Trường Đại học Sài Gòn

**Khoa công nghệ thông tin**

A blue circle with purple letters

Description automatically generated

**CTDLGT - TH5 - Bai Tap ve Cay nhi phan**

Sinh viên thực hiện: Bùi Viết Quang Vinh

MSSV:3124411347

Thành phố Hồ Chí Minh, tháng 3 năm 2025

**BÀI 001:**

Cho một cây nhị phân có nút gốc là Root, mỗi nút trong cây chứa một số

nguyên:

Viết chương trình tính trung bình cộng các nút trong cây.

Viết chương trình tính trung bình cộng các số dương trong cây.

Viết chương trình tính trung bình cộng các số âm trong cây.

Viết chương trình tính tính tỉ số R=a/b. Với a là tổng các nút có giá trị

dương, b là tổng các nút có giá trị âm.

#include<iostream>

using namespace std;

//Tao cấu trúc cây nhị phân

struct *Node*

{

    int data;

*Node*\* left;

*Node*\* right;

};

typedef *Node*\* *Tree*;

*Node*\* CreateNode(int *value*)

{

*Node*\* p = **new** *Node*;

    if(p == NULL) return NULL;

    p->left = NULL;

    p->right = NULL;

    p->data = *value*;

    return p;

}

//Them node vao cay

void ThemNode(*Tree* &*root*, *Node*\* *p*)

{

    if (*root* == NULL)

*root* = *p*;

    else

    {

        if (*p*->data < *root*->data)

            ThemNode(*root*->left, *p*);

        else if (*p*->data > *root*->data)

            ThemNode(*root*->right, *p*);

    }

}

//Nhap Node

void NhapNode(*Tree* &*root*)

{

    int n, value;

    cout << "Nhap so luong Node: ";

    cin >> n;

    for (int i = 0; i < n; i++)

    {

        cout << "Nhap gia tri Node thu " << i + 1 << ": ";

        cin >> value;

*Node*\* p = CreateNode(value);

        if (p == NULL)

        {

            cout << "Khong tao duoc Node" << endl;

            return;

        }

        ThemNode(*root*, p);

    }

}

//Cau a: Viết chương trình tính trung bình cộng các nút trong cây.

//dem Node

int DemNode(*Tree* *root*)

{

    if (*root* == NULL) return 0;

    return DemNode(*root*->left) + DemNode(*root*->right) + 1;

}

//Tinh tong Node

int TinhTong(*Tree* *root*)

{

    if(*root* == NULL) return 0;

    return *root*->data + TinhTong(*root*->left) + TinhTong(*root*->right);

}

//Tinh trung binh cong

float TrungBinhCong(*Tree* *root*)

{

    int count = DemNode(*root*);

    if(count == 0) return 0;

    return (float)TinhTong(*root*)/count;

}

// Cau b: Viết chương trình tính trung bình cộng các số dương trong cây.

int DemNodeDuong(*Tree* *root*)

{

    if (*root* == NULL) return 0;

    int count = DemNodeDuong(*root*->left) + DemNodeDuong(*root*->right);

    if (*root*->data > 0) return count + 1;

    return count;

}

int TinhTongDuong(*Tree* *root*)

{

    if (*root* == NULL) return 0;

    int sum = TinhTongDuong(*root*->left) + TinhTongDuong(*root*->right);

    if (*root*->data > 0) return sum + *root*->data;

    return sum;

}

float TrungBinhCongDuong(*Tree* *root*)

{

    int count = DemNodeDuong(*root*);

    if (count == 0) return 0;

    return (float)TinhTongDuong(*root*) / count;

}

// Cau c: Viết chương trình tính trung bình cộng các số âm trong cây.

int DemNodeAm(*Tree* *root*)

{

    if (*root* == NULL) return 0;

    int count = DemNodeAm(*root*->left) + DemNodeAm(*root*->right);

    if (*root*->data < 0) return count + 1;

    return count;

}

int TongAm(*Tree* *root*)

{

    if (*root* == NULL) return 0;

    int sum = TongAm(*root*->left) + TongAm(*root*->right);

    if (*root*->data < 0) return sum + *root*->data;

    return sum;

}

float TrungBinhCongAm(*Tree* *root*)

{

    int count = DemNodeAm(*root*);

    if (count == 0) return 0;

    return (float)TongAm(*root*) / count;

}

// Cau d: Viết chương trình tính tính tỉ số R=a/b. Với a là tổng các nút có giá trị

// dương, b là tổng các nút có giá trị âm.

float tinhtyso(*Tree* *root*)

{

    int a = TinhTongDuong(*root*);

    int b = TongAm(*root*);

    if (b == 0) return 0;

    return (float)a / b;

}

void TestCase()

{

*Tree* root = NULL;

    int testValues[] = {10, -5, 15, 3, -8, 12, 20, -2, 4};

    int n = sizeof(testValues) / sizeof(testValues[0]);

    for (int i = 0; i < n; i++)

    {

*Node*\* p = CreateNode(testValues[i]);

        ThemNode(root, p);

    }

    cout << "\nTest Case: \n";

    cout << "Cau a: Trung binh cong cac node trong cay la: " << TrungBinhCong(root) << endl;

    cout << "Cau b: Trung binh cong cac node duong trong cay la: " << TrungBinhCongDuong(root) << endl;

    cout << "Cau c: Trung binh cong cac node am trong cay la: " << TrungBinhCongAm(root) << endl;

    cout << "Cau d: Ty so R = a/b la: " << tinhtyso(root) << endl;

}

int main()

{

*Tree* root = NULL;

    NhapNode(root);

    cout << "Cau a: Trung binh cong cac node trong cay la: " << TrungBinhCong(root) << endl;

    cout << "Cau b: Trung binh cong cac node duong trong cay la: " << TrungBinhCongDuong(root) << endl;

    cout << "Cau c: Trung binh cong cac node am trong cay la: " << TrungBinhCongAm(root) << endl;

    cout << "Cau d: Ty so R = a/b la: " << tinhtyso(root) << endl;

    TestCase();

    return 0;

}

**Bài 002:**

NODE \*temp=p->pLeft;

p->pLeft = p->pRight;

p->pRight=temp;

temp = p->pLeft->pLeft;

p->pLeft->pLeft=p->pLeft->pRight;

p->pLeft->pRight=temp;

**Bài 003:**

**10**

**/ \**

**5 15**

**/ \ / \**

**3 9 12 18**

**/ \**

**7 20**

**3, 7, 9, 5, 12, 20, 18, 15, 10:LRN**

**Bài 004:**

//// hàm tìm phần tử lớn nhất trong cây

int TimMax(TREE t)

{

// CÁCH TỐI ƯU NHẤT

// nếu node cha mà không tồn cây con phải - thì node cha này chính là node ngoài cùng nhất của cây con phải - cũng chính là node có giá trị lớn nhất

if (t->pRight == NULL)

{

return t->data;

}

return TimMax(t->pRight); // duyệt đến node cuối cùng nhất bên cây con phải ==> để tìm giá trị lớn nhất

}

//hàm tìm giá trị MIN

int TimMin(TREE t)

{

// CÁCH TỐI ƯU NHẤT

// nếu node cha mà không tồn cây con trái - thì node cha này chính là node ngoài cùng nhất của cây con trái - cũng chính là node có giá trị nhỏ nhất

if (t->pLeft == NULL)

{

return t->data;

}

return TimMin(t->pLeft); // duyệt đến node cuối cùng nhất bên cây con trái ==> để tìm giá trị nhỏ nhất

}

**Bài 005:**

//Hàm tính node con

int DemMotCon(Tree root)

{

if(root==NULL) return 0;

//Kiểm tra nếu node có đúng 1 con

//Kiểm tra điều kiện có con trái nhưng không có con phải hoặc có con phải nhưng không có con trái

if(root->left && ! root->right || ! root->left && root->right)

return 1 + DemMotCon(root->left) + DemMotCon(root->right);

return DemMotCon(root->left) +DemMotCon(root->right);

}

**Bài 006:**

#include <iostream>

using namespace std;

struct *Node* {

    int data;

*Node*\* left;

*Node*\* right;

};

typedef *Node*\* *Tree*;

// Hàm tạo một nút mới

*Node*\* CreateNode(int *value*) {

*Node*\* p = **new** *Node*;

    if (p == NULL) return NULL;

    p->left = p->right = NULL;

    p->data = *value*;

    return p;

}

// Hàm thêm một nút vào cây BST

void ThemNode(*Tree* &*root*, int *value*) {

    if (*root* == NULL) {

*root* = CreateNode(*value*);

    } else {

        if (*value* < *root*->data) ThemNode(*root*->left, *value*);

        else if (*value* > *root*->data) ThemNode(*root*->right, *value*);

    }

}

// Hàm đếm số lượng nút trong cây

int DemNode(*Tree* *root*) {

    if (*root* == NULL) return 0;

    return 1 + DemNode(*root*->left) + DemNode(*root*->right);

}

// Hàm tính tổng giá trị các nút trong cây

int TinhTong(*Tree* *root*) {

    if (*root* == NULL) return 0;

    return *root*->data + TinhTong(*root*->left) + TinhTong(*root*->right);

}

// ✅ Hàm kiểm thử cây BST

void TestBST() {

*Tree* root = NULL;

    // Tạo danh sách giá trị kiểm tra

    int testValues[] = {50, 30, 70, 20, 40, 60, 80};

    int n = sizeof(testValues) / sizeof(testValues[0]);

    // Thêm các giá trị vào cây

    for (int i = 0; i < n; i++) {

        ThemNode(root, testValues[i]);

    }

    // Kiểm tra số lượng nút

    cout << "✅ Test: So luong nut trong cay BST: " << DemNode(root) << " (Ky vong: 7)" << endl;

    // Kiểm tra tổng giá trị các nút

    cout << "✅ Test: Tong cac nut trong cay BST: " << TinhTong(root) << " (Ky vong: 350)" << endl;

    // Kiểm tra thêm một node mới

    cout << "➡ Them gia tri 25 vao cay BST..." << endl;

    ThemNode(root, 25);

    cout << "✅ Test: So luong nut moi sau khi them: " << DemNode(root) << " (Ky vong: 8)" << endl;

}

int main() {

    TestBST();

    return 0;

}

**Bài 007:**

\*\*\*Giống nhau:

+CTDL động.

+các thao tác cơ bản ,thêm xóa cập nhật 1 cách linh hoạt

\*\*\*Khác nhau:

+Dữ liệu trên cây nhị phân tìm kiếm được tổ chức còn dslk đơn thì không.

+Chi phí tìm kiếm ,thêm cây nhanh hơn dslk đơn

**Bài 008:**

\*\*\* Định nghĩa hàm

int Xuat(chả \*filename,Tree t)

{

FILE\*fp=fopen (filename,"wb");

if(fp==NULL) return 0;

NLR(t,fp);

fclose(fp);

return 1;

}

void LNR(Tree t,FILE \*fp)

{

if(t==NULL) return ;

LNR(t->pLeft,fp);

fwrite (&t->info,sizeof(float),1,fp);

NLR(t->pLeft,fp);

NLR(t->pRight,fp);

}

**Bài 009:**

int Xuat(char \*filename,Tree t)

{

FILE\*fp=fopen (filename,"wb");

if(fp==NULL) return 0;

NLR(t,fp);

fclose(fp);

return 1;

}

void NLR(Tree t,FILE \*fp)

{

if(t==NULL) return ;

fwrite (&t->info,sizeof(float),1,fp);

NLR(t->pLeft,fp);

NLR(t->pRight,fp);

}

**Bài 10:**

int Xuat(char \*filename,Tree t)

{

FILE\*fp=fopen (filename,"wb");

if(fp==NULL) return 0;

NLR(t,fp);

fclose(fp);

return 1;

}

void LNR(Tree t,FILE\*fp)

{

if(t==NULL) return ;

LRN(t->pLeft,fp);

LRN(t->pRight,fp);

fwrite(&t->info,sizeof(float),1,fp);

}

**Bài 11+12:**

Duyệt cây theo phương pháp

LNR ta sẽ được các giá trị tăng

dần

Duyệt cây theo phương pháp

RNL ta sẽ được các giá trị giảm

dần

**Bài 13:**

int Xuat(char \*filename,TREE t)

{

FILE\*fp=fopen(filename,

”wb”);

if(fp==NULL)

return 0;

NLR(t,fp);

fclose(fp);

return 1

}

void NLR(TREE t,FILE\*fp)

{

if(t==NULL)

return;

fwrite(&t->info,

sizeof(float),1,fp);

NLR(t->pLeft,fp);

NLR(t->pRight,fp);

}

**Bài 14:**

#include <iostream>

using namespace std;

// Cấu trúc nút trong cây nhị phân tìm kiếm (BST)

struct *TreeNode* {

    int data;

*TreeNode*\* left;

*TreeNode*\* right;

};

// Cấu trúc nút trong danh sách liên kết đơn

struct *NodeList* {

    int data;

*NodeList*\* next;

};

// Hàm tạo một nút mới trong BST

*TreeNode*\* CreateNode(int *value*) {

*TreeNode*\* p = **new** *TreeNode*;

    p->data = *value*;

    p->left = p->right = NULL;

    return p;

}

// Tìm vị trí cần chèn một nút vào cây BST

*TreeNode*\* FindInsertNode(*TreeNode*\* *root*, int *value*) {

    if (*root* == NULL) return NULL;

*TreeNode*\* p = *root*;

*TreeNode*\* f = p;

    while (p != NULL) {

        f = p;

        if (p->data > *value*)

            p = p->left;

        else

            p = p->right;

    }

    return f;

}

// Chèn một nút vào cây BST

void InsertNode(*TreeNode*\*& *root*, int *value*) {

*TreeNode*\* n = CreateNode(*value*);

    if (*root* == NULL) {

*root* = n;

        return;

    } else {

*TreeNode*\* f = FindInsertNode(*root*, *value*);

        if (f != NULL) {

            if (f->data > *value*)

                f->left = n;

            else

                f->right = n;

        }

    }

}

// Tạo cây nhị phân tìm kiếm từ một mảng

void CreateTree(*TreeNode*\*& *root*, int *arr*[], int *n*) {

    for (int i = 0; i < *n*; i++) {

        InsertNode(*root*, *arr*[i]);

    }

}

//  Hàm duyệt cây theo thứ tự NRL (Root - Right - Left)

void Duyet\_NRL(*TreeNode*\* *root*) {

    if (*root* != NULL) {

        cout << *root*->data << " "; // Xuất dữ liệu trong nút gốc

        Duyet\_NRL(*root*->right); // Duyệt nhánh phải trước

        Duyet\_NRL(*root*->left);  // Duyệt nhánh trái sau

    }

}

void InsertToList(*NodeList*\*& *head*, int *value*) {

*NodeList*\* newNode = **new** *NodeList*;

    newNode->data = *value*;

    newNode->next = *head*;

*head* = newNode;

}

void TreeToList(*TreeNode*\* *root*, *NodeList*\*& *head*) {

    if (*root* == NULL) return;

    // Duyệt nhánh phải trước (giá trị lớn hơn)

    TreeToList(*root*->right, *head*);

    // Chèn nút hiện tại vào danh sách liên kết đơn (chèn đầu để giữ thứ tự giảm dần)

    InsertToList(*head*, *root*->data);

    // Duyệt nhánh trái sau (giá trị nhỏ hơn)

    TreeToList(*root*->left, *head*);

}

// ✅ Hàm in danh sách liên kết đơn

void PrintList(*NodeList*\* *head*) {

    while (*head* != NULL) {

        cout << *head*->data << " -> ";

*head* = *head*->next;

    }

    cout << "NULL" << endl;

}

int main() {

*TreeNode*\* root = NULL;

*NodeList*\* head = NULL;

    int values[] = {50, 30, 70, 20, 40, 60, 80};

    int n = sizeof(values) / sizeof(values[0]);

    CreateTree(root, values, n);

    cout << "Cay duyet theo thu tu NRL: ";

    Duyet\_NRL(root);

    cout << endl;

    TreeToList(root, head);

    // In danh sách liên kết đơn

    cout << "Danh sach lien ket giam dan tu BST: ";

    PrintList(head);

    return 0;

}

**Bài 15:**

Giữa cấu trúc cây nhị phân tìm kiếm và cấu trúc

mảng các phần tử được sắp thứ tự tăng dần có

những điểm giống và khác nhau như thế nào.

\*\*\* Giống nhau

• Dữ liệu được tổ chức.

• Chi phí tìm kiếm một phần tử trên cả hai ctdl là như nhau.

\*\*\* Khác nhau

• Chi phí thêm và xoá phần tử vào mảng lớn hơn chi phí cây nhị

phân tìm kiếm.

**Bài 16:**

#include <iostream>

using namespace std;

struct *TreeNode* {

    int key; // Khóa của node

    int So\_lan; // Số lần xuất hiện

*TreeNode*\* left;

*TreeNode*\* right;

};

struct *BST\_TREE* {

*TreeNode*\* pRoot; // Nút gốc của cây

};

// Hàm tạo node mới

*TreeNode*\* CreateNode(int *value*) {

*TreeNode*\* p = **new** *TreeNode*;

    p->key = *value*;

    p->So\_lan = 1;

    p->left = p->right = NULL;

    return p;

}

// Hàm chèn một phần tử vào BST

void InsertNode(*TreeNode*\*& *root*, int *value*) {

    if (*root* == NULL) {

*root* = CreateNode(*value*);

        return;

    }

    if (*value* < *root*->key)

        InsertNode(*root*->left, *value*);

    else if (*value* > *root*->key)

        InsertNode(*root*->right, *value*);

    else

*root*->So\_lan++; // Nếu trùng giá trị, tăng So\_lan

}

// Tìm node nhỏ nhất trong cây con phải (dùng cho xóa)

*TreeNode*\* FindMin(*TreeNode*\* *root*) {

    while (*root*->left != NULL)

*root* = *root*->left;

    return *root*;

}

// Hàm xóa nút trong BST

*TreeNode*\* DeleteNode(*TreeNode*\* *root*, int *value*) {

    if (*root* == NULL) {

        cout << "Khong tim thay " << *value* << " trong cay." << endl;

        return *root*;

    }

    if (*value* < *root*->key)

*root*->left = DeleteNode(*root*->left, *value*);

    else if (*value* > *root*->key)

*root*->right = DeleteNode(*root*->right, *value*);

    else {

        // Nếu So\_lan > 1, chỉ giảm số lần xuất hiện

        if (*root*->So\_lan > 1) {

*root*->So\_lan--;

            cout << "Giam So\_lan cua " << *value* << " xuong " << *root*->So\_lan << endl;

            return *root*;

        }

        // Nếu So\_lan == 1, xóa theo quy tắc BST

        if (*root*->left == NULL) {

*TreeNode*\* temp = *root*->right;

**delete** *root*;

            return temp;

        }

        else if (*root*->right == NULL) {

*TreeNode*\* temp = *root*->left;

**delete** *root*;

            return temp;

        }

        // Nếu có 2 con, tìm phần tử nhỏ nhất của cây con phải

*TreeNode*\* temp = FindMin(*root*->right);

*root*->key = temp->key;

*root*->So\_lan = temp->So\_lan; // Sao chép số lần xuất hiện

        temp->So\_lan = 1; // Đặt về 1 để xóa

*root*->right = DeleteNode(*root*->right, temp->key);

    }

    return *root*;

}

// Hàm xóa giá trị trong cây BST\_TREE

void XoaGiaTri(*BST\_TREE*& *t*, int *value*) {

*t*.pRoot = DeleteNode(*t*.pRoot, *value*);

}

// Hàm duyệt cây theo NLR

void NLR(*TreeNode*\* *root*) {

    if (*root* == NULL) return;

    cout << *root*->key << " (" << *root*->So\_lan << ") ";

    NLR(*root*->left);

    NLR(*root*->right);

}

// Hàm khởi tạo cây

void KhoiTaoCay(*BST\_TREE*& *t*) {

*t*.pRoot = NULL;

}

// Hàm nhập dữ liệu cho cây BST\_TREE

void NhapCay(*BST\_TREE*& *t*, int *arr*[], int *n*) {

    for (int i = 0; i < *n*; i++)

        InsertNode(*t*.pRoot, *arr*[i]);

}

// Hàm test key

void TestBST() {

*BST\_TREE* t;

    KhoiTaoCay(t);

    int arr[] = {50, 30, 70, 20, 40, 60, 80, 30, 70, 50}; // 30, 50, 70 xuất hiện 2 lần

    int n = sizeof(arr) / sizeof(arr[0]);

    // Chèn các phần tử vào BST

    NhapCay(t, arr, n);

    // Xuất cây trước khi xóa

    cout << "Cay truoc khi xoa: ";

    NLR(t.pRoot);

    cout << endl;

    // Thử nghiệm xóa các phần tử

    XoaGiaTri(t, 30); // Giảm So\_lan

    XoaGiaTri(t, 50); // Giảm So\_lan

    XoaGiaTri(t, 80); // Xóa hoàn toàn

    XoaGiaTri(t, 100); // Không tồn tại

    // Xuất cây sau khi xóa

    cout << "Cay sau khi xoa: ";

    NLR(t.pRoot);

    cout << endl;

}

int main() {

    TestBST();

    return 0;

}

**Bài 17:**

\*\*\*Tận cùng bên trái của nhánh phải

void SearchStandFor(TREE&p,TREE&q)

{

if(q->pLeft)

SearchStandFor(p,q->pLeft); else

{

p->info=q->info;

p=q;

q=q->pRight;

}

}

Tận cùng bên trái của nhánh phải

void SearchStandFor(TREE&p,TREE&q)

{

if(q->pLeft)

SearchStandFor(p,q->pLeft);

else

{

p->info=q->info;

p=q;

q=q->pRight;

}

}

Tận cùng bên phải của nhánh trái

void SearchStandFor(TREE&p,

TREE&q)

{

if(q->pRight)

SearchStandFor(p,q->pRight);

else

{

p->info = q->info;

p = q;

q = q->pLeft;

}

}

**Bài 18:**

#include<iostream>

using namespace std;

struct *NODE*

{

    int data;

    struct *NODE*\*left;

    struct *NODE*\*right;

};

typedef struct *NODE* *Node*;

typedef *Node*\**TREE*;

//Tao cay nhi phan tu mang

void Init(*TREE* &*t*)

{

*t*=nullptr;

}

*Node*\* GetNode(int *value*){

*Node*\*p=**new** *Node*;

    if(p==nullptr) return NULL;

    p->data=*value*;

    p->left=p->right=nullptr;

    return p;

}

int InsertNode(*TREE* &*t*,int *value*)

{

    if(*t*)

    {

        if(*t*->data==*value*)

            return 0;

        if(*t*->data <*value*) return InsertNode(*t*->right,*value*);

        else return InsertNode(*t*->left,*value*);

    }

*t*=GetNode(*value*);

    if(*t*==nullptr) return -1;

    return 1;

}

int TaoCay(*TREE* &*t*,int *arr*[],int *n*)

{

    Init(*t*);

    for(int i=0;i<*n*;i++)

    {

        if(InsertNode(*t*,*arr*[i])==-1) return 0;

    }

    return 1;

}

//Duyet cay dau tro laii vao mang

void LNR(*TREE* *t*,int *arr*[],int &*n*)

{

    if(*t*==nullptr) return ;

    LNR(*t*->left,*arr*,*n*);

*arr*[*n*++]=*t*->data;

*n*++;

    LNR(*t*->right,*arr*,*n*);

}

void PrintArray(int *arr*[], int *n*) {

    for (int i = 0; i < *n*; i++) {

        cout << *arr*[i] << " ";

    }

    cout << endl;

}

// Test case

int main() {

    int arr[] = {50, 30, 70, 20, 40, 60, 80};

    int n = sizeof(arr) / sizeof(arr[0]);

*TREE* t;

    TaoCay(t, arr, n);

    int sortedArr[100], sortedSize = 0;

    LNR(t, sortedArr, sortedSize);

    cout << "Mảng sau khi sắp xếp tăng dần bằng BST: ";

    PrintArray(sortedArr, sortedSize);

    return 0;

}

**Bài 33+37:**

#include <iostream>

#include <stack>

#include <queue>

#include <cmath> // Dùng abs() để kiểm tra độ cao

using namespace std;

// \*\*Định nghĩa cấu trúc cây nhị phân\*\*

struct *NODE* {

    int info;

*NODE*\* pLeft;

*NODE*\* pRight;

};

typedef *NODE*\* *TREE*;

// \*\*Tạo một nút mới\*\*

*NODE*\* CreateNode(int *value*) {

*NODE*\* p = **new** *NODE*;

    if (!p) return NULL;

    p->info = *value*;

    p->pLeft = p->pRight = NULL;

    return p;

}

// \*\*Chèn một nút vào cây nhị phân tìm kiếm (BST)\*\*

void InsertNode(*TREE* &*root*, int *value*) {

    if (*root* == NULL) {

*root* = CreateNode(*value*);

        return;

    }

    if (*value* < *root*->info)

        InsertNode(*root*->pLeft, *value*);

    else if (*value* > *root*->info)

        InsertNode(*root*->pRight, *value*);

}

// \*\*1️⃣ Duyệt cây theo thứ tự NLR không dùng đệ quy (dùng stack)\*\*

void DuyetNLR\_KhongDeQuy(*TREE* *root*) {

    if (*root* == NULL) return;

    stack<*NODE*\*> s;

    s.push(*root*);

    while (!s.empty()) {

*NODE*\* temp = s.top();

        s.pop();

        cout << temp->info << " "; // Xử lý nút hiện tại

        if (temp->pRight) s.push(temp->pRight);

        if (temp->pLeft) s.push(temp->pLeft);

    }

}

// \*\*2️⃣ Duyệt cây theo mức không dùng đệ quy (dùng queue)\*\*

void DuyetTheoMuc(*TREE* *root*) {

    if (*root* == NULL) return;

    queue<*NODE*\*> q;

    q.push(*root*);

    while (!q.empty()) {

*NODE*\* temp = q.front();

        q.pop();

        cout << temp->info << " ";

        if (temp->pLeft) q.push(temp->pLeft);

        if (temp->pRight) q.push(temp->pRight);

    }

}

// \*\*3️⃣ Tìm phần tử lớn nhất trong cây\*\*

*NODE*\* LonNhat(*TREE* *t*) {

    if (*t* == NULL) return NULL;

*NODE*\* lc = *t*;

*NODE*\* a = LonNhat(*t*->pLeft);

    if (a && a->info > lc->info) lc = a;

*NODE*\* b = LonNhat(*t*->pRight);

    if (b && b->info > lc->info) lc = b;

    return lc;

}

// \*\*4️⃣ Tìm phần tử nhỏ nhất trong cây\*\*

*NODE*\* NhoNhat(*TREE* *t*) {

    if (*t* == NULL) return NULL;

*NODE*\* lc = *t*;

*NODE*\* a = NhoNhat(*t*->pLeft);

    if (a && a->info < lc->info) lc = a;

*NODE*\* b = NhoNhat(*t*->pRight);

    if (b && b->info < lc->info) lc = b;

    return lc;

}

// \*\*5️⃣ Tính chiều cao của cây\*\*

int ChieuCao(*TREE* *t*) {

    if (*t* == NULL) return 0;

    int a = ChieuCao(*t*->pLeft);

    int b = ChieuCao(*t*->pRight);

    return max(a, b) + 1;

}

// \*\*6️⃣ Kiểm tra cây có cân bằng không\*\*

int ktCanBang(*TREE* *root*) {

    if (*root* == NULL) return 1;

    if (!ktCanBang(*root*->pLeft) || !ktCanBang(*root*->pRight)) return 0;

*NODE*\* a = LonNhat(*root*->pLeft);

    if (a && a->info > *root*->info) return 0;

    a = NhoNhat(*root*->pRight);

    if (a && a->info < *root*->info) return 0;

    int x = ChieuCao(*root*->pLeft);

    int y = ChieuCao(*root*->pRight);

    return abs(x - y) <= 1;

}

// \*\*Hàm kiểm tra và hiển thị kết quả\*\*

void TestCanBang(*TREE* *root*) {

    if (ktCanBang(*root*))

        cout << "Cay can bang\n";

    else

        cout << "Cay khong can bang\n";

}

// \*\*Hàm chính để test\*\*

int main() {

*TREE* t = NULL;

    int arr[] = {50, 30, 70, 20, 40, 60, 80};

    int n = sizeof(arr) / sizeof(arr[0]);

    // Tạo cây từ mảng

    for (int i = 0; i < n; i++) {

        InsertNode(t, arr[i]);

    }

    // Test các thuật toán

    cout << "Duyet NLR (khong de quy): ";

    DuyetNLR\_KhongDeQuy(t);

    cout << "\n";

    cout << "Duyet theo muc: ";

    DuyetTheoMuc(t);

    cout << "\n";

    cout << "Phan tu lon nhat: " << LonNhat(t)->info << "\n";

    cout << "Phan tu nho nhat: " << NhoNhat(t)->info << "\n";

    cout << "Chieu cao cua cay: " << ChieuCao(t) << "\n";

    // Kiểm tra cây có cân bằng không

    TestCanBang(t);

    return 0;

}