#include <iostream>

#include <vector>

using namespace std;

// TreeNode structure

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

// EulerTour class to perform Euler Tour and answer queries

class EulerTour {

private:

vector<int> euler; // Stores the Euler Tour sequence

vector<int> level; // Stores the level of each node in the Euler Tour

vector<int> firstOccurrence; // Stores the first occurrence of each node in the Euler Tour

// Helper function to perform Euler Tour

void tour(TreeNode\* node, int depth, int& index) {

if (!node) return;

// Record first occurrence of the node

if (firstOccurrence[node->val] == -1) {

firstOccurrence[node->val] = index;

}

// Add node to Euler and Level arrays

euler.push\_back(node->val);

level.push\_back(depth);

index++;

// Recur for left subtree

tour(node->left, depth + 1, index);

if (node->left) {

euler.push\_back(node->val);

level.push\_back(depth);

index++;

}

// Recur for right subtree

tour(node->right, depth + 1, index);

if (node->right) {

euler.push\_back(node->val);

level.push\_back(depth);

index++;

}

}

public:

// Constructor to initialize Euler Tour

EulerTour(TreeNode\* root, int maxNodeValue) {

// Initialize firstOccurrence with -1 (indicating no occurrence yet)

firstOccurrence.resize(maxNodeValue + 1, -1);

int index = 0;

tour(root, 0, index);

}

// Function to find LCA of two nodes

int findLCA(int u, int v) {

**// To-do: Implement this function**

// Hint: Use the firstOccurrence array to find the indices of u and v

// Find the node with the minimum level in the range [uIdx, vIdx]

}

// Function to find the number of dependents (subtree size) of a node

int findDependents(int x) {

**// To-do: Implement this function**

// Hint: Count the number of occurrences of x in the Euler array

// Subtree size = (number of occurrences + 1) / 2

}

};

// Main function

int main() {

// Build the binary tree using linked structure

TreeNode\* root = new TreeNode(1);

root->left = new TreeNode(2);

root->right = new TreeNode(3);

root->left->left = new TreeNode(4);

root->left->right = new TreeNode(5);

root->right->right = new TreeNode(6);

// Perform Euler Tour

// Pass the maximum node value in the tree (6 in this case)

EulerTour eulerTour(root, 6);

// Find LCA of 4 and 5

int lca = eulerTour.findLCA(4, 5);

cout << "LCA of 4 and 5 is: " << lca << endl;

// Find dependents of 2

int dependents = eulerTour.findDependents(2);

cout << "Dependents of 2 is: " << dependents << endl;

return 0;

}