Problem 2360: Longest Cycle in a Graph

Problem Information

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

Problem Description

You are given a

directed

graph of

n

nodes numbered from

0

to

n - 1

, where each node has

at most one

outgoing edge.

The graph is represented with a given

0-indexed

array

edges

of size

n

, indicating that there is a directed edge from node

i

to node

edges[i]

. If there is no outgoing edge from node

i

, then

edges[i] == -1

.

Return

the length of the

longest

cycle in the graph

. If no cycle exists, return

-1

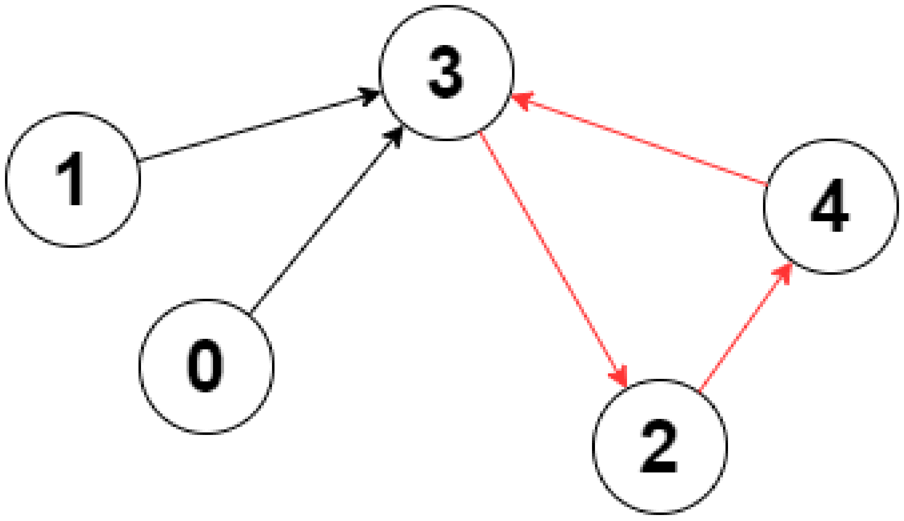
.

A cycle is a path that starts and ends at the

same

node.

Example 1:



Input:

edges = [3,3,4,2,3]

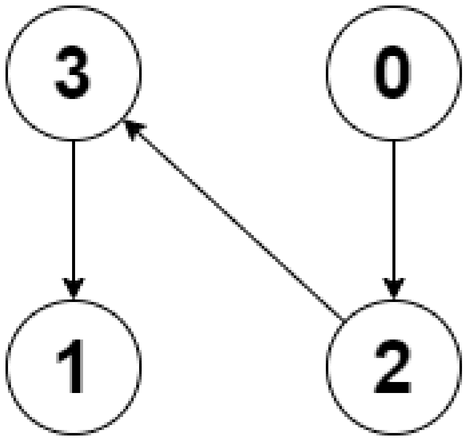
Output:

3

Explanation:

The longest cycle in the graph is the cycle: 2 -> 4 -> 3 -> 2. The length of this cycle is 3, so 3 is returned.

Example 2:



Input:

edges = [2,-1,3,1]

Output:

-1

Explanation:

There are no cycles in this graph.

Constraints:

n == edges.length

2 <= n <= 10

5

-1 <= edges[i] < n

edges[i] != i

Code Snippets

C++:

```
class Solution {
public:
int longestCycle(vector<int>& edges) {

}
};
```

Java:

```
class Solution {
public int longestCycle(int[] edges) {

}
}
```

Python3:

```
class Solution:
def longestCycle(self, edges: List[int]) -> int:
```

Python:

```
class Solution(object):
def longestCycle(self, edges):
"""
:type edges: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} edges
* @return {number}
*/
var longestCycle = function(edges) {

};
```

TypeScript:

```
function longestCycle(edges: number[]): number {
};
```

C#:

```
public class Solution {
public int LongestCycle(int[] edges) {

}
}
```

C:

```
int longestCycle(int* edges, int edgesSize) {

}
```

Go:

```
func longestCycle(edges []int) int {

}
```

Kotlin:

```
class Solution {
fun longestCycle(edges: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func longestCycle(_ edges: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn longest_cycle(edges: Vec<i32>) -> i32 {
}
}
```

Ruby:

```
# @param {Integer[]} edges
# @return {Integer}
def longest_cycle(edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $edges
* @return Integer
*/
function longestCycle($edges) {

}
}
```

Dart:

```
class Solution {
int longestCycle(List<int> edges) {

}
}
```

Scala:

```
object Solution {
def longestCycle(edges: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_cycle(edges :: [integer]) :: integer
def longest_cycle(edges) do

end
end
```

Erlang:

```
-spec longest_cycle(Edges :: [integer()]) -> integer().
longest_cycle(Edges) ->
.
```

Racket:

```
(define/contract (longest-cycle edges)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Cycle in a Graph
* Difficulty: Hard
* Tags: array, graph, sort, 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:
int longestCycle(vector<int>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Cycle in a Graph
* Difficulty: Hard
* Tags: array, graph, sort, 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 int longestCycle(int[] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Cycle in a Graph
Difficulty: Hard
Tags: array, graph, sort, 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 longestCycle(self, edges: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def longestCycle(self, edges):
"""
:type edges: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Cycle in a Graph
* Difficulty: Hard
* Tags: array, graph, sort, 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[]} edges
* @return {number}
*/
var longestCycle = function(edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Cycle in a Graph
* Difficulty: Hard
* Tags: array, graph, sort, 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 longestCycle(edges: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Longest Cycle in a Graph
* Difficulty: Hard
* Tags: array, graph, sort, 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 int LongestCycle(int[] edges) {

}
}
```

C Solution:

```
/*
* Problem: Longest Cycle in a Graph
* Difficulty: Hard
* Tags: array, graph, sort, 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
*/

int longestCycle(int* edges, int edgesSize) {

}
```

Go Solution:

```
// Problem: Longest Cycle in a Graph
// Difficulty: Hard
// Tags: array, graph, sort, 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 longestCycle(edges []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestCycle(edges: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func longestCycle(_ edges: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Cycle in a Graph
// Difficulty: Hard
// Tags: array, graph, sort, 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 longest_cycle(edges: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} edges
# @return {Integer}
def longest_cycle(edges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $edges
* @return Integer
*/
function longestCycle($edges) {

}
}
```

Dart Solution:

```
class Solution {
int longestCycle(List<int> edges) {

}
}
```

Scala Solution:

```
object Solution {
def longestCycle(edges: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_cycle(edges :: [integer]) :: integer
def longest_cycle(edges) do

end
end
```

Erlang Solution:

```
-spec longest_cycle(Edges :: [integer()]) -> integer().
longest_cycle(Edges) ->
.
```

Racket Solution:

```
(define/contract (longest-cycle edges)
(-> (listof exact-integer?) exact-integer?)
)
```