**Approach #1: Depth-First Search [Accepted]**

**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.

See the comments of the code for more details.

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