Problem 3526: Range XOR Queries with Subarray Reversals

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

and a 2D integer array

queries

of length

q

, where each query is one of the following three types:

Update

:

queries[i] = [1, index, value]

Set

nums[index] = value

.

Range XOR Query

:

queries[i] = [2, left, right]

Compute the bitwise XOR of all elements in the

subarray

nums[left...right]

, and record this result.

Reverse Subarray

:

queries[i] = [3, left, right]

Reverse the

subarray

nums[left...right]

in place.

Return

an array of the results of all range XOR queries

in the order they were encountered.

Example 1:

Input:

nums = [1,2,3,4,5], queries = [[2,1,3],[1,2,10],[3,0,4],[2,0,4]]

Output:

[5,8]

Explanation:

Query

1

:

[2, 1, 3]

– Compute XOR of subarray

[2, 3, 4]

resulting in 5.

Query 2:

[1, 2, 10]

– Update

nums[2]

to 10, updating the array to

[1, 2, 10, 4, 5]

.

Query 3:

[3, 0, 4]

– Reverse the entire array to get

[5, 4, 10, 2, 1]

.

Query 4:

[2, 0, 4]

– Compute XOR of subarray

[5, 4, 10, 2, 1]

resulting in 8.

Example 2:

Input:

nums = [7,8,9], queries = [[1,0,3],[2,0,2],[3,1,2]]

Output:

[2]

Explanation:

Query 1:

[1, 0, 3]

– Update

nums[0]

to 3, updating the array to

[3, 8, 9]

.

Query 2:

[2, 0, 2]

– Compute XOR of subarray

[3, 8, 9]

resulting in 2.

Query 3:

[3, 1, 2]

– Reverse the subarray

[8, 9]

to get

[9, 8]

.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

1 <= queries.length <= 10

5

queries[i].length == 3

queries[i][0] ∈ {1, 2, 3}

If

queries[i][0] == 1

:

0 <= index < nums.length

0 <= value <= 10

9

If

queries[i][0] == 2

or

queries[i][0] == 3

:

0 <= left <= right < nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def getResults(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 getResults = function(nums, queries) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (get-results nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Range XOR Queries with Subarray Reversals
* Difficulty: Hard
* Tags: array, tree
*
* 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> getResults(vector<int>& nums, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Range XOR Queries with Subarray Reversals
* Difficulty: Hard
* Tags: array, tree
*
* 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[] getResults(int[] nums, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Range XOR Queries with Subarray Reversals
Difficulty: Hard
Tags: array, tree

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Range XOR Queries with Subarray Reversals
* Difficulty: Hard
* Tags: array, tree
*
* 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 getResults = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Range XOR Queries with Subarray Reversals
* Difficulty: Hard
* Tags: array, tree
```

```
*
* 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 getResults(nums: number[], queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Range XOR Queries with Subarray Reversals
* Difficulty: Hard
* Tags: array, tree
*
* 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[] GetResults(int[] nums, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Range XOR Queries with Subarray Reversals
* Difficulty: Hard
* Tags: array, tree
*
* 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* getResults(int* nums, int numsSize, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Range XOR Queries with Subarray Reversals
// Difficulty: Hard
// Tags: array, tree
//
// 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 getResults(nums []int, queries [][]int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Range XOR Queries with Subarray Reversals
// Difficulty: Hard
```

```
// Tags: array, tree
//
// 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 get_results(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
```

```
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (get-results nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```