EXERCISES BINARY TREE

LÝ QUANG THẮNG - 22110202

> ⚠ **Warning**
>
> Hãy copy các đoạn mã hóa theo thứ tự do mỗi cây có số lượng cạnh khác nhau

**SOURCE CODE ONLINE LINK
source code bài 1
source code bài 2
source code bài 3

BÀI 1:

- Mã hóa cây 1:
  Số cạnh: 14

```
1 2 L 1 3 R 2 4 L 2 5 R 4 8 L 4 9 R 5 10 L 5 11 R 3 6 L 3 7 R 6 12 L 6 13 R 7 14 L 7 15 R
```

- Mã hóa cây 2:
  Số cạnh: 10

```
50 17 L 50 76 R 17 9 L 17 23 R 9 14 R 14 12 L 23 19 L 76 54 L 54 72 R 72 67 L
```

- Mã hóa cây 3:
  Số cạnh: 10

```
15 11 L 15 26 R 11 8 L 11 12 R 8 6 L 8 9 R 12 14 R 26 20 L 26 30 R 30 35 R
```

- Mã hóa cây 4:
  Số cạnh: 11

```
3 1 L 3 10 R 1 13 L 1 5 R 5 6 L 10 11 L 10 16 R 16 15 L 16 2 R 15 9 L 15
4 R
```

FULL SOURCE CODE:

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct NodeType {
    int data;
    struct NodeType* left, * right;
} TreeNode;

typedef struct BinaryTreeType {
    struct NodeType* root;
} BinaryTree;

TreeNode* makeNode(int data) {
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void print(TreeNode* Node) {
    if(!Node) return;
    print(Node->left);
    printf("%d ", Node->data);
    print(Node->right);
}

void init(BinaryTree* tree) {
    tree->root = NULL;
}
```

```
void makeRoot(TreeNode* root, int u, int v, char c) {
    if(c == 'L') root->left = makeNode(v);
    else root->right = makeNode(v);
}

// chú thích (u v c): node v là lá bên c của u
void insert(TreeNode* Node, int u, int v, char c) {
    if(!Node) return;
    if(Node->data == u) {
        makeRoot(Node , u, v, c);
    }else {
        insert(Node->left, u, v, c);
        insert(Node->right, u, v, c);
    }
}

void search(TreeNode* node, int val, int* check) {
    if(!node) return;
    if(node->data == val) {
        *check = 1;
    }
    search(node->left, val, check);
    search(node->right, val, check);
}


void solve(int n) {
    BinaryTree tree;
    init(&tree);
    printf("Input your %d edges tree:\n", n);
    for(int i = 0; i < n; i++) {
        int u, v; char c, khoangcach;
        scanf("%d%d", &u, &v); scanf("%c", &khoangcach);
        scanf("%c", &c);
        if(tree.root == NULL) {
            tree.root = makeNode(u);
            makeRoot(tree.root, u, v, c);
        }else {
            insert(tree.root, u, v, c);
        }
    }
```

```
    printf("Tree : ");
    print(tree.root);
    printf("\n");
}

int main() {
    // Copy dòng mã hóa và dán vào.
    // cay 1:
    // 1 2 L 1 3 R 2 4 L 2 5 R 4 8 L 4 9 R 5 10 L 5 11 R 3 6 L
3 7 R 6 12 L 6 13 R 7 14 L 7 15 R
    int n1 = 14; // là số cạnh của cây
    printf("Tree 1------------\n");
    solve(n1);
    // cay 2:
    // 50 17 L 50 76 R 17 9 L 17 23 R 9 14 R 14 12 L 23 19 L 76
54 L 54 72 R 72 67 L
    int n2 = 10; // là số cạnh của cây
    printf("Tree 2------------\n");
    solve(n2);
    // cay 3:
    // 15 11 L 15 26 R 11 8 L 11 12 R 8 6 L 8 9 R 12 14 R 26 20
L 26 30 R 30 35 R
    int n3 = 10; // là số cạnh của cây
    printf("Tree 3------------\n");
    solve(n3);
    // cay 4:
    // 3 1 L 3 10 R 1 13 L 1 5 R 5 6 L 10 11 L 10 16 R 16 15 L
16 2 R 15 9 L 15 4 R
    int n4 = 11; // là số cạnh của cây
    printf("Tree 4------------\n");
    solve(n4);
    return 0;
}
```

BÀI 2:

Đầu tiên cần kiểm tra xem cây nhị phân nhập vào có phải cây tìm kiếm không.

```
int a[100]; int idx = 0; // dùng để check thứ tự của cây
void check(TreeNode* node) {
    if(!node) return;
    check(node->left);
    a[++idx] = node->data; // lưu vào mảng, nếu mảng tăng dần
or giảm dần -> cây tìm kiếm
    check(node->right);
}
```

Sau đó trong hàm solve ta gọi hàm và duyệt mảng `a`. Nếu `a` tăng dần thì cây là cây tìm kiếm.

```
int ok = 1;
    check(tree.root);
    for(int i = 0; i < n; i++) {
        if(a[i] > a[i + 1]) {
            ok = 0; break;
        }
    }
```

Biến `ok = 1` nếu cây là cây tìm kiếm, ngược lại là cây bình thường.

Các hàm cho cây nhị phân tìm kiếm

Theo thứ tự `search, insert, delete`

```
// use for binary search tree
TreeNode* search(TreeNode* node, int val) {
    if(!node) return NULL; // ko tìm thấy
    if(val > node->data) {
        // tìm bên phải
        return search(node->right, val);
    }else if(val < node->data) {
        // tìm bên trái
        return search(node->left, val);
    }else {
        // đã tìm thấy
        return node;
```

```
    }
}

// use for binary search tree
void insert(TreeNode** Node, int data) {
    if(!(*Node)) {
        *Node = makeNode(data);
    }else if((*Node)->data < data) {
        // gán vào lá bên phải
        insert(&(*Node)->right, data);
    }else if((*Node)->data > data) {
        // gán vào lá bên trái
        insert(&(*Node)->left, data);
    }
}

// use for binary search tree
void deleleNode(TreeNode** node, int val) {
    if(!(*node)) {
        return;
    } // không tồn tại val trong cây
    if(val > (*node)->data) {
        deleleNode(&(*node)->right, val);
    }else if(val < (*node)->data) {
        deleleNode(&(*node)->left, val);
    }else {
        // Nếu node này k có lá
        if(!(*node)->left && !(*node)->right) {
            free(*node);
            *node = NULL;
        }
        // Nếu node này chỉ có lá phải
        else if(!(*node)->left) {
            TreeNode* tmp = (*node)->right;
            *node = tmp;
        }
        // Nếu node này chỉ có lá trái
        else if(!(*node)->right) {
            TreeNode* tmp = (*node)->left;
            *node = tmp;
        }
```

```
        // Nếu node này có cả 2 lá
        else {
            // ta đi tìm Node nhỏ nhất lớn hơn val
            TreeNode* tmp = minNode((*node)->right);
            // gán giá trị của tmp cho node và xóa tmp
            (*node)->data = tmp->data;
            deleleNode(&(*node)->right, tmp->data);
        }
    }
}
```

Các hàm cho cây nhị phân bình thường

Theo thứ tự `search, insert, delete`

```
// use for binary tree
void search1(TreeNode* node, int val, int *ok) {
    if(!node) return;
    if(node->data == val) {
        *ok = 1; return;
    }else {
        search1(node->left, val, ok);
        search1(node->right, val, ok);
    }
}

// Do cây không phải cây tìm kiếm
// ta đi đến tận cùng của node bên trái để chèn
    TreeNode* tmp = tree.root;
    while(tmp != NULL && tmp->left != NULL) {
        // hàm dừng khi node trái = NULL
        tmp = tmp->left;
    }
    tmp->left = makeNode(z); // insert lá trái là 23

// use for binary tree
void deleleNode1(TreeNode** node, int val) {
    if(!(*node)) {
        return;
    }
```

```
    if((*node)->data == val) {
        // Nếu node không có lá nào
        if(!(*node)->left && !(*node)->right) {
            free(*node);
            *node = NULL;
        }
        // Nếu node không có lá trái
        else if(!(*node)->left) {
            TreeNode* tmp = (*node)->right;
            *node = tmp;
        }
        // Nếu node có không có lá phải
        else if(!(*node)->right) {
            TreeNode* tmp = (*node)->left;
            *node = tmp;
        }
        // Nếu node có cả 2 lá
        else {
            // Cho node hiện tại bằng 1 trong 2 lá của nó
            TreeNode* tmp = (*node)->right;
            (*node)->data = tmp->data;
            // sau đó dời lên
            deleleNode1(&(*node)->right, tmp->data);
        }
    }else {
        deleleNode1(&(*node)->left, val);
        deleleNode1(&(*node)->right, val);
    }
}
```

Hàm tìm node có level k

```
// use for find level of leaf
void levelSolve(TreeNode* node, int cur_level, int find_level)
{
    if(!node) return;
    if(cur_level == find_level) {
        printf("%d ", node->data);
    }
    levelSolve(node->left, cur_level + 1, find_level);
```

```
    levelSolve(node->right, cur_level + 1, find_level);
}
```

FULL SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>

typedef struct NodeType {
    int data;
    struct NodeType* left, * right;
} TreeNode;

typedef struct BinaryTreeType {
    struct NodeType* root;
} BinaryTree;

TreeNode* makeNode(int data) {
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void print(TreeNode* node) {
    if(!node) return;
    //
    print(node->left);
    printf("%d ", node->data);
    print(node->right);
    //
}

void init(BinaryTree* tree) {
    tree->root = NULL;
}
// use for binary search tree
TreeNode* search(TreeNode* node, int val) {
    if(!node) return NULL; // ko tìm thấy
```

```
    if(val > node->data) {
        // tìm bên phải
        return search(node->right, val);
    }else if(val < node->data) {
        // tìm bên trái
        return search(node->left, val);
    }else {
        // đã tìm thấy
        return node;
    }
}
// use for binary tree
void search1(TreeNode* node, int val, int *ok) {
    if(!node) return;
    if(node->data == val) {
        *ok = 1; return;
    }else {
        search1(node->left, val, ok);
        search1(node->right, val, ok);
    }
}
// use for binary search tree
void insert(TreeNode** Node, int data) {
    if(!(*Node)) {
        *Node = makeNode(data);
    }else if((*Node)->data < data) {
        // gán vào lá bên phải
        insert(&(*Node)->right, data);
    }else if((*Node)->data > data) {
        // gán vào lá bên trái
        insert(&(*Node)->left, data);
    }
}

void makeRoot(TreeNode* root, int u, int v, char c) {
    if(c == 'L') root->left = makeNode(v);
    else root->right = makeNode(v);
}
// use for binary tree
// chú thích (u v c): node v là lá bên c của u
void insert1(TreeNode* Node, int u, int v, char c) {
```

```
    if(!Node) return;
    if(Node->data == u) {
        makeRoot(Node , u, v, c);
    }else {
        insert1(Node->left, u, v, c);
        insert1(Node->right, u, v, c);
    }
}
// Hàm tìm node đầu tiên nhỏ hơn val
TreeNode* minNode(TreeNode* node) {
    TreeNode* tmp = node;
    while(tmp != NULL && tmp->left != NULL) {
        tmp = tmp->left;
    }
    return tmp;
}
// use for binary search tree
void deleleNode(TreeNode** node, int val) {
    if(!(*node)) {
        return;
    } // không tồn tại val trong cây
    if(val > (*node)->data) {
        deleleNode(&(*node)->right, val);
    }else if(val < (*node)->data) {
        deleleNode(&(*node)->left, val);
    }else {
        // Nếu node này k có lá
        if(!(*node)->left && !(*node)->right) {
            free(*node);
            *node = NULL;
        }
        // Nếu node này chỉ có lá phải
        else if(!(*node)->left) {
            TreeNode* tmp = (*node)->right;
            *node = tmp;
        }
        // Nếu node này chỉ có lá trái
        else if(!(*node)->right) {
            TreeNode* tmp = (*node)->left;
            *node = tmp;
        }
```

```
        // Nếu node này có cả 2 lá
        else {
            // ta đi tìm Node nhỏ nhất lớn hơn val
            TreeNode* tmp = minNode((*node)->right);
            // gán giá trị của tmp cho node và xóa tmp
            (*node)->data = tmp->data;
            deleleNode(&(*node)->right, tmp->data);
        }
    }
}
// use for binary tree
void deleleNode1(TreeNode** node, int val) {
    if(!(*node)) {
        return;
    }
    if((*node)->data == val) {
        // Nếu node không có lá nào
        if(!(*node)->left && !(*node)->right) {
            free(*node);
            *node = NULL;
        }
        // Nếu node không có lá trái
        else if(!(*node)->left) {
            TreeNode* tmp = (*node)->right;
            *node = tmp;
        }
        // Nếu node có không có lá phải
        else if(!(*node)->right) {
            TreeNode* tmp = (*node)->left;
            *node = tmp;
        }
        // Nếu node có cả 2 lá
        else {
            // Cho node hiện tại bằng 1 trong 2 lá của nó
            TreeNode* tmp = (*node)->right;
            (*node)->data = tmp->data;
            // sau đó dời lên
            deleleNode1(&(*node)->right, tmp->data);
        }
    }else {
        deleleNode1(&(*node)->left, val);
```

```
        deleleNode1(&(*node)->right, val);
    }
}
// use for find level of leaf
void levelSolve(TreeNode* node, int cur_level, int find_level)
{
    if(!node) return;
    if(cur_level == find_level) {
        printf("%d ", node->data);
    }
    levelSolve(node->left, cur_level + 1, find_level);
    levelSolve(node->right, cur_level + 1, find_level);
}
int a[100]; int idx = 0; // dùng để check thứ tự của cây
void check(TreeNode* node) {
    if(!node) return;
    check(node->left);
    a[++idx] = node->data; // lưu vào mảng, nếu mảng tăng dần
or giảm dần -> cây tìm kiếm
    check(node->right);
}

void solve(int n) {
    // tao ra cay
    idx = 0; // reset mảng check
    BinaryTree tree;
    init(&tree);
    printf("Input your %d edges tree:\n", n);
    for(int i = 0; i < n; i++) {
        int u, v; char c, khoangcach;
        scanf("%d%d", &u, &v); scanf("%c", &khoangcach);
        scanf("%c", &c);
        if(tree.root == NULL) {
            tree.root = makeNode(u);
            makeRoot(tree.root, u, v, c);
        }else {
            insert1(tree.root, u, v, c);
        }
    }
    printf("Tree : ");
    print(tree.root);
```

```
    printf("\n");
    // ta sẽ kiểm tra xem đây có phải cây nhị phân tìm kiếm
không
    int ok = 1;
    check(tree.root);
    for(int i = 0; i < n; i++) {
        if(a[i] > a[i + 1]) {
            ok = 0; break;
        }
    }
    if(ok) {
        printf("This's a binary search tree!\n");
    }else {
        printf("This's not a binary search tree!\n");
    }
    // 2.1 Tìm nút có giá trị 25
    int x = 25;
    printf("Find %d: ", x);
    if(ok) {
        // Nếu là cây nhị phân tìm kiếm ta dùng hàm search
        TreeNode* tmp = search(tree.root, x);
        if(tmp) {
            printf("Tree has a %d node\n", x);
        }else {
            printf("Not found!\n");
        }
    }else {
        // nếu không ta tìm mọi lá của cây
        int check = 0;
        search1(tree.root, x, &check);
        if(check) {
            printf("Tree has a %d node\n", x);
        }else {
            printf("Not found!\n");
        }
    }
    // 2.2 Xóa nút có giá trị 12
    int y = 12;
    if(ok) {
        // Nếu là cây nhị phân tìm kiếm thì ta sẽ dùng
deleteNode để đảm bảo tính chất
```

```
        deleleNode(&tree.root, y);
    }else {
        // Nếu không phải cây nhị phân tìm kiếm thì ta sẽ dùng
deleteNode1
        deleleNode1(&tree.root, y);
    }
    printf("Tree after deleted %d: ", y);
    print(tree.root); printf("\n");
    // 2.3 Chèn nút có giá trị 23
    int z = 23;
    if(ok) {
        // Nếu là cây nhị phân tìm kiếm thì ta sẽ dùng insert
để đảm bảo tính chất
        insert(&tree.root, z);
    }else {
        // ta đi đến tận cùng của node bên trái để chèn
        TreeNode* tmp = tree.root;
        while(tmp != NULL && tmp->left != NULL) {
            // hàm dừng khi node trái = NULL
            tmp = tmp->left;
        }
        tmp->left = makeNode(z); // insert lá trái là 23
    }
    printf("Tree after inserted %d: ", z);
    print(tree.root); printf("\n");
    // 2.4 Cho biết level 3 của cây gồm nút nào?
    printf("Nodes level 3: ");
    int cur_level = 1, find_level = 3;
    levelSolve(tree.root, cur_level, find_level);
    printf("\n----------------\n");
}

int main() {
    // Copy dòng mã hóa và dán vào.
    // cay 1:
    // 1 2 L 1 3 R 2 4 L 2 5 R 4 8 L 4 9 R 5 10 L 5 11 R 3 6 L
3 7 R 6 12 L 6 13 R 7 14 L 7 15 R
    int n1 = 14; // là số cạnh của cây
    printf("Tree 1------------\n");
    solve(n1);
    // cay 2:
```

```
    // 50 17 L 50 76 R 17 9 L 17 23 R 9 14 R 14 12 L 23 19 L 76
54 L 54 72 R 72 67 L
    int n2 = 10; // là sô cạnh của cây
    printf("Tree 2------------\n");
    solve(n2);
    // cay 3:
    // 15 11 L 15 26 R 11 8 L 11 12 R 8 6 L 8 9 R 12 14 R 26 20
L 26 30 R 30 35 R
    int n3 = 10; // là số cạnh của cây
    printf("Tree 3------------\n");
    solve(n3);
    // cay 4:
    // 3 1 L 3 10 R 1 13 L 1 5 R 5 6 L 10 11 L 10 16 R 16 15 L
16 2 R 15 9 L 15 4 R
    int n4 = 11; // là số cạnh của cây
    printf("Tree 4------------\n");
    solve(n4);
    return 0;
}
```

BÀI 3:

FULL SOURCE CODE:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct NodeType {
    int data;
    struct NodeType* left, * right;
} TreeNode;

typedef struct BinaryTreeType {
    struct NodeType* root;
} BinaryTree;

TreeNode* makeNode(int data) {
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    newNode->data = data;
```

```
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

void print(TreeNode* Node) {
    if(!Node) return;
    print(Node->left);
    printf("%d ", Node->data);
    print(Node->right);
}

void init(BinaryTree* tree) {
    tree->root = NULL;
}

void makeRoot(TreeNode* root, int u, int v, char c) {
    if(c == 'L') root->left = makeNode(v);
    else root->right = makeNode(v);
}

// chú thích (u v c): node v là lá bên c của u
void insert(TreeNode* Node, int u, int v, char c) {
    if(!Node) return;
    if(Node->data == u) {
        makeRoot(Node , u, v, c);
    }else {
        insert(Node->left, u, v, c);
        insert(Node->right, u, v, c);
    }
}

void search(TreeNode* node, int val, int* check) {
    if(!node) return;
    if(node->data == val) {
        *check = 1;
    }
    search(node->left, val, check);
    search(node->right, val, check);
}
```

```
int parent[100];
int minHight = 1e9;
int end = -1;
int start = -1;

void min_path(TreeNode* root, int pr, int hight) {
    if(!root) return;
    if(!root->right && !root->left) {
        // đây là nốt lá
        if(hight < minHight) {
            end = root->data;
            minHight = hight;
        }
    }
    if(root->left) {
        parent[root->left->data] = root->data;
        min_path(root->left, root->data, hight + 1);
    }
    if(root->right) {
        parent[root->right->data] = root->data;
        min_path(root->right, root->data, hight + 1);
    }
}

void level_count(TreeNode* root, int curr_level, int
level_arr[], int *max_count) {
    if(!root) return;
    level_arr[curr_level]++;
    if(level_arr[curr_level] > *max_count) {
        *max_count = level_arr[curr_level];
    }
    level_count(root->left, curr_level + 1, level_arr,
max_count);
    level_count(root->right, curr_level + 1, level_arr,
max_count);
}

void count_one_child(TreeNode* root, int* count) {
    if(!root) return;
    if((!root->left && root->right) || (root->left && !root-
>right)) {
```

```
        *count += 1;
    }
    if(root->left) {
        count_one_child(root->left, count);
    }
    if(root->right) {
        count_one_child(root->right, count);
    }
}

void count_only_left_child(TreeNode* root, int* count) {
    if(!root) return;
    if(!root->right && root->left) {
        *count += 1;
    }
    count_only_left_child(root->left, count);
    count_only_left_child(root->right, count);
}

void find_nearest(TreeNode* root, int *distance, int diff, int
*nearest) {
    if(!root) return;
    if(abs(root->data - diff) < *distance) {
        *distance = abs(root->data - diff);
        *nearest = root->data;
    }
    find_nearest(root->left, distance, diff, nearest);
    find_nearest(root->right, distance, diff, nearest);
}

void solve(int n) {
    // init lai parent
    for(int i = 0; i < 100; i++) {
        parent[i] = 0;
    }
    minHight = 1e9;
    end = -1;
    start = -1;
    //---------------
    BinaryTree tree;
    init(&tree);
```

```
    printf("Input your %d edges tree:\n", n);
    for(int i = 0; i < n; i++) {
        int u, v; char c, khoangcach;
        scanf("%d%d", &u, &v); scanf("%c", &khoangcach);
        scanf("%c", &c);
        if(tree.root == NULL) {
            tree.root = makeNode(u);
            makeRoot(tree.root, u, v, c);
        }else {
            insert(tree.root, u, v, c);
        }
    }
    printf("Tree : ");
    print(tree.root);
    printf("\n");
    // ------- 3.1----------
    printf("3.1:\n");
    int duongdi[100] = {};
    int idx = 0;
    start = tree.root->data;
    min_path(tree.root, -1, 0);
    while(end!=start) {
        duongdi[idx++] = end;
        end = parent[end];
    }
    printf("The shortest path: ");
    printf("%d ", tree.root->data);
    for(int i = idx - 1; i >= 0; i--) {
        printf("%d ", duongdi[i]);
    }
    printf("\n");
    // ------- 3.2----------
    printf("3.2:\n");
    int level[100] = {}; // mảng lưu số lượng node có level là
chỉ số
    int max_count = -1; // biến max count
    int max_level = -1;
    level_count(tree.root, 1, level, &max_count); // curr level
= 1 nghĩa là gốc cây có level 1
    for(int i = 0; i < 100; i++) {
        if(level[i]) {
```

```
            printf("Number of level %d node: %d\n", i,
level[i]);
            if(level[i] == max_count) {
                max_level = i;
            }
        }
    }
    printf("Level %d has the most nodes\n", max_level);
    // ------- 3.3----------
    printf("3.3:\n");
    int count_oneChild_node = 0;
    count_one_child(tree.root, &count_oneChild_node);
    printf("Tree has %d one child node\n",
count_oneChild_node);
    // ------- 3.4----------
    printf("3.4:\n");
    int count_left = 0;
    count_only_left_child(tree.root, &count_left);
    printf("Tree has %d node which have only left child\n",
count_left);
    printf("Difference: %d\n", count_oneChild_node -
count_left);
    // ------- 3.5----------
    printf("3.5:\n");
    int diff = count_oneChild_node - count_left;
    int distance = 1e9;
    int nearest_val = 0;
    find_nearest(tree.root, &distance, diff, &nearest_val);
    printf("Nearest Node Value: %d\n", nearest_val);
}

int main() {
    // Copy dòng mã hóa và dán vào.
    // cay 1:
    // 1 2 L 1 3 R 2 4 L 2 5 R 4 8 L 4 9 R 5 10 L 5 11 R 3 6 L
3 7 R 6 12 L 6 13 R 7 14 L 7 15 R
    int n1 = 14; // là số cạnh của cây
    printf("Tree 1------------\n");
    solve(n1);
    // cay 2:
    // 50 17 L 50 76 R 17 9 L 17 23 R 9 14 R 14 12 L 23 19 L 76
```

```
54 L 54 72 R 72 67 L
    int n2 = 10; // là sô cạnh của cây
    printf("Tree 2------------\n");
    solve(n2);
    // cay 3:
    // 15 11 L 15 26 R 11 8 L 11 12 R 8 6 L 8 9 R 12 14 R 26 20
L 26 30 R 30 35 R
    int n3 = 10; // là số cạnh của cây
    printf("Tree 3------------\n");
    solve(n3);
    // cay 4:
    // 3 1 L 3 10 R 1 13 L 1 5 R 5 6 L 10 11 L 10 16 R 16 15 L
16 2 R 15 9 L 15 4 R
    int n4 = 11; // là số cạnh của cây
    printf("Tree 4------------\n");
    solve(n4);
    return 0;
}
```