Problem 457: Circular Array Loop

Problem Information

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

Problem Description

You are playing a game involving a

circular

array of non-zero integers

nums

. Each

nums[i]

denotes the number of indices forward/backward you must move if you are located at index

i

:

If

nums[i]

is positive, move

nums[i]

steps

forward

, and

If

nums[i]

is negative, move

abs(nums[i])

steps

backward

.

Since the array is

circular

, you may assume that moving forward from the last element puts you on the first element, and moving backwards from the first element puts you on the last element.

A

cycle

in the array consists of a sequence of indices

seq

of length

k

where:

Following the movement rules above results in the repeating index sequence

seq[0] -> seq[1] -> ... -> seq[k - 1] -> seq[0] -> ...

Every

nums[seq[j]]

is either

all positive

or

all negative

.

k > 1

Return

true

if there is a

cycle

in

nums

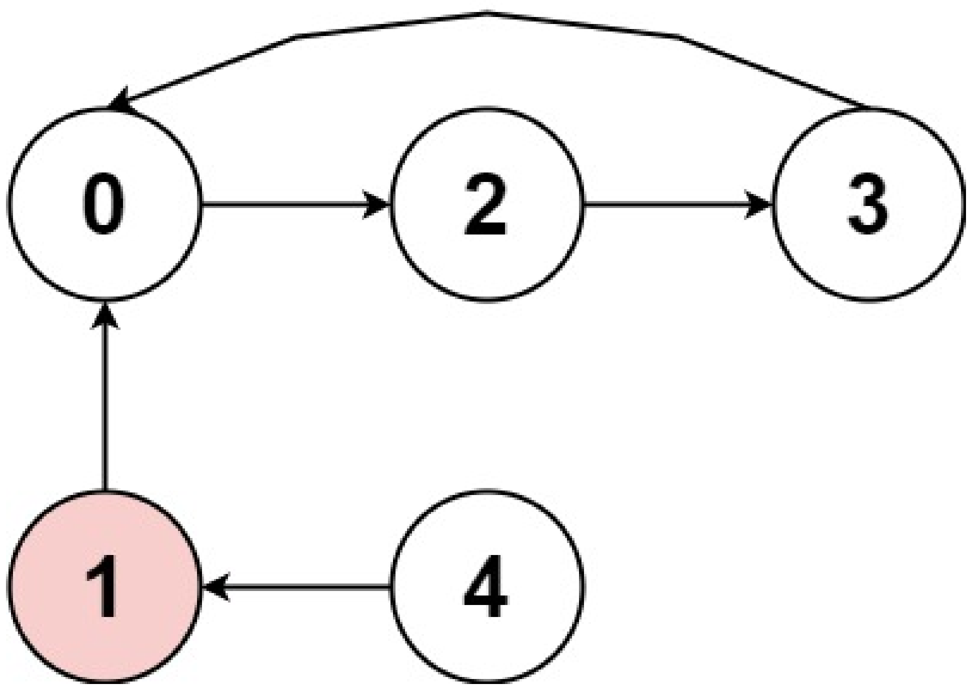
, or

false

otherwise

.

Example 1:



Input:

nums = [2,-1,1,2,2]

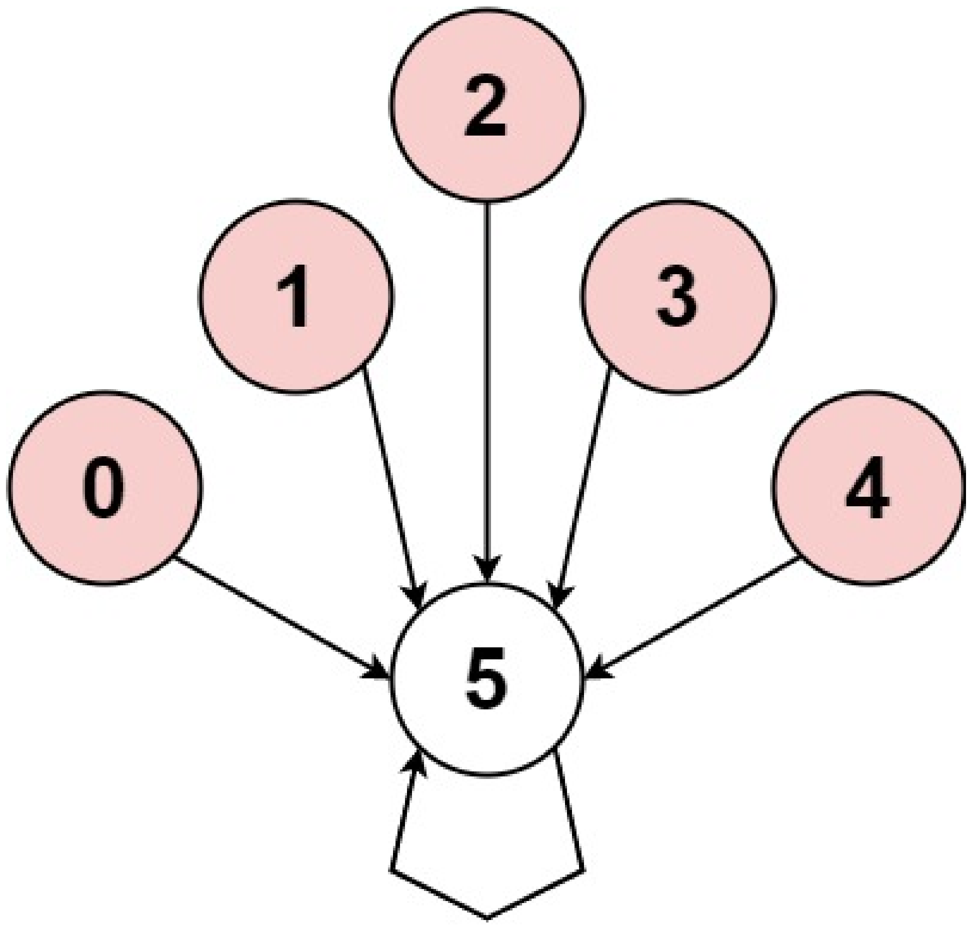
Output:

true

Explanation:

The graph shows how the indices are connected. White nodes are jumping forward, while red is jumping backward. We can see the cycle 0 --> 2 --> 3 --> 0 --> ..., and all of its nodes are white (jumping in the same direction).

Example 2:



Input:

nums = [-1,-2,-3,-4,-5,6]

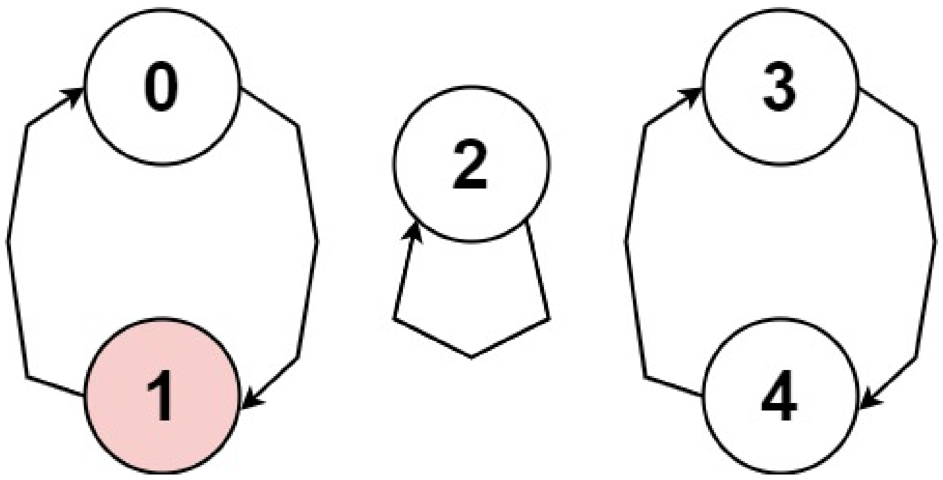
Output:

false

Explanation:

The graph shows how the indices are connected. White nodes are jumping forward, while red is jumping backward. The only cycle is of size 1, so we return false.

Example 3:



Input:

nums = [1,-1,5,1,4]

Output:

true

Explanation:

The graph shows how the indices are connected. White nodes are jumping forward, while red is jumping backward. We can see the cycle 0 --> 1 --> 0 --> ..., and while it is of size > 1, it has a node jumping forward and a node jumping backward, so

it is not a cycle

. We can see the cycle 3 --> 4 --> 3 --> ..., and all of its nodes are white (jumping in the same direction).

Constraints:

1 <= nums.length <= 5000

-1000 <= nums[i] <= 1000

nums[i] != 0

Follow up:

Could you solve it in

O(n)

time complexity and

O(1)

extra space complexity?

Code Snippets

C++:

```
class Solution {
public:
bool circularArrayLoop(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public boolean circularArrayLoop(int[] nums) {

}
}
```

Python3:

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

Python:

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

```
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

```
function circularArrayLoop(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool CircularArrayLoop(int[] nums) {

}
}
```

C:

```
bool circularArrayLoop(int* nums, int numsSize) {

}
```

Go:

```
func circularArrayLoop(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun circularArrayLoop(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Boolean}
def circular_array_loop(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean
*/
function circularArrayLoop($nums) {

}
}
```

Dart:

```
class Solution {
bool circularArrayLoop(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def circularArrayLoop(nums: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec circular_array_loop(nums :: [integer]) :: boolean
def circular_array_loop(nums) do

end
end
```

Erlang:

```
-spec circular_array_loop(Nums :: [integer()]) -> boolean().
circular_array_loop(Nums) ->
.
```

Racket:

```
(define/contract (circular-array-loop nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Circular Array Loop
* Difficulty: Medium
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool circularArrayLoop(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Circular Array Loop
* Difficulty: Medium
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean circularArrayLoop(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Circular Array Loop
Difficulty: Medium
Tags: array, graph, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def circularArrayLoop(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Circular Array Loop
* Difficulty: Medium
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @return {boolean}
*/
var circularArrayLoop = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Circular Array Loop
* Difficulty: Medium
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function circularArrayLoop(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Circular Array Loop
* Difficulty: Medium
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool CircularArrayLoop(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Circular Array Loop
* Difficulty: Medium
* Tags: array, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

bool circularArrayLoop(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Circular Array Loop
// Difficulty: Medium
// Tags: array, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func circularArrayLoop(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun circularArrayLoop(nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Circular Array Loop
// Difficulty: Medium
// Tags: array, graph, hash
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn circular_array_loop(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Boolean}
def circular_array_loop(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Boolean
*/
function circularArrayLoop($nums) {

}
}
```

Dart Solution:

```
class Solution {
bool circularArrayLoop(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def circularArrayLoop(nums: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec circular_array_loop(nums :: [integer]) :: boolean
def circular_array_loop(nums) do

end
end
```

Erlang Solution:

```
-spec circular_array_loop(Nums :: [integer()]) -> boolean().
circular_array_loop(Nums) ->
.
```

Racket Solution:

```
(define/contract (circular-array-loop nums)
(-> (listof exact-integer?) boolean?)
)
```