Problem 997: Find the Town Judge

Problem Information

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

Problem Description

In a town, there are

n

people labeled from

1

to

n

. There is a rumor that one of these people is secretly the town judge.

If the town judge exists, then:

The town judge trusts nobody.

Everybody (except for the town judge) trusts the town judge.

There is exactly one person that satisfies properties

1

and

2

.

You are given an array

trust

where

trust[i] = [a

i

, b

i

]

representing that the person labeled

a

i

trusts the person labeled

b

i

. If a trust relationship does not exist in

trust

array, then such a trust relationship does not exist.

Return

the label of the town judge if the town judge exists and can be identified, or return

-1

otherwise

.

Example 1:

Input:

n = 2, trust = [[1,2]]

Output:

2

Example 2:

Input:

n = 3, trust = [[1,3],[2,3]]

Output:

3

Example 3:

Input:

n = 3, trust = [[1,3],[2,3],[3,1]]

Output:

-1

Constraints:

1 <= n <= 1000

0 <= trust.length <= 10

4

trust[i].length == 2

All the pairs of

trust

are

unique

.

a

i

!= b

i

1 <= a

i

, b

i

<= n

Code Snippets

C++:

```
class Solution {
public:
int findJudge(int n, vector<vector<int>>& trust) {

}
};
```

Java:

```
class Solution {
public int findJudge(int n, int[][] trust) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function findJudge(n: number, trust: number[][]): number {

};
```

C#:

```
public class Solution {
public int FindJudge(int n, int[][] trust) {

}
}
```

C:

```
int findJudge(int n, int** trust, int trustSize, int* trustColSize) {

}
```

Go:

```
func findJudge(n int, trust [][]int) int {

}
```

Kotlin:

```
class Solution {
fun findJudge(n: Int, trust: Array<IntArray>): Int {

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} trust
# @return {Integer}
def find_judge(n, trust)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $trust
* @return Integer
*/
function findJudge($n, $trust) {

}
}
```

Dart:

```
class Solution {
int findJudge(int n, List<List<int>> trust) {

}
}
```

Scala:

```
object Solution {
def findJudge(n: Int, trust: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_judge(n :: integer, trust :: [[integer]]) :: integer
def find_judge(n, trust) do

end
end
```

Erlang:

```
-spec find_judge(N :: integer(), Trust :: [[integer()]]) -> integer().
find_judge(N, Trust) ->
.
```

Racket:

```
(define/contract (find-judge n trust)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Town Judge
* Difficulty: Easy
* 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:
int findJudge(int n, vector<vector<int>>& trust) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Town Judge
* Difficulty: Easy
* 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 int findJudge(int n, int[][] trust) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Town Judge
Difficulty: Easy
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 findJudge(self, n: int, trust: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Town Judge
* Difficulty: Easy
* 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} n
* @param {number[][]} trust
* @return {number}
*/
var findJudge = function(n, trust) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Town Judge
* Difficulty: Easy
* 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 findJudge(n: number, trust: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Town Judge
* Difficulty: Easy
* 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 int FindJudge(int n, int[][] trust) {

}
}
```

C Solution:

```
/*
* Problem: Find the Town Judge
* Difficulty: Easy
* 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
*/

int findJudge(int n, int** trust, int trustSize, int* trustColSize) {

}
```

Go Solution:

```
// Problem: Find the Town Judge
// Difficulty: Easy
// 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 findJudge(n int, trust [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findJudge(n: Int, trust: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Town Judge
// Difficulty: Easy
// 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 find_judge(n: i32, trust: Vec<Vec<i32>>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} trust
# @return {Integer}
def find_judge(n, trust)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $trust
* @return Integer
*/
function findJudge($n, $trust) {

}
}
```

Dart Solution:

```
class Solution {
int findJudge(int n, List<List<int>> trust) {

}
}
```

Scala Solution:

```
object Solution {
def findJudge(n: Int, trust: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_judge(n :: integer, trust :: [[integer]]) :: integer
def find_judge(n, trust) do

end
end
```

Erlang Solution:

```
-spec find_judge(N :: integer(), Trust :: [[integer()]]) -> integer().
find_judge(N, Trust) ->
.
```

Racket Solution:

```
(define/contract (find-judge n trust)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```