Problem 2924: Find Champion II

Problem Information

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

Problem Description

There are

n

teams numbered from

0

to

n - 1

in a tournament; each team is also a node in a

DAG

.

You are given the integer

n

and a

0-indexed

2D integer array

edges

of length

m

representing the

DAG

, where

edges[i] = [u

i

, v

i

]

indicates that there is a directed edge from team

u

i

to team

v

i

in the graph.

A directed edge from

a

to

b

in the graph means that team

a

is

stronger

than team

b

and team

b

is

weaker

than team

a

.

Team

a

will be the

champion

of the tournament if there is no team

b

that is

stronger

than team

a

.

Return

the team that will be the

champion

of the tournament if there is a

unique

champion, otherwise, return

-1

.

Notes

A

cycle

is a series of nodes

a

1

, a

2

, ..., a

n

, a

n+1

such that node

a

1

is the same node as node

a

n+1

, the nodes

a

1

, a

2

, ..., a

n

are distinct, and there is a directed edge from the node

a

i

to node

a

i+1

for every

i

in the range

[1, n]

.

A

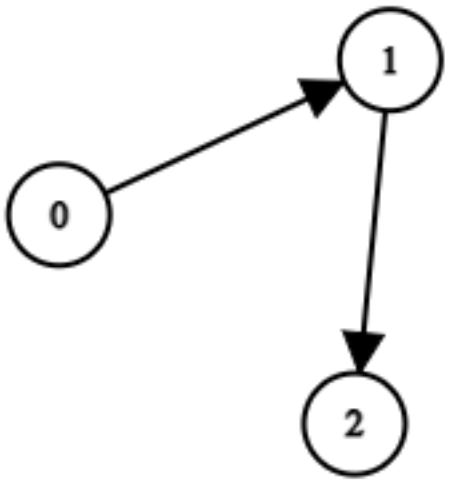
DAG

is a directed graph that does not have any

cycle

.

Example 1:



Input:

n = 3, edges = [[0,1],[1,2]]

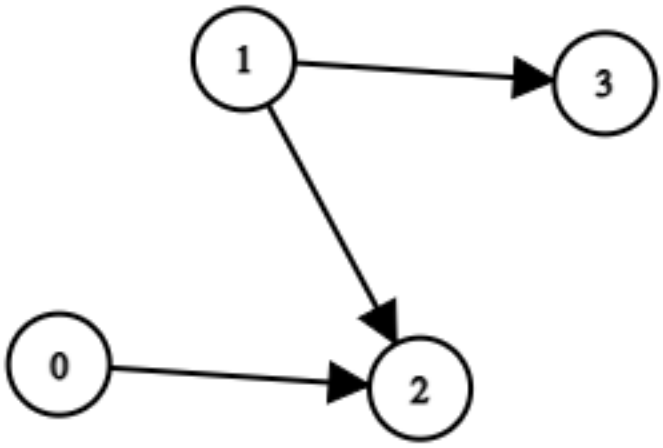
Output:

0

Explanation:

Team 1 is weaker than team 0. Team 2 is weaker than team 1. So the champion is team 0.

Example 2:



Input:

n = 4, edges = [[0,2],[1,3],[1,2]]

Output:

-1

Explanation:

Team 2 is weaker than team 0 and team 1. Team 3 is weaker than team 1. But team 1 and team 0 are not weaker than any other teams. So the answer is -1.

Constraints:

1 <= n <= 100

m == edges.length

0 <= m <= n * (n - 1) / 2

edges[i].length == 2

0 <= edge[i][j] <= n - 1

edges[i][0] != edges[i][1]

The input is generated such that if team

a

is stronger than team

b

, team

b

is not stronger than team

a

.

The input is generated such that if team

a

is stronger than team

b

and team

b

is stronger than team

c

, then team

a

is stronger than team

c

.

Code Snippets

C++:

```
class Solution {
public:
int findChampion(int n, vector<vector<int>>& edges) {

}
};
```

Java:

```
class Solution {
public int findChampion(int n, int[][] edges) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def findChampion(self, n, edges):
"""
:type n: int
```

```
:type edges: List[List[int]]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function findChampion(n: number, edges: number[][]): number {

};
```

C#:

```
public class Solution {
public int FindChampion(int n, int[][] edges) {

}
}
```

C:

```
int findChampion(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go:

```
func findChampion(n int, edges [][]int) int {

}
```

Kotlin:

```
class Solution {
fun findChampion(n: Int, edges: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func findChampion(_ n: Int, _ edges: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_champion(n: i32, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer}
def find_champion(n, edges)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Integer
*/
function findChampion($n, $edges) {

}
```

```
}
```

Dart:

```
class Solution {
int findChampion(int n, List<List<int>> edges) {

}
}
```

Scala:

```
object Solution {
def findChampion(n: Int, edges: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_champion(n :: integer, edges :: [[integer]]) :: integer
def find_champion(n, edges) do

end
end
```

Erlang:

```
-spec find_champion(N :: integer(), Edges :: [[integer()]]) -> integer().
find_champion(N, Edges) ->
.
```

Racket:

```
(define/contract (find-champion n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Champion II
* Difficulty: Medium
* Tags: array, graph
*
* 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 findChampion(int n, vector<vector<int>>& edges) {

}
};
```

Java Solution:

```
/**
* Problem: Find Champion II
* Difficulty: Medium
* Tags: array, graph
*
* 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 findChampion(int n, int[][] edges) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Champion II
Difficulty: Medium
Tags: array, graph
```

```
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 findChampion(self, n: int, edges: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Champion II
* Difficulty: Medium
* Tags: array, graph
*
* 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} n
* @param {number[][]} edges
* @return {number}
*/
var findChampion = function(n, edges) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Champion II
* Difficulty: Medium
* Tags: array, graph
*
* 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 findChampion(n: number, edges: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Find Champion II
* Difficulty: Medium
* Tags: array, graph
*
* 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 FindChampion(int n, int[][] edges) {

}
}
```

C Solution:

```
/*
* Problem: Find Champion II
* Difficulty: Medium
* Tags: array, graph
*
* 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 findChampion(int n, int** edges, int edgesSize, int* edgesColSize) {

}
```

Go Solution:

```
// Problem: Find Champion II
// Difficulty: Medium
// Tags: array, graph
//
// 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 findChampion(n int, edges [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findChampion(n: Int, edges: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func findChampion(_ n: Int, _ edges: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find Champion II
// Difficulty: Medium
// Tags: array, graph
//
// 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 find_champion(n: i32, edges: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @return {Integer}
def find_champion(n, edges)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @return Integer
*/
function findChampion($n, $edges) {

}
}
```

Dart Solution:

```
class Solution {
int findChampion(int n, List<List<int>> edges) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def findChampion(n: Int, edges: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_champion(n :: integer, edges :: [[integer]]) :: integer
def find_champion(n, edges) do

end
end
```

Erlang Solution:

```
-spec find_champion(N :: integer(), Edges :: [[integer()]]) -> integer().
find_champion(N, Edges) ->
.
```

Racket Solution:

```
(define/contract (find-champion n edges)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```