277. Find the Celebrity

Two Pointers Medium Interactive **Graph**

Problem Description

n - 1. Your goal is to identify a possible celebrity among them. A celebrity is defined by two conditions: first, every other person at the party knows the celebrity, and second, the celebrity does not know anyone else at the party. Your tool is a function knows (a, b), which checks if person a knows person b. You need to use this function to determine who,

The given problem presents a scenario where you're at a party with n people, each labeled with a unique number between 0 and

if anyone, is the celebrity at the party. It is important to minimize the number of calls to the knows function, since your objective is to find the celebrity efficiently.

The output of your task is to return the unique label of the celebrity if one exists, or -1 if there is no celebrity present.

Intuition

1. Every person other than the celebrity must know the celebrity, which means if knows (a, b) is true, a cannot be the celebrity.

2. The celebrity does not know anyone else, which implies if knows (a, b) is false, b cannot be the celebrity.

candidate is the celebrity. Otherwise, there is no celebrity at the party.

anyone who was known by ans has been eliminated from contention.

To solve this problem, we take advantage of the unique characteristics of the celebrity:

Based on these points, we can iterate through the list of people at the party and use the knows function to determine potential

candidates for being the celebrity. One way to find the celebrity is to assume the first person (labeled 0) is the celebrity and then

assumption is wrong, and the other person becomes the new candidate for the celebrity. This process continues until we finish checking everybody. After we find a candidate through the first iteration, we need to verify two conditions to confirm the candidate is indeed the celebrity:

check this assumption against every other person using knows. If the assumed celebrity knows another person, then the

• Every person knows the candidate. If there is any person who does not know the candidate, then the candidate cannot be the celebrity. The iteration for verification ensures that these conditions are met. If both conditions are satisfied for the candidate, then the

The solution capitalizes on these ideas, using the intuition that knowing a person instantly disqualifies one from being a celebrity

• The candidate does not know any person. If the candidate knows any person, then the candidate cannot be the celebrity.

and not being known by at least one person also leads to a disqualification. Solution Approach

The implementation provided is a straightforward two-pass solution that uses a simple elimination approach to find the celebrity.

In the first for loop, we start with an assumption that person 0 might be the celebrity. We iterate over the range from 1 to n.

need to pass:

definition of a celebrity.

For each person i, we check if the current candidate (ans) knows person i using the knows (ans, i) function.

• If knows (ans, i) is true, it means that ans cannot be the celebrity because a celebrity does not know anyone else. So we update the ans to

• If knows (ans, i) is false, we do nothing and move on. This is because ans still could be the celebrity, as not knowing i aligns with the

By the end of this loop, the variable ans holds the label of the only person who could potentially be the celebrity, because

First Pass - Identifying a Celebrity Candidate:

be i since i is now a possible celebrity candidate.

Second Pass - Validating the Candidate: Next, we perform a second for loop to verify that the candidate (held in ans) is indeed the celebrity. There are two checks we

person's label. If the candidate knows any person i, we return -1 as this disqualifies them from being the celebrity. • Every other person must know the candidate. We check this by ensuring not knows (i, ans) is false, meaning person i knows ans. If we find someone who doesn't know ans, we return -1 as this also disqualifies ans from being the celebrity.

their label. If either condition fails for any person, we return -1 to indicate that there is no celebrity.

If both conditions are satisfied for all other people at the party, then we can safely conclude that ans is the celebrity and return

There is no need for any additional data structure as we can keep track of the potential celebrity with a single variable ans. This

• The candidate should not know any other person. This is checked by if knows (ans, i), where ans is the candidate and i is any other

approach is efficient because it requires at most 2*(n - 1) calls to knows — once to eliminate non-celebrities and once again to confirm that the remaining candidate is indeed the celebrity.

described. First Pass - Identifying a Celebrity Candidate: ○ Initialize the candidate to person (ans = 0). We'll check if person (b) is the celebrity.

• Check if 0 knows 2, using knows (0, 2). This time it returns true, which means 0 cannot be the celebrity. Update the candidate to person

• Check if 0 knows 1, using knows (0, 1). Let's say it returns false, so we do nothing because 0 could still be the celebrity.

• Check if 2 knows 3, using knows (2, 3). It returns false, so we do nothing; 2 is still a potential celebrity.

Let's consider a small example with 4 people at the party, where the unique labels for them are 0, 1, 2, and 3. Assume that

person 3 is the celebrity according to the problem's definition. We want to identify this person using the solution approach

By the end of the first pass, our candidate is person 2.

passes the first check.

2 (ans = 2).

Example Walkthrough

Second Pass - Validating the Candidate: • We need to verify if 2 is truly the celebrity by doing two checks: does not know anyone else (knows(ans, i)) and is known by everyone else (knows(i, ans)). • Check if 2 knows 0, 1, or 3, using knows(2, 0), knows(2, 1), and knows(2, 3). They all should return false. Let's say they do, so 2

○ Next, we need to check if all others (0, 1, 3) know 2. We check knows (0, 2), knows (1, 2), and knows (3, 2). They all should return true

for 2 to be the celebrity. Let's say knows (0, 2) is true, knows (1, 2) is true, but knows (3, 2) is false because the celebrity (3) should

not know anyone else. Person 2 fails the second check because the celebrity (3) does not know them.

Person 1 knows 2, so 1 is not a celebrity.

def findCelebrity(self, n: int) -> int:

if knows(candidate, i):

candidate = i

if candidate != i:

boolean knows(int a, int b); */

public class Solution extends Relation {

Initialize the candidate for celebrity to 0

Iterate over the range from 1 to n-1

Verify candidate is indeed a celebrity

/* The knows API is defined in the parent class Relation.

* Determine the celebrity by knockout method

// This method is to find the celebrity within 'n' people.

// Initialize the candidate for celebrity to 0

// A celebrity is defined as somebody who everyone knows but who knows nobody themselves.

// If the candidate knows 'i', then 'candidate' can't be a celebrity.

// The second loop is to confirm whether the candidate is the celebrity.

// If all conditions are met, return the candidate as the celebrity.

// If these conditions are not met, return -1 indicating no celebrity.

if (candidate != i && (knows(candidate, i) || !knows(i, candidate))) {

// The candidate should not know any other person,

// and all other people should know the candidate.

// The first loop is to find a candidate who might be the celebrity.

We will need to continue the same check for other persons 0 and 1: Person 0 knows 2, so 0 is not a celebrity.

indicating no celebrity is present. However, in this case, we return 3 as the label of the celebrity at the party.

Finally, when it comes to person 3, the checks are as follows: ○ 3 does not know 0, 1, or 2, fulfilling the first condition of being a celebrity. All persons 0, 1, and 2 know 3, fulfilling the second condition.

By the end of the verification process, we determine that person 3 meets both conditions, and since no other person does,

we conclude that the label of the celebrity is 3. If the checks had failed for 3 as they did for 2, then we would return -1,

Solution Implementation

If the candidate knows person i, then switch candidate to i

Make sure we skip comparing the candidate with themselves

if knows(candidate, i) or not knows(i, candidate):

Candidate should not know anyone, and everyone should know the candidate

If the condition fails, return -1 because there is no celebrity

Python # The knows API is already defined for you. # Returns a boolean, indicating whether person a knows person b. # def knows(a: int, b: int) -> bool:

return -1 # If all checks pass, return the candidate as the celebrity return candidate

candidate = 0

for i in range(1, n):

for i in range(n):

class Solution:

Java

/**

class Solution {

int findCelebrity(int n) {

int candidate = 0;

for (int i = 1; i < n; ++i) {

candidate = i;

for (int i = 0; i < n; ++i) {

return -1;

return candidate;

// 'i' might be the celebrity.

if (knows(candidate, i)) {

public:

```
* @param n Number of people in the party
     * @return The index of the celebrity if exists, otherwise return -1
     */
    public int findCelebrity(int n) {
        // Initial assumption: the first person (0) may be the celebrity
        int candidate = 0;
        // First Pass: Find a candidate for celebrity
        for (int i = 1; i < n; ++i) {
            // If the candidate knows i, candidate cannot be a celebrity, thus i might be
            if (knows(candidate, i)) {
                candidate = i;
        // Second Pass: Verify the candidate is really a celebrity
        for (int i = 0; i < n; ++i) {
            // Skip if i is the same as the candidate
            if (candidate != i) {
                // If the candidate knows i, or i doesn't know the candidate,
                // then the candidate cannot be a celebrity
                if (knows(candidate, i) || !knows(i, candidate)) {
                    return -1;
        // If all checks pass, the candidate is a celebrity
        return candidate;
C++
/* The knows API is defined for you.
      bool knows(int a, int b); */
```

// TypeScript function representing the knows API // It should return true if person a knows person b, false otherwise. declare function knows(a: number, b: number): boolean;

TypeScript

};

```
// This function finds the celebrity within 'n' people.
// A celebrity is someone whom everyone knows but who knows no one themselves.
function findCelebrity(n: number): number {
    // Initialize the candidate for the celebrity to 0
    let candidate: number = 0;
    // The first loop is to find a candidate who might be the celebrity.
    for (let i = 1; i < n; i++) {
        // If the candidate knows 'i', then 'candidate' cannot be the celebrity.
        // In this case, 'i' might be the new candidate.
        if (knows(candidate, i)) {
            candidate = i;
    // The second loop is to confirm whether the candidate is the celebrity.
    for (let i = 0; i < n; i++) {
        // The candidate should not know any other person.
        // and all other people should know the candidate.
        // If these conditions are not met, return -1 indicating there is no celebrity.
        if (candidate !== i && (knows(candidate, i) || !knows(i, candidate))) {
            return -1;
    // If all conditions are met, return the candidate as the celebrity.
    return candidate;
# The knows API is already defined for you.
# Returns a boolean, indicating whether person a knows person b.
# def knows(a: int, b: int) -> bool:
class Solution:
    def findCelebrity(self, n: int) -> int:
        # Initialize the candidate for celebrity to 0
        candidate = 0
        # Iterate over the range from 1 to n-1
        for i in range(1, n):
            # If the candidate knows person i, then switch candidate to i
            if knows(candidate, i):
                candidate = i
```

If all checks pass, return the candidate as the celebrity return candidate Time and Space Complexity

Time Complexity

for i in range(n):

if candidate != i:

return -1

Verify candidate is indeed a celebrity

The given Python code is comprised of two separate for loops that are not nested. The first for loop runs from 1 to n-1 and involves a constant amount of work for each iteration, i.e., one call to the knows API

Make sure we skip comparing the candidate with themselves

if knows(candidate, i) or not knows(i, candidate):

Candidate should not know anyone, and everyone should know the candidate

If the condition fails, return -1 because there is no celebrity

function. This loop determines the candidate for the celebrity. The second for loop checks whether the found candidate is actually a celebrity by making sure that everyone knows the

candidate and the candidate knows no one (except themselves, which is not checked). This loop makes up to 2*(n-1) calls to the knows API. Therefore, the time complexity for the code is O(n) + O(n), which simplifies to O(n).

Space Complexity

The code uses a fixed amount of extra space (only variable ans is used to store the celebrity candidate). It does not allocate any space that is dependent on the input size n.

Thus, the space complexity is 0(1).