Problem 3152: Special Array II

Problem Information

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

Problem Description

An array is considered

special

if every pair of its adjacent elements contains two numbers with different parity.

You are given an array of integer

nums

and a 2D integer matrix

queries

, where for

queries[i] = [from

i

, to

i

]

your task is to check that

subarray

nums[from

i

..to

i

]

is

special

or not.

Return an array of booleans

answer

such that

answer[i]

is

true

if

nums[from

i

..to

i

]

is special.

Example 1:

Input:

nums = [3,4,1,2,6], queries = [[0,4]]

Output:

[false]

Explanation:

The subarray is

[3,4,1,2,6]

. 2 and 6 are both even.

Example 2:

Input:

nums = [4,3,1,6], queries = [[0,2],[2,3]]

Output:

[false,true]

Explanation:

The subarray is

[4,3,1]

. 3 and 1 are both odd. So the answer to this query is

false

.

The subarray is

[1,6]

. There is only one pair:

(1,6)

and it contains numbers with different parity. So the answer to this query is

true

.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

1 <= queries.length <= 10

5

queries[i].length == 2

0 <= queries[i][0] <= queries[i][1] <= nums.length - 1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

```
func isArraySpecial(nums []int, queries [][]int) []bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (is-array-special nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Special Array 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:
vector<bool> isArraySpecial(vector<int>& nums, vector<vector<int>>& queries)
{

}
};
```

Java Solution:

```
/**
* Problem: Special Array 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 boolean[] isArraySpecial(int[] nums, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Special Array 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 isArraySpecial(self, nums: List[int], queries: List[List[int]]) ->
List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Special Array 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 {boolean[]}
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Special Array 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 isArraySpecial(nums: number[], queries: number[][]): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Special Array 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 bool[] IsArraySpecial(int[] nums, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Special Array 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
*/

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

}
```

Go Solution:

```
// Problem: Special Array 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 isArraySpecial(nums []int, queries [][]int) []bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Special Array 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 is_array_special(nums: Vec<i32>, queries: Vec<Vec<i32>>) -> Vec<bool>
{

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Boolean[]
*/
function isArraySpecial($nums, $queries) {
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (is-array-special nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
boolean?))
)
```