DSA Lab

Mr. ALEEM AHMAD

A logo of a university

Description automatically generated

Bahria University

**Lab # 8**

**Binary Tree Implementation**

LAB Journal

Asim Ali (01-131232-015)

**Lab 8: Binary Tree Implementation**

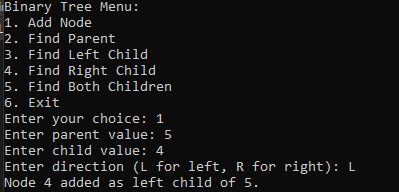
**TASK:**

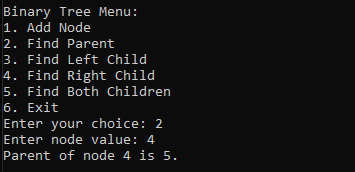
Binary Tree Implementation.

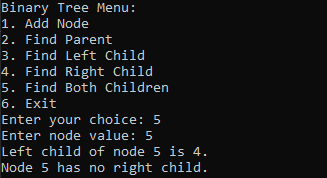
**Lab Task GitHub Link:**

[Link](https://github.com/iasimkhan2005/DSA.git)

**OUTPUT:**







**CODE:**

#include <iostream>

#include <unordered\_map>

#include <memory>

using namespace std;

// Define a Node structure

struct Node {

int value;

Node\* left;

Node\* right;

Node(int val) : value(val), left(nullptr), right(nullptr) {}

};

class BinaryTree {

private:

Node\* root;

unordered\_map<int, Node\*> nodes; // Map to store value -> Node pointer

public:

BinaryTree() : root(nullptr) {}

// Function to add a node

void addNode(int parentValue, int childValue, char direction) {

if (root == nullptr) {

root = new Node(parentValue);

nodes[parentValue] = root;

}

if (nodes.find(parentValue) == nodes.end()) {

cout << "Parent node " << parentValue << " not found!\n";

return;

}

Node\* parent = nodes[parentValue];

Node\* child = new Node(childValue);

if (direction == 'L' || direction == 'l') {

if (parent->left != nullptr) {

cout << "Left child already exists for node " << parentValue << "!\n";

return;

}

parent->left = child;

} else if (direction == 'R' || direction == 'r') {

if (parent->right != nullptr) {

cout << "Right child already exists for node " << parentValue << "!\n";

return;

}

parent->right = child;

} else {

cout << "Invalid direction! Use 'L' for left and 'R' for right.\n";

delete child;

return;

}

nodes[childValue] = child;

cout << "Node " << childValue << " added as " << (direction == 'L' ? "left" : "right") << " child of " << parentValue << ".\n";

}

// Function to find and print the parent of a node

void findParent(int value) {

if (root == nullptr || nodes.find(value) == nodes.end()) {

cout << "Node " << value << " not found!\n";

return;

}

for (auto& pair : nodes) {

Node\* parent = pair.second;

if ((parent->left && parent->left->value == value) ||

(parent->right && parent->right->value == value)) {

cout << "Parent of node " << value << " is " << parent->value << ".\n";

return;

}

}

cout << "Node " << value << " is the root and has no parent.\n";

}

// Function to find and print the left child of a node

void findLeftChild(int value) {

if (root == nullptr || nodes.find(value) == nodes.end()) {

cout << "Node " << value << " not found!\n";

return;

}

Node\* node = nodes[value];

if (node->left) {

cout << "Left child of node " << value << " is " << node->left->value << ".\n";

} else {

cout << "Node " << value << " has no left child.\n";

}

}

// Function to find and print the right child of a node

void findRightChild(int value) {

if (root == nullptr || nodes.find(value) == nodes.end()) {

cout << "Node " << value << " not found!\n";

return;

}

Node\* node = nodes[value];

if (node->right) {

cout << "Right child of node " << value << " is " << node->right->value << ".\n";

} else {

cout << "Node " << value << " has no right child.\n";

}

}

// Function to find and print both children of a node

void findBothChildren(int value) {

findLeftChild(value);

findRightChild(value);

}

};

int main() {

BinaryTree tree;

int choice;

do {

cout << "\nBinary Tree Menu:\n";

cout << "1. Add Node\n";

cout << "2. Find Parent\n";

cout << "3. Find Left Child\n";

cout << "4. Find Right Child\n";

cout << "5. Find Both Children\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int parentValue, childValue;

char direction;

cout << "Enter parent value: ";

cin >> parentValue;

cout << "Enter child value: ";

cin >> childValue;

cout << "Enter direction (L for left, R for right): ";

cin >> direction;

tree.addNode(parentValue, childValue, direction);

break;

}

case 2: {

int value;

cout << "Enter node value: ";

cin >> value;

tree.findParent(value);

break;

}

case 3: {

int value;

cout << "Enter node value: ";

cin >> value;

tree.findLeftChild(value);

break;

}

case 4: {

int value;

cout << "Enter node value: ";

cin >> value;

tree.findRightChild(value);

break;

}

case 5: {

int value;

cout << "Enter node value: ";

cin >> value;

tree.findBothChildren(value);

break;

}

case 6:

cout << "Exiting program.\n";

break;

default:

cout << "Invalid choice! Please try again.\n";

}

} while (choice != 6);

return 0;

}