Problem 2683: Neighboring Bitwise XOR

Problem Information

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

Problem Description

A

0-indexed

array

derived

with length

n

is derived by computing the

bitwise XOR

(⊕) of adjacent values in a

binary array

original

of length

n

.

Specifically, for each index

i

in the range

$[0, n - 1]$

:

If

$i = n - 1$

, then

$derived[i] = original[i] \oplus original[0]$

.

Otherwise,

$derived[i] = original[i] \oplus original[i + 1]$

.

Given an array

derived

, your task is to determine whether there exists a

valid binary array

original

that could have formed

derived

.

Return

true

if such an array exists or

false

otherwise.

A binary array is an array containing only

0's

and

1's

Example 1:

Input:

derived = [1,1,0]

Output:

true

Explanation:

A valid original array that gives derived is [0,1,0]. derived[0] = original[0] ⊕ original[1] = 0 ⊕ 1 = 1 derived[1] = original[1] ⊕ original[2] = 1 ⊕ 0 = 1 derived[2] = original[2] ⊕ original[0] = 0 ⊕ 0 = 0

Example 2:

Input:

derived = [1,1]

Output:

true

Explanation:

A valid original array that gives derived is [0,1]. derived[0] = original[0] ⊕ original[1] = 1 derived[1] = original[1] ⊕ original[0] = 1

Example 3:

Input:

derived = [1,0]

Output:

false

Explanation:

There is no valid original array that gives derived.

Constraints:

n == derived.length

1 <= n <= 10

5

The values in

derived

are either

0's

or

1's

Code Snippets

C++:

```
class Solution {
public:
bool doesValidArrayExist(vector<int>& derived) {

}
};
```

Java:

```
class Solution {
public boolean doesValidArrayExist(int[] derived) {

}
}
```

Python3:

```
class Solution:
def doesValidArrayExist(self, derived: List[int]) -> bool:
```

Python:

```
class Solution(object):
def doesValidArrayExist(self, derived):
"""
:type derived: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} derived
* @return {boolean}
*/
var doesValidArrayExist = function(derived) {

};
```

TypeScript:

```
function doesValidArrayExist(derived: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool DoesValidArrayExist(int[] derived) {

}
}
```

C:

```
bool doesValidArrayExist(int* derived, int derivedSize) {

}
```

Go:

```
func doesValidArrayExist(derived []int) bool {

}
```

Kotlin:

```
class Solution {
fun doesValidArrayExist(derived: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func doesValidArrayExist(_ derived: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn does_valid_array_exist(derived: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} derived
# @return {Boolean}
def does_valid_array_exist(derived)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $derived
* @return Boolean
*/
function doesValidArrayExist($derived) {

}
}
```

Dart:

```
class Solution {
bool doesValidArrayExist(List<int> derived) {

}
```

```
}
```

Scala:

```
object Solution {
def doesValidArrayExist(derived: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec does_valid_array_exist(derived :: [integer]) :: boolean
def does_valid_array_exist(derived) do

end
end
```

Erlang:

```
-spec does_valid_array_exist(Derived :: [integer()]) -> boolean().
does_valid_array_exist(Derived) ->
.
```

Racket:

```
(define/contract (does-valid-array-exist derived)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Neighboring Bitwise XOR
* Difficulty: Medium
* Tags: array
*
```

```
* 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:
bool doesValidArrayExist(vector<int>& derived) {

}
};
```

Java Solution:

```
/**
* Problem: Neighboring Bitwise XOR
* Difficulty: Medium
* Tags: array
*
* 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 doesValidArrayExist(int[] derived) {

}
}
```

Python3 Solution:

```
"""
Problem: Neighboring Bitwise XOR
Difficulty: Medium
Tags: array

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 doesValidArrayExist(self, derived: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def doesValidArrayExist(self, derived):
"""
:type derived: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Neighboring Bitwise XOR
* Difficulty: Medium
* Tags: array
*
* 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[]} derived
* @return {boolean}
*/
var doesValidArrayExist = function(derived) {

};
```

TypeScript Solution:

```
/**
* Problem: Neighboring Bitwise XOR
* Difficulty: Medium
* Tags: array
```

```
*
* 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 doesValidArrayExist(derived: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Neighboring Bitwise XOR
* Difficulty: Medium
* Tags: array
*
* 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 DoesValidArrayExist(int[] derived) {

}
}
```

C Solution:

```
/*
* Problem: Neighboring Bitwise XOR
* Difficulty: Medium
* Tags: array
*
* 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
*/

bool doesValidArrayExist(int* derived, int derivedSize) {
```

```
}
```

Go Solution:

```
// Problem: Neighboring Bitwise XOR
// Difficulty: Medium
// Tags: array
//
// 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 doesValidArrayExist(derived []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun doesValidArrayExist(derived: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func doesValidArrayExist(_ derived: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Neighboring Bitwise XOR
// Difficulty: Medium
// Tags: array
//
// 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 does_valid_array_exist(derived: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} derived
# @return {Boolean}
def does_valid_array_exist(derived)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $derived
* @return Boolean
*/
function doesValidArrayExist($derived) {

}
}
```

Dart Solution:

```
class Solution {
bool doesValidArrayExist(List<int> derived) {

}
}
```

Scala Solution:

```
object Solution {
def doesValidArrayExist(derived: Array[Int]): Boolean = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec does_valid_array_exist(derived :: [integer]) :: boolean
def does_valid_array_exist(derived) do

end
end
```

Erlang Solution:

```
-spec does_valid_array_exist(Derived :: [integer()]) -> boolean().
does_valid_array_exist(Derived) ->
.
```

Racket Solution:

```
(define/contract (does-valid-array-exist derived)
(-> (listof exact-integer?) boolean?)
)
```