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 maximumCount(nums []int, queries [][]int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Count of Distinct Primes After Split
// 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 maximum_count(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
def maximum_count(nums, queries) do

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-count nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```