B1: Kiểm tra node lá

Cho cây nhị phân, nhiệm vụ của bạn là kiểm tra xem tất cả các node lá của cây có cùng một mức hay không? Ví dụ với cây dưới đây sẽ cho ta kết quả là Yes.

A picture containing clock, clipart

Description automatically generated

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
* T, N, u, v, thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤u, v≤104;

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  2 1 2 R 1 3 L 4 10 20 L 10 30 R 20 40 L 20 60 R | 1 0 |

Code :

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

Node \*ConstructTree(int n){

map<int, Node\*> m;

Node \*root = NULL;

while (n--){

int a, b; char c;

cin >> a >> b >> c;

Node \*parent;

if (m.find(a) == m.end()){

parent = CreateNode(a);

m[a] = parent;

if (root == NULL)

root = parent;

}

else

parent = m[a];

Node \*child = CreateNode(b);

if (c == 'L')

parent->left = child;

else

parent->right = child;

m[b] = child;

}

return root;

}

bool isSameLevel(Node \*root, int currLevel, int reset){

static int level = -1;

if (reset)

level = -1;

if (root->left==NULL && root->right==NULL){

if (level == -1){

level = currLevel;

return true;

}

else if (level == currLevel)

return true;

else

return false;

}

int lRes = true;

int rRes = true;

if (root->left)

lRes = isSameLevel(root->left, currLevel+1, false);

if (root->right)

rRes = isSameLevel(root->right, currLevel+1, false);

if (!lRes || !rRes)

return false;

return true;

}

int main(){

int t; cin >> t;

while (t--){

int n; cin >> n;

Node \*root = ConstructTree(n);

if (isSameLevel(root, 0, true))

cout << 1;

else

cout << 0;

cout << endl;

}

}

CÂY BIỂU THỨC 2

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11002/top)

Cho một cây biểu thức là một cây nhị phân đầy đủ bao gồm các phép toán +, -, \*. / và một số toán hạng có giá trị nguyên. Nhiệm vụ của bạn là hãy tính toán giá trị biểu thức được biểu diễn trên cây nhị phân đầy đủ. Ví dụ với cây dưới đây là biểu diễn của biểu thức P = ( (5\*4) + (100-20)) sẽ cho ta giá trị là 100.

**Input:**

* Dòng đầu tiên đưa vào số lượng bộ test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test là gồm hai dòng: dòng thứ nhất đưa vào N là số lượng node của cây; dòng thứ hai đưa vào nội dung các node của cây; các node được viết cách nhau một vài khoảng trống. Các số có giá trị nguyên không vượt quá 1000.
* T, N, P thỏa mãn ràng buộc : 1≤T≤100; 1≤N, lenght(P)≤100.

**Output:**

* Đưa ra giá trị của cây biểu thức.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 7 + \* - 5 4 100 20 3 - 4 7 | 100 -3 |

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

struct node{

string val;

node \*left, \*right;

};

node \*newNode(string s){

node \*tmp = new node;

tmp->val = s;

tmp->left = tmp->right = NULL;

return tmp;

}

bool isOperator(string c) {

if(c == "+" || c == "-" || c == "\*" || c == "/")

return true;

return false;

}

int gtri(string s){

int res = 0, i = 0;

if(s[0] == '-') i++;

for(i = 0; i < s.size(); i++) res = res\*10 + (s[i]-'0');

if(s[0] == '-') return -res;

return res;

}

int kq(node \*t){

if(t == NULL) return 0;

if(t->left == 0 && t->right == 0) return gtri(t->val);

int l = kq(t->left);

int r = kq(t->right);

if(t->val == "+") return l+r;

if(t->val == "-") return l-r;

if(t->val == "\*") return l\*r;

return l/r;

}

main(){

int t; cin >> t;

while(t--){

int n; cin >> n;

string s[n];

int i;

for(i = 0; i < n; i++){

cin >> s[i];

}

i=0;

node \*t = newNode(s[i++]);

queue<node\*> q;

q.push(t);

while(!q.empty()){

node \*t1, \*t2, \*t3;

t1 = q.front(); q.pop();

if(isOperator(t1 -> val)){

t2 = newNode(s[i++]);

t3 = newNode(s[i++]);

t1 -> left = t2;

t1 -> right = t3;

q.push(t2);

q.push(t3);

}

}

cout << kq(t) << endl;

}

}

DUYỆT CÂY 1

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11003/top)

Cho phép duyệt cây nhị phân Inorder và Preorder, hãy đưa ra kết quả phép duyệt Postorder của cây nhị phân. Ví dụ với cây nhị phân có các phép duyệt cây nhị phân của cây dưới đây:

Inorder     : 4  2  5  1  3  6

Preorder:  : 1  2  4  5  3  6

Postorder : 4  5  2  6  3  1

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng đầu tiên đưa vào số N là số lượng node; dòng tiếp theo đưa vào N số theo phép duyệt Inorder; dòng cuối cùng đưa vào N số là kết quả của phép duyệt Preorder; các số được viết cách nhau một vài khoảng trống.
* T, N, node thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤1000; 1≤ giá trị node ≤104;

**Output:**

* Đưa ra kết quả phép duyệt Postorder theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1  6 4  2  5  1  3  6  1  2  4  5  3  6 | 4  5  2  6  3  1 |

#include<bits/stdc++.h>

using namespace std;

int search(int arr[], int x, int n){

for (int i=0; i<n; i++)

if (arr[i] == x)

return i;

return -1;

}

void PrintPostOrder(int inOrder[], int preOrder[], int n){

int root = search(inOrder, preOrder[0], n);

if (root != 0)

PrintPostOrder(inOrder, preOrder+1, root);

if (root != n-1)

PrintPostOrder(inOrder+root+1, preOrder+root+1, n-root-1);

cout << preOrder[0] << " ";

}

int main(){

int t; cin >> t;

while (t--){

int n; cin >> n;

int \*inOrder = new int[n+1];

int \*preOrder = new int[n+1];

for (int i=0; i<n; i++) cin >> inOrder[i];

for (int i=0; i<n; i++) cin >> preOrder[i];

PrintPostOrder(inOrder, preOrder, n);

cout << endl;

}

}

DUYỆT CÂY THEO MỨC

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11004/top)

Cho cây nhị phân, nhiệm vụ của bạn là duyệt cây theo Level-order. Phép duyệt level-order trên cây là phép thăm node theo từng mức của cây. Ví dụ với cây dưới đây sẽ cho ta kết quả của phép duyệt level-order: 20  8  22  4  12  10  14.

Diagram

Description automatically generated

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
* T, N, u, v, thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤u, v≤104;

**Output:**

* Đưa ra kết quả phép duyệt level-order theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 2 1 2 R 1 3 L 4 10 20 L 10 30 R 20 40 L 20 60 R | 1 3 2  10 20 30 40 60 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

Node \*ConstructTree(int n){

map<int, Node\*> m;

Node \*root = NULL;

while (n--){

int a, b; char c;

cin >> a >> b >> c;

Node \*parent;

if (m.find(a) == m.end()){

parent = CreateNode(a);

m[a] = parent;

if (root == NULL)

root = parent;

}

else

parent = m[a];

Node \*child = CreateNode(b);

if (c == 'L')

parent->left = child;

else

parent->right = child;

m[b] = child;

}

return root;

}

void solve(Node \*root){

queue<Node \*> q;

q.push(root);

while (!q.empty()){

Node \*node = q.front();

cout << node->data << " ";

q.pop();

if (node->left != NULL)

q.push(node->left);

if (node->right != NULL)

q.push(node->right);

}

}

int main(){

int t; cin >> t;

while (t--){

int n; cin >> n;

Node \*root = ConstructTree(n);

solve(root);

cout << endl;

}

}

DUYỆT CÂY 2

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11005/top)

Cho hai mảng là phép duyệt Inorder và Level-order, nhiệm vụ của bạn là xây dựng cây nhị phân và đưa ra kết quả phép duyệt Postorder. Level-order là phép duyệt theo từng mức của cây. Ví dụ như cây dưới đây ta có phép Inorder và Level-order như dưới đây:

Inorder : 4  8  10  12 14 20 22

Level order: 20  8  22  4  12  10  14

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng đầu tiên đưa vào số N là số lượng node; dòng tiếp theo đưa vào N số là phép duyệt Inorder; dòng cuối cùng đưa vào N số là phép duyệt Level-order; các số được viết cách nhau một vài khoảng trống.
* T, N, node thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤A[i]≤104;

**Output:**

* Đưa ra kết quả phép duyệt Postorder theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 3 1 0 2  0 1 2  7 3 1 4 0 5 2 6  0 1 2 3 4 5 6 | 1 2 0  3 4 1 5 6 2 0 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

Node \*ConstructTree(int inorder[], int levelorder[], int start, int end, int n){

if (n <= 0)

return NULL;

Node \*root = CreateNode(levelorder[0]);

int index = -1;

for (int i=start; i<=end; i++){

if (levelorder[0] == inorder[i]){

index = i;

break;

}

}

unordered\_set<int> s;

for (int i=start; i<index; i++) //insert left nodes

s.insert(inorder[i]);

int ls[s.size()]; //left

int rs[end-start-s.size()]; //right

int li = 0, ri = 0;

for (int i=1; i<n; i++){

if (s.find(levelorder[i]) != s.end()) // not exist

ls[li++] = levelorder[i];

else

rs[ri++] = levelorder[i];

}

root->left = ConstructTree(inorder, ls, start, index-1, index-start);

root->right = ConstructTree(inorder, rs, index+1, end, end-index);

return root;

}

void printPostorder(Node \*root){

if(root){

printPostorder(root->left);

printPostorder(root->right);

cout << root->data << " ";

}

}

int main(){

int t; cin >> t;

while (t--){

int n; cin >> n;

int \*inorder = new int[n+1];

int \*levelorder = new int[n+1];

for (int i=0; i<n; i++) cin >> inorder[i];

for (int i=0; i<n; i++) cin >> levelorder[i];

Node \*root = ConstructTree(inorder, levelorder, 0, n-1, n);

printPostorder(root);

cout << endl;

}

}

 DUYỆT CÂY NHỊ PHÂN TÌM KIẾM 1

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11017/top)

Cho mảng A[] gồm N node là biểu diễn phép duyệt theo thứ tự giữa (Preorder) của cây nhị phân tìm kiếm. Nhiệm vụ của bạn là đưa ra phép duyệt theo thứ tự sau của cây nhị phân tìm kiếm.

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng node; dòng tiếp theo đưa vào N số A[i]; các số được viết cách nhau một vài khoảng trống.
* T, N, node thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤A[i]≤104;

**Output:**

* Đưa ra kết quả phép duyệt Postorder theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 5 40 30 35 80 100 8 40 30 32 35 80 90 100 120 | 35 30 100 80 40 35 32 30 120 100 90 80 40 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

typedef Node \*Tree;

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

void InsertNode(Tree &root, int x){

if (root){

if (root->data == x) return;

else{

if (x < root->data) InsertNode(root->left, x);

else InsertNode(root->right, x);

}

}

else{

Node \*p = CreateNode(x);

root = p;

}

}

void CreateTree(Tree &root){

int n; cin >> n;

int x;

while (n--){

cin >> x;

InsertNode(root, x);

}

}

void solve(Tree root){

if(root){

solve(root->left);

solve(root->right);

cout<< root->data << " ";

}

}

int main(){

int t; cin >> t;

while (t--){

Tree root;

root = NULL;

CreateTree(root);

solve(root);

cout << endl;

}

}

DUYỆT CÂY KIỂU XOẮN ỐC

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11006/top)

Cho cây nhị phân, nhiệm vụ của bạn là duyệt cây theo xoắn ốc (spiral-order). Phép. Ví dụ với cây dưới đây sẽ cho ta kết quả của phép duyệt spiral-order: 1 2 3  4 5 6 7.

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
* T, N, u, v, thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤u, v≤104;

**Output:**

* Đưa ra kết quả phép duyệt level-order theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 2 1 2 R 1 3 L 4 10 20 L 10 30 R 20 40 L 20 60 R | 1 3 2  10 0 30 60 40 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

Node \*ConstructTree(int n){

map<int, Node\*> m;

Node \*root = NULL;

while (n--){

int a, b; char c;

cin >> a >> b >> c;

Node \*parent;

if (m.find(a) == m.end()){

parent = CreateNode(a);

m[a] = parent;

if (root == NULL)

root = parent;

}

else

parent = m[a];

Node \*child = CreateNode(b);

if (c == 'L')

parent->left = child;

else

parent->right = child;

m[b] = child;

}

return root;

}

void solve(Node \*root){

stack<Node \*> s1; //right to left

stack<Node \*> s2; //left to right

s1.push(root);

while (!s1.empty() || !s2.empty()){

while (!s1.empty()){

Node \*temp = s1.top();

s1.pop();

cout << temp->data << " ";

if (temp->right)

s2.push(temp->right);

if (temp->left)

s2.push(temp->left);

}

while (!s2.empty()){

Node \*temp = s2.top();

s2.pop();

cout << temp->data << " ";

if (temp->left)

s1.push(temp->left);

if (temp->right)

s1.push(temp->right);

}

}

}

int main(){

int t; cin >> t;

while (t--){

int n; cin >> n;

Node \*root = ConstructTree(n);

solve(root);

cout << endl;

}

}

CÂY NHỊ PHÂN HOÀN HẢO

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11010/top)

Cho cây nhị phân, nhiệm vụ của bạn là kiểm tra xem cây nhị phân có phải là một cây hoàn hảo hay không (perfect tree)? Một cây nhị phân được gọi là cây hoàn hảo nếu tất cả các node trung gian của nó đều có hai node con và tất cả các node lá đều có cùng một mức.

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 2 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
* T, N, u, v, thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤u, v≤104;

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 6 10 20 L 10 30 R 20 40 L 20 50 R 30 60 L 30 70 R 2 18 15 L 18 30 R 5 1 2 L 2 4 R 1 3 R 3 5 L 3 6 R | Yes Yes No |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

Node \*ConstructTree(int n){

map<int, Node\*> m;

Node \*root = NULL;

while (n--){

int a, b; char c;

cin >> a >> b >> c;

Node \*parent;

if (m.find(a) == m.end()){

parent = CreateNode(a);

m[a] = parent;

if (root == NULL)

root = parent;

}

else

parent = m[a];

Node \*child = CreateNode(b);

if (c == 'L')

parent->left = child;

else

parent->right = child;

m[b] = child;

}

return root;

}

int findDepth(Node \*node){

int depth = 0;

while (node != NULL){

depth++;

node = node->left;

}

return depth;

}

bool isPerfectTree(Node \*root, int depth, int level=0){

if (root == NULL)

return true;

if (root->left==NULL && root->right==NULL)

return (depth == level+1);

if (root->left==NULL || root->right==NULL)

return false;

return (isPerfectTree(root->left, depth, level+1) && isPerfectTree(root->right, depth, level+1));

}

int main(){

int t; cin >> t;

while (t--){

int n; cin >> n;

Node \*root = ConstructTree(n);

int depth = findDepth(root);

if (isPerfectTree(root, depth))

cout << "Yes";

else

cout << "No";

cout << endl;

}

}

CÂY NHỊ PHÂN BẰNG NHAU

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11012/top)

Cho hai cây nhị phân, nhiệm vụ của bạn là kiểm tra xem cây nhị phân có giống nhau hay không?

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái của mỗi cây; u, v, x được viết cách nhau một vài khoảng trống.
* T, N, u, v, thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤u, v≤104;

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 2 1 2 L 1 3 R 2 1 2 L 1 3 R 2 1 2 L 1 3 R 2 1 3 L 1 2 R | 1 0 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

Node \*ConstructTree(int n){

map<int, Node\*> m;

Node \*root = NULL;

while (n--){

int a, b; char c;

cin >> a >> b >> c;

Node \*parent;

if (m.find(a) == m.end()){

parent = CreateNode(a);

m[a] = parent;

if (root == NULL)

root = parent;

}

else

parent = m[a];

Node \*child = CreateNode(b);

if (c == 'L')

parent->left = child;

else

parent->right = child;

m[b] = child;

}

return root;

}

bool isEqualTrees(Node \*root1, Node \*root2){

if (root1==NULL && root2==NULL)

return true;

if (root1!=NULL && root2!=NULL)

return(root1->data==root2->data && isEqualTrees(root1->left, root2->left)

&& isEqualTrees(root1->right, root2->right));

return false;

}

int main(){

int t; cin >> t;

while (t--){

int n; cin >> n;

Node \*root1 = ConstructTree(n);

int m; cin >> m;

Node \*root2 = ConstructTree(m);

if (isEqualTrees(root1, root2))

cout << 1;

else

cout << 0;

cout << endl;

}

}

TỔNG NODE LÁ BÊN PHẢI

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11014/top)

Cho cây nhị phân, nhiệm vụ của bạn là tính tổng của tất cả các node lá bên phải trên cây? Ví dụ với cây dưới đây ta có kết quả là 2.

Diagram

Description automatically generated with low confidence

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
* T, N, u, v, thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤u, v≤104;

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 2 1 2 L 1 3 R 5 10 20 L 10 30 R 20 40 L 20 60 R 30 90 L | 3 60 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

Node \*ConstructTree(int n){

map<int, Node\*> m;

Node \*root = NULL;

while (n--){

int a, b; char c;

cin >> a >> b >> c;

Node \*parent;

if (m.find(a) == m.end()){

parent = CreateNode(a);

m[a] = parent;

if (root == NULL)

root = parent;

}

else

parent = m[a];

Node \*child = CreateNode(b);

if (c == 'L')

parent->left = child;

else

parent->right = child;

m[b] = child;

}

return root;

}

void leftSum(Node \*root, Node \*parent, int &sum){

if (root){

leftSum(root->left, root, sum);

if (!root->left && !root->right && parent->right==root)

sum += root->data;

leftSum(root->right, root, sum);

}

}

int main(){

int t; cin >> t;

while (t--){

int n; cin >> n;

Node \*root = ConstructTree(n);

int sum = 0;

leftSum(root, root, sum);

cout << sum << endl;

}

}

TỔNG LỚN NHẤT

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11015/top)

Cho cây nhị phân có giá trị mỗi node là một số, nhiệm vụ của bạn là tìm tổng lớn nhất từ một node lá này sang một node lá khác? Ví dụ với cây dưới đây ta có tổng lớn nhất là 27.

Diagram

Description automatically generated

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
* T, N, u, v, thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤u, v≤104;

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 12 -15 5 L -15 6 R 5 -8 L 5 1 R -8 2 L -8 -3 R 6 3 L 6 9 R 9 0 R 0 4 L 0 -1 R -1 10 L | 27 |
|  |  |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

void Insert(Node \*root, int a, int b, char c){

if(a == root->data){

if(c == 'L'){

root->left = CreateNode(b);

return;

}

if(c=='R'){

root->right= CreateNode(b);

return;

}

}

else if(root->left==NULL && root->right==NULL)

return;

else{

if(root->left)

Insert(root->left, a, b, c);

if(root->right)

Insert(root->right, a, b, c);

}

}

int maxSum(Node \*root, int &res){

if (!root) return 0;

if (!root->left && !root->right) return root->data;

int ls = maxSum(root->left, res);

int rs = maxSum(root->right, res);

if (root->left && root->right){

res = max(res, ls+rs+root->data);

return max(ls, rs) + root->data;

}

if (!root->left)

return rs + root->data;

else

return ls + root->data;

}

int Result(Node \*root){

int res = INT\_MIN;

maxSum(root, res);

return res;

}

int main(){

int t; cin >> t;

while (t--){

int n; cin >> n;

Node \*root = NULL;

while (n--){

int a, b; char c;

cin >> a >> b >> c;

if (root == NULL)

root = CreateNode(a);

Insert(root, a, b, c);

}

cout << Result(root) << endl;

}

}

 BIẾN ĐỔI SANG CÂY NHỊ PHÂN TÌM KIẾM

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11016/top)

Cho cây nhị phân, nhiệm vụ của bạn là dịch chuyển cây nhị phân thành cây nhị phân tìm kiếm. Phép dịch chuyển phải bảo toàn được cấu trúc cây nhị phân ban đầu. Ví dụ dưới đây sẽ minh họa phép dịch chuyển:

A set of medical tools

Description automatically generated with low confidence

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng đầu tiên đưa vào số N là số lượng cạnh của cây; dòng tiếp theo đưa vào N bộ ba (u, v, x), trong đó u là node cha, v là node con, x= R nếu v là con phải, x=L nếu v là con trái; u, v, x được viết cách nhau một vài khoảng trống.
* T, N, u, v, thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤u, v≤104;

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng là phép duyệt Inorder của cây tìm kiếm.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 2 1 2 R 1 3 L 4 10 20 L 10 30 R 20 40 L 20 60 R | 1 2 3 10 20 30 40 60 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

typedef Node \*Tree;

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

Node \*ConstructTree(int n){

map<int, Node\*> m;

Node \*root = NULL;

while (n--){

int a, b; char c;

cin >> a >> b >> c;

Node \*parent;

if (m.find(a) == m.end()){

parent = CreateNode(a);

m[a] = parent;

if (root == NULL)

root = parent;

}

else

parent = m[a];

Node \*child = CreateNode(b);

if (c == 'L')

parent->left = child;

else

parent->right = child;

m[b] = child;

}

return root;

}

void store(Node \*root, auto &set){

if (!root)

return;

store(root->left, set);

set.insert(root->data);

store(root->right, set);

}

void convert(Node \*root, auto &it){

if (!root)

return;

convert(root->left, it);

root->data = \*it;

it++;

convert(root->right, it);

}

void inorder(Node \*root){

if (root){

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

}

int main(){

int t; cin >> t;

while (t--){

int n; cin >> n;

Node \*root = ConstructTree(n);

set<int> set;

store(root, set);

auto it = set.begin();

convert(root, it);

inorder(root);

cout << endl;

}

}

XÂY DỰNG LẠI CÂY NHỊ PHÂN TÌM KIẾM

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11018/top)

Cho một mảng là phép duyệt level-order của cây nhị phân tìm kiếm. Nhiệm vụ của bạn là xây dựng lại cây nhị phân tìm kiếm bảo toàn được cấu trúc cây nhị phân ban đầu.

**Input:**

* Dòng đầu tiên đưa vào số lượng test T.
* Những dòng tiếp theo đưa vào các bộ test. Mỗi bộ test gồm dòng: dòng đầu tiên đưa vào số N là số lượng node của cây tìm kiếm; dòng tiếp theo đưa vào phép duyệt level-order của cây tìm kiếm; các số được viết cách nhau một vài khoảng trống.
* T, N, node thỏa mãn ràng buộc: 1≤T≤100; 1≤N≤103; 1≤node≤104;

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng là phép duyệt trước (preOrder) của cây tìm kiếm.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 9 7 4 12 3 6 8 1 5 10 6 1 3 4 6 7 8 | 7 4 3 1 6 5 12 8 10 1 3 4 6 7 8 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

typedef Node \*Tree;

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

void InsertNode(Tree &root, int x){

if (root){

if (root->data == x) return;

else{

if (x < root->data) InsertNode(root->left, x);

else InsertNode(root->right, x);

}

}

else{

Node \*p = CreateNode(x);

root = p;

}

}

void CreateTree(Tree &root){

int n; cin >> n;

int x;

while (n--){

cin >> x;

InsertNode(root, x);

}

}

void solve(Tree root){

if(root){

cout<< root->data << " ";

solve(root->left);

solve(root->right);

}

}

int main(){

int t; cin >> t;

while (t--){

Tree root;

root = NULL;

CreateTree(root);

solve(root);

cout << endl;

}

}

NODE LÁ CỦA CÂY NHỊ PHÂN TÌM KIẾM

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11021/top)

Cho dãy số gồm N số là phép duyệt theo thứ tự trước (Preorder) của một cây nhị phân tìm kiếm. Hãy in ra tất cả các node lá của cây ?

Ví dụ với dãy A[] = {30, 20, 15, 25, 23, 28, 40, 35, 33, 38, 45} là phép duyệt cây theo thứ tự trước sẽ cho ta kết quả: 15, 23, 28, 33, 38, 45.

**Input:**

* Dòng đầu tiên đưa vào số lượng bộ test T (T≤100).
* Những dòng kế tiếp đưa vào T bộ test. Mỗi bộ test gồm 2 dòng: dòng thứ nhất là số tự nhiên N (N≤1000). Dòng tiếp theo là N số là phép duyệt theo thứ tự trước của cây BST.

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  6  10 5 1 7 40 50  11  30 20 15 25 23 28 40 35 33 38 45 | 1 7 50  15 23 28 33 38 45 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

typedef Node \*Tree;

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

void InsertNode(Tree &root, int x){

if (root){

if (root->data == x) return;

else{

if (x < root->data) InsertNode(root->left, x);

else InsertNode(root->right, x);

}

}

else{

Node \*p = CreateNode(x);

root = p;

}

}

void CreateTree(Tree &root){

int n; cin >> n;

int x;

while (n--){

cin >> x;

InsertNode(root, x);

}

}

void printLeafNodes(Tree root){

if (!root)

return;

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

cout << root->data << " ";

return;

}

if (root->left)

printLeafNodes(root->left);

if (root->right)

printLeafNodes(root->right);

}

int main(){

int t; cin >> t;

while (t--){

Tree root;

root = NULL;

CreateTree(root);

printLeafNodes(root);

cout << endl;

}

}

NODE TRUNG GIAN CỦA CÂY NHỊ PHÂN TÌM KIẾM

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11022/top)

Cho dãy số gồm N số là phép duyệt theo thứ tự trước (Preorder) của một cây nhị phân tìm kiếm. Hãy đưa ra số các node trung gian của cây ?

Ví dụ với dãy A[] = {30, 20, 15, 25, 23, 28, 40, 35, 33, 38, 45} là phép duyệt cây theo thứ tự trước sẽ cho ta kết quả là 5 bao gồm các node: 30, 20, 25, 40, 35.

**Input:**

* Dòng đầu tiên đưa vào số lượng bộ test T (T≤100).
* Những dòng kế tiếp đưa vào T bộ test. Mỗi bộ test gồm 2 dòng: dòng thứ nhất là số tự nhiên N (N≤1000). Dòng tiếp theo là N số là phép duyệt theo thứ tự trước của cây BST.

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input:** | **Output** |
| 2  6  10 5 1 7 40 50  11  30 20 15 25 23 28 40 35 33 38 45 | 3  5 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

typedef Node \*Tree;

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

void InsertNode(Tree &root, int x){

if (root){

if (root->data == x) return;

else{

if (x < root->data) InsertNode(root->left, x);

else InsertNode(root->right, x);

}

}

else{

Node \*p = CreateNode(x);

root = p;

}

}

void CreateTree(Tree &root){

int n; cin >> n;

int x;

while (n--){

cin >> x;

InsertNode(root, x);

}

}

void solve(Tree root, int &c){

if (root){

solve(root->left, c);

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

c++;

solve(root->right, c);

}

}

int main(){

int t; cin >> t;

while (t--){

Tree root;

root = NULL;

CreateTree(root);

int c = 0;

solve(root, c);

cout << c << endl;

}

}

ĐỘ SÂU CÂY NHỊ PHÂN TÌM KIẾM

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11023/top)

Cho dãy số gồm N số là phép duyệt theo thứ tự trước (Preorder) của một cây nhị phân tìm kiếm. Hãy tìm độ sâu của cây ?

Ví dụ với dãy A[] = {30, 20, 15, 25, 23, 28, 40, 35, 33, 38, 45} là phép duyệt cây theo thứ tự trước sẽ cho ta kết quả là 3.

**Input:**

* Dòng đầu tiên đưa vào số lượng bộ test T (T≤100).
* Những dòng kế tiếp đưa vào T bộ test. Mỗi bộ test gồm 2 dòng: dòng thứ nhất là số tự nhiên N (N≤1000). Dòng tiếp theo là N số là phép duyệt theo thứ tự trước của cây BST.

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input:** | **Output** |
| 2  6  10 5 1 7 40 50  11  30 20 15 25 23 28 40 35 33 38 45 | 2  3 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

typedef Node \*Tree;

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

void InsertNode(Tree &root, int x){

if (root){

if (root->data == x) return;

else{

if (x < root->data) InsertNode(root->left, x);

else InsertNode(root->right, x);

}

}

else{

Node \*p = CreateNode(x);

root = p;

}

}

void CreateTree(Tree &root){

int n; cin >> n;

int x;

while (n--){

cin >> x;

InsertNode(root, x);

}

}

int MaxDepth(Tree root){

if (!root) return -1;

return max(MaxDepth(root->left), MaxDepth(root->right))+1;

}

int main(){

int t; cin >> t;

while (t--){

Tree root;

root = NULL;

CreateTree(root);

cout << MaxDepth(root) << endl;

}

}

 CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG 1

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11024/top)

Hãy xây dựng một cây nhị phân tìm kiếm cân bằng từ dãy số A[] =(a0, a1, .., an-1}. Đưa ra nội dung node gốc của cây tìm kiếm cân bằng.  Ví dụ với dãy A[]={40, 28, 45, 38, 33, 15, 25, 20, 23, 35, 30} ta sẽ có cây nhị phân tìm kiếm cân bằng với node gốc là 33.

**Input:**

* Dòng đầu tiên đưa vào số lượng bộ test T (T≤100).
* Những dòng kế tiếp đưa vào T bộ test. Mỗi bộ test gồm 2 dòng: dòng thứ nhất là số tự nhiên N (N≤106). Dòng tiếp theo là N số của mảng A[].

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  11  40  28  45 38  33  15  25  20  23  35  30  10  1  2  3  4  5  6  7  8  9  10 | 30  5 |

#include<bits/stdc++.h>

using namespace std;

int main(){

int t; cin >> t;

while (t--){

int n, \*arr; cin >> n;

arr = new int[n+1];

for (int i=0; i<n; i++) cin >> arr[i];

sort(arr, arr+n);

int mid = (n-1)/2;

int root = arr[mid];

cout << root << endl;

}

}

 CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG 2

[Bài làm tốt nhất](https://code.ptit.edu.vn/student/question/DSA11025/top)

Hãy xây dựng một cây nhị phân tìm kiếm cân bằng từ dãy số A[] =(a0, a1, .., an-1}. Đưa ra phép duyệt theo thứ tự trước (preorder) của cây tìm kiếm cân bằng.  Ví dụ với dãy A[]={40, 28, 45, 38, 33, 15, 25, 20, 23, 35, 30} ta sẽ có phép duyệt theo thứ tự trước của cây nhị phân tìm kiếm cân bằng với node gốc là 33 : 33, 25, 20, 15, 23, 28, 30, 40, 38, 35, 45.

**Input:**

* Dòng đầu tiên đưa vào số lượng bộ test T (T≤100).
* Những dòng kế tiếp đưa vào T bộ test. Mỗi bộ test gồm 2 dòng: dòng thứ nhất là số tự nhiên N (N≤106). Dòng tiếp theo là N số của mảng A[].

**Output:**

* Đưa ra kết quả mỗi test theo từng dòng.

**Ví dụ:**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  11  40  28  45 38  33  15  25  20  23  35  30  10  1  2  3  4  5  6  7  8  9  10 | 30 23 15 20 25 28 38 33 35 40 45  5 2 1 3 4 8 6 7 9 10 |

#include<bits/stdc++.h>

using namespace std;

struct Node{

int data;

Node \*left, \*right;

};

Node \*CreateNode(int x){

Node \*p = new Node();

if (!p) exit(1);

p->data = x;

p->right = p->left = NULL;

return p;

}

Node \*binarySearch(int arr[], int s, int e){

if (s > e)

return NULL;

int mid = (s+e)/2;

Node \*root = CreateNode(arr[mid]);

root->left = binarySearch(arr, s, mid-1);

root->right = binarySearch(arr, mid+1, e);

return root;

}

void preOrder(Node \*root){

if (root){

cout << root->data << " ";

preOrder(root->left);

preOrder(root->right);

}

}

int main(){

int t; cin >> t;

while (t--){

int n, \*arr; cin >> n;

arr = new int[n+1];

for (int i=0; i<n; i++) cin >> arr[i];

sort(arr, arr+n);

Node \*root = binarySearch(arr, 0, n-1);

preOrder(root);

cout << endl;

}

}