Description

277. Find the Celebrity

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 him/her but he/she does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. The only thing you are allowed to do is to ask questions like: "Hi, A. Do you know B?" to get information of 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 a helper function bool knows(a, b) which tells you whether A knows B. Implement a function int findCelebrity(n), your function should minimize the number of calls to knows.

Note: There will be exactly one celebrity if he/she is in the party. Return the celebrity's label if there is a celebrity in the party. If there is no celebrity, return -1.

Idea

The primitive idea is brute force. A nested loop to enumerate all possible pairs, which takes O(n^2).

Java

```java
public class Solution extends Relation {
    // O(n^2) brute force AC
    public int findCelebrity(int n) {
        int[] hash = new int[n];
        Arrays.fill(hash, 1);
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (knows(i, j)) {
                    hash[i]--;
                    hash[j]++;
                }
                if (knows(j, i)) {
                    hash[j]--;
                    hash[i]++;
                }
            }
        }

        for (int i = 0; i < n; i++) {
            if (hash[i] == n) {
                return i;
            }
        }
        return -1;
    }
}
```

There are a lot of duplications when using brute force. So for any i and j, if knows(i,j) is true, we can get two information: 1. i is not celebrity; 2. j may be celebrity. Similarly, if knows(i,j) is false, we can get two information: 1. i may be celebrity; 2. j is not celebrity. So we can have two chasing pointers, celebrity and i. If knows(celebrity, i) is true, we can assign i as the new candidate. Otherwise we increase i. After O(n) check, we have to check if our candidate doens't know anyone, and everyone knows our candidate.

Java

```
public class Solution extends Relation {
    // O(n) AC
    public int findCelebrity(int n) {
        int celebrity = 0;
        for (int i = 1; i < n; i++) {
            if (knows(celebrity, i)) {
                celebrity = i;
            }
        }

        for (int i = 0; i < n; i++) {
            if (i != celebrity
                && (knows(celebrity, i) || !knows(i, celebrity))) {
                return -1;
            }
        }

        return celebrity;
    }
}
```

C++

```
bool knows(int a, int b);

class Solution {
public:
    int findCelebrity(int n) {
        int celebrity = 0;
        for (int i = 1; i < n; i++) {
            if (knows(celebrity, i)) {
                celebrity = i;
            }
        }
        for (int i = 0; i < n; i++) {
            if (celebrity != i
                && ((knows(celebrity, i) || !knows(i, celebrity)))) {
                return -1;
            }
        }
        return celebrity;
    }
}
```

Summary

- Two pointers. Chasing pointers.