Interviewer A Packet - TIPS #6

# Interviewer:

## Behavioral:

Tell me about a time when your responsibilities got a little overwhelming. How did you handle it?

## Question:

<https://leetcode.com/problems/keys-and-rooms/>

There are n rooms labeled from 0 to n - 1 and all the rooms are locked except for room 0. Your goal is to visit all the rooms. However, you cannot enter a locked room without having its key.

When you visit a room, you may find a set of distinct keys in it. Each key has a number on it, denoting which room it unlocks, and you can take all of them with you to unlock the other rooms.

Given an array rooms where rooms[i] is the set of keys that you can obtain if you visited room i, return true *if you can visit all the rooms, or* false *otherwise*.

Examples:

Example 1:

Input: rooms = [[1],[2],[3],[]]

Output: true

Explanation:

We visit room 0 and pick up key 1.

We then visit room 1 and pick up key 2.

We then visit room 2 and pick up key 3.

We then visit room 3.

Since we were able to visit every room, we return true.

Example 2:

Input: rooms = [[1,3],[3,0,1],[2],[0]]

Output: false

Explanation: We can not enter room number 2 since the only key that unlocks it is in that room.

## Follow up Q&A:

Constraints:

* n == rooms.length
* 2 <= n <= 1000
* 0 <= rooms[i].length <= 1000
* 1 <= sum(rooms[i].length) <= 3000
* 0 <= rooms[i][j] < n
* All the values of rooms[i] are unique.

## Hint(s):

*Ask if they would like a hint before giving a hint*

## Solution(s): (General concept and time/space complexity)

#### Approach #1: Depth-First Search [Accepted] (Java)

Intuition and Algorithm

When visiting a room for the first time, look at all the keys in that room. For any key that hasn't been used yet, add it to the todo list (stack) for it to be used.

class Solution {

public boolean canVisitAllRooms(List<List<Integer>> rooms) {

boolean[] seen = new boolean[rooms.size()];

seen[0] = true;

Stack<Integer> stack = new Stack();

stack.push(0);

//At the beginning, we have a todo list "stack" of keys to use.

//'seen' represents at some point we have entered this room.

while (!stack.isEmpty()) { // While we have keys...

int node = stack.pop(); // Get the next key 'node'

for (int nei: rooms.get(node)) // For every key in room # 'node'...

if (!seen[nei]) { // ...that hasn't been used yet

seen[nei] = true; // mark that we've entered the room

stack.push(nei); // add the key to the todo list

}

}

for (boolean v: seen) // if any room hasn't been visited, return false

if (!v) return false;

return true;

}

}

Complexity Analysis

* Time Complexity:
  + O(N + E), where *N* is the number of rooms, and *E* is the total number of keys.
* Space Complexity:
  + *O*(*N*) in additional space complexity, to store stack and seen

### Other questions follow up

*Ask if there is more than 5 minutes remaining when they finish their code and testing.*

# Interviewee:

## Question: <https://leetcode.com/problems/find-the-town-judge/>

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:

1. The town judge trusts nobody.
2. Everybody (except for the town judge) trusts the town judge.
3. There is exactly one person that satisfies properties 1 and 2.

You are given an array trust where trust[i] = [ai, bi] representing that the person labeled ai trusts the person labeled bi.

Return *the label of the town judge if the town judge exists and can be identified, or return* -1 *otherwise*.

## Example(s):

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

Example 4:

* Input: n = 3, trust = [[1,2],[2,3]]
* Output: -1

Example 5:

* Input: n = 4, trust = [[1,3],[1,4],[2,3],[2,4],[4,3]]
* Output: 3

## Code below or on leetcode