Problem 277: Find the Celebrity

Problem Information

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

Problem Description

Suppose you are at a party with

n

people labeled from

0

to

n - 1

and among them, there may exist one celebrity. The definition of a celebrity is that all the other

n - 1

people know the celebrity, but the celebrity does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. You are only allowed to ask questions like: "Hi, A. Do you know B?" to get information about whether A knows B. You need to find out the celebrity (or verify there is not one) by asking as few questions as possible (in the asymptotic sense).

You are given an integer

n

and a helper function

bool knows(a, b)

that tells you whether

a

knows

b

. Implement a function

int findCelebrity(n)

. There will be exactly one celebrity if they are at the party.

Return

the celebrity's label if there is a celebrity at the party

. If there is no celebrity, return

-1

.

Note

that the

n x n

2D array

graph

given as input is

not

directly available to you, and instead

only

accessible through the helper function

knows

.

graph[i][j] == 1

represents person

i

knows person

j

, wherease

graph[i][j] == 0

represents person

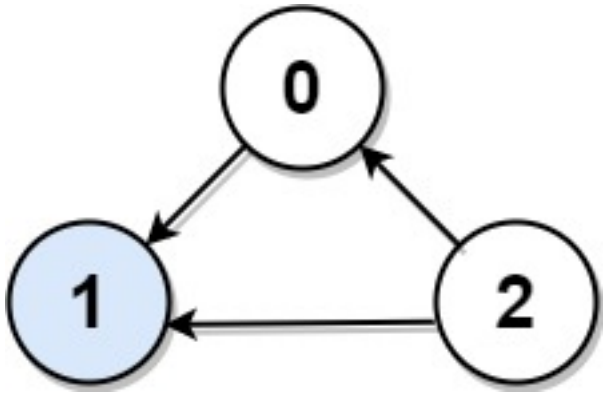
j

does not know person

i

.

Example 1:



Input:

graph = [[1,1,0],[0,1,0],[1,1,1]]

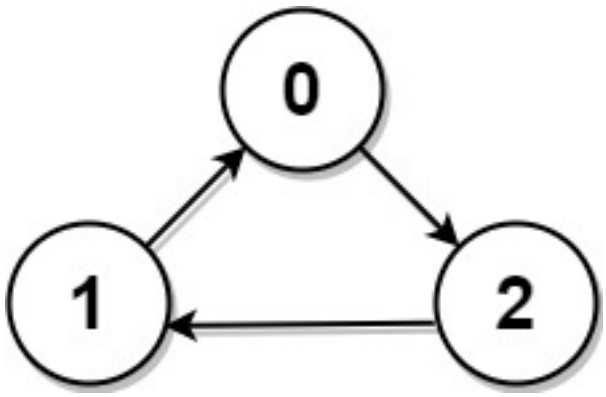
Output:

1

Explanation:

There are three persons labeled with 0, 1 and 2. graph[i][j] = 1 means person i knows person j, otherwise graph[i][j] = 0 means person i does not know person j. The celebrity is the person labeled as 1 because both 0 and 2 know him but 1 does not know anybody.

Example 2:



Input:

graph = [[1,0,1],[1,1,0],[0,1,1]]

Output:

-1

Explanation:

There is no celebrity.

Constraints:

n == graph.length == graph[i].length

2 <= n <= 100

graph[i][j]

is

0

or

1

.

graph[i][i] == 1

Follow up:

If the maximum number of allowed calls to the API

knows

is

3 * n

, could you find a solution without exceeding the maximum number of calls?

Code Snippets

C++:

```
/* The knows API is defined for you.
bool knows(int a, int b); */

class Solution {
public:
int findCelebrity(int n) {

}
};
```

Java:

```
/* The knows API is defined in the parent class Relation.
boolean knows(int a, int b); */

public class Solution extends Relation {
public int findCelebrity(int n) {

}
}
```

Python3:

```
# The knows API is already defined for you.
# return a bool, whether a knows b
# def knows(a: int, b: int) -> bool:

class Solution:
def findCelebrity(self, n: int) -> int:
```

Python:

```
# The knows API is already defined for you.
# @param a, person a
# @param b, person b
```

```
# @return a boolean, whether a knows b
# def knows(a, b):

class Solution(object):
def findCelebrity(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* Definition for knows()
*
* @param {integer} person a
* @param {integer} person b
* @return {boolean} whether a knows b
* knows = function(a, b) {
* ...
* };
*/

/**
* @param {function} knows()
* @return {function}
*/
var solution = function(knows) {
/**
* @param {integer} n Total people
* @return {integer} The celebrity
*/
return function(n) {

};
};
```

TypeScript:

```
/**
* The knows API is defined in the parent class Relation.
* knows(a: number, b: number): boolean {
```

```
* ...
* };
*/

var solution = function(knows: any) {

return function(n: number): number {

};
};
```

C#:

```
/* The Knows API is defined in the parent class Relation.
bool Knows(int a, int b); */

public class Solution : Relation {
public int FindCelebrity(int n) {

}
}
```

C:

```
/* The knows API is defined for you.
bool knows(int a, int b); */

int findCelebrity(int n) {

}
```

Go:

```
/**
* The knows API is already defined for you.
* knows := func(a int, b int) bool
*/
func solution(knows func(a int, b int) bool) func(n int) int {
return func(n int) int {

}
}
```

Kotlin:

```
/* The knows API is defined in the parent class Relation.
fun knows(a: Int, b: Int) : Boolean {} */

class Solution: Relation() {
override fun findCelebrity(n: Int) : Int {

}
}
```

Swift:

```
/**
* The knows API is defined in the parent class Relation.
* func knows(_ a: Int, _ b: Int) -> Bool;
*/

class Solution : Relation {
func findCelebrity(_ n: Int) -> Int {

}
}
```

Rust:

```
/* The knows API is defined for you.
knows(a: i32, b: i32)->bool;
to call it use self.knows(a,b)
*/

impl Solution {
pub fn find_celebrity(&self, n: i32) -> i32 {

}
}
```

Ruby:

```
# The knows API is already defined for you.
# @param {Integer} person a
# @param {Integer} person b
# @return {Boolean} whether a knows b
```

```
# def knows(a, b)

# @param {Integer} n
# @return {Integer}
def find_celebrity(n)

end
```

PHP:

```
/* The knows API is defined in the parent class Relation.
public function knows($a, $b){} */

class Solution extends Relation {
/**
* @param Integer $n
* @return Integer
*/
function findCelebrity($n) {

}
}
```

Scala:

```
/* The knows API is defined in the parent class Relation.
def knows(a: Int, b: Int): Boolean = {} */

class Solution extends Relation {
def findCelebrity(n: Int): Int = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Celebrity
```

```
* 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
*/

/* The knows API is defined for you.
bool knows(int a, int b); */

class Solution {
public:
int findCelebrity(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Celebrity
* 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
*/

/* The knows API is defined in the parent class Relation.
boolean knows(int a, int b); */

public class Solution extends Relation {
public int findCelebrity(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Celebrity
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
"""

# The knows API is already defined for you.
# return a bool, whether a knows b
# def knows(a: int, b: int) -> bool:

class Solution:
def findCelebrity(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# The knows API is already defined for you.
# @param a, person a
# @param b, person b
# @return a boolean, whether a knows b
# def knows(a, b):

class Solution(object):
def findCelebrity(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Celebrity
* 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
*/

/**
* Definition for knows()
*
* @param {integer} person a
* @param {integer} person b
* @return {boolean} whether a knows b
* knows = function(a, b) {
* ...
* };
*/

/**
* @param {function} knows()
* @return {function}
*/
var solution = function(knows) {
/**
* @param {integer} n Total people
* @return {integer} The celebrity
*/
return function(n) {

};
};
```

TypeScript Solution:

```
/**
* Problem: Find the Celebrity
* 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
*/
```

```
/**
* The knows API is defined in the parent class Relation.
* knows(a: number, b: number): boolean {
* ...
* };
*/

var solution = function(knows: any) {

return function(n: number): number {

};
};
```

C# Solution:

```
/*
* Problem: Find the Celebrity
* 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
*/

/* The Knows API is defined in the parent class Relation.
bool Knows(int a, int b); */

public class Solution : Relation {
public int FindCelebrity(int n) {

}
}
```

C Solution:

```
/*
* Problem: Find the Celebrity
* 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
*/

/* The knows API is defined for you.
bool knows(int a, int b); */

int findCelebrity(int n) {

}
```

Go Solution:

```
// Problem: Find the Celebrity
// 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

/**
* The knows API is already defined for you.
* knows := func(a int, b int) bool
*/
func solution(knows func(a int, b int) bool) func(n int) int {
return func(n int) int {

}
}
```

Kotlin Solution:

```
/* The knows API is defined in the parent class Relation.
fun knows(a: Int, b: Int) : Boolean {} */

class Solution: Relation() {
override fun findCelebrity(n: Int) : Int {
```

```
}
}
```

Swift Solution:

```
/**
* The knows API is defined in the parent class Relation.
* func knows(_ a: Int, _ b: Int) -> Bool;
*/

class Solution : Relation {
func findCelebrity(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Celebrity
// 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

/* The knows API is defined for you.
knows(a: i32, b: i32)->bool;
to call it use self.knows(a,b)
*/

impl Solution {
pub fn find_celebrity(&self, n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# The knows API is already defined for you.
# @param {Integer} person a
```

```
# @param {Integer} person b
# @return {Boolean} whether a knows b
# def knows(a, b)

# @param {Integer} n
# @return {Integer}
def find_celebrity(n)

end
```

PHP Solution:

```
/* The knows API is defined in the parent class Relation.
public function knows($a, $b){} */

class Solution extends Relation {
/**
* @param Integer $n
* @return Integer
*/
function findCelebrity($n) {

}
}
```

Scala Solution:

```
/* The knows API is defined in the parent class Relation.
def knows(a: Int, b: Int): Boolean = {} */

class Solution extends Relation {
def findCelebrity(n: Int): Int = {

}
}
```