Splay tree

#include <iostream>

#include <math.h>

#include <queue>

using namespace std;

#define SEPARATOR "#<ab@17943918#@>#"

enum BalanceValue

{

LH = -1,

EH = 0,

RH = 1

};

void printNSpace(int n)

{

for (int i = 0; i < n - 1; i++)

cout << " ";

}

void printInteger(int &n)

{

cout << n << " ";

}

class SplayTree {

struct Node {

int val;

Node\* pLeft;

Node\* pRight;

Node\* pParent;

Node(int val = 0, Node\* l = nullptr, Node\* r = nullptr, Node\* par = nullptr) : val(val), pLeft(l), pRight(r), pParent(par) { }

};

Node\* root;

// print the tree structure for local testing

void printBinaryTree(string prefix, const Node\* root, bool isLeft, bool hasRightSibling) {

if (!root && isLeft && hasRightSibling) {

cout << prefix << "├──\n";

}

if (!root) return;

cout << prefix;

if (isLeft && hasRightSibling)

cout << "├──";

else

cout << "└──";

cout << root->val << '\n';

printBinaryTree(prefix + (isLeft && hasRightSibling ? "| " : " "), root->pLeft, true, root->pRight);

printBinaryTree(prefix + (isLeft && hasRightSibling ? "| " : " "), root->pRight, false, root->pRight);

}

void printPreorder(Node\* p) {

if (!p) {

return;

}

cout << p->val << ' ';

printPreorder(p->pLeft);

printPreorder(p->pRight);

}

public:

SplayTree() {

root = nullptr;

}

~SplayTree() {

// Ignore deleting all nodes in the tree

}

void printBinaryTree() {

printBinaryTree("", root, false, false);

}

void printPreorder() {

printPreorder(root);

cout << "\n";

}

void left\_rotate( Node \*x)

{

Node \*y = x->pRight;

x->pRight = y->pLeft;

if(y->pLeft != NULL) {

y->pLeft->pParent = x;

}

y->pParent = x->pParent;

if(x->pParent == NULL) { //x is root

this->root = y;

}

else if(x == x->pParent->pLeft) { //x is pLeft child

x->pParent->pLeft = y;

}

else { //x is pRight child

x->pParent->pRight = y;

}

y->pLeft = x;

x->pParent = y;

}

void right\_rotate(Node \*x)

{

Node \*y = x->pLeft;

x->pLeft = y->pRight;

if(y->pRight != NULL) {

y->pRight->pParent = x;

}

y->pParent = x->pParent;

if(x->pParent == NULL) { //x is root

this->root = y;

}

else if(x == x->pParent->pRight) { //x is pLeft child

x->pParent->pRight = y;

}

else { //x is pRight child

x->pParent->pLeft = y;

}

y->pRight = x;

x->pParent = y;

}

void splay(Node\* p) {

// To Do

if(p==nullptr)

{

return;

}

if(p->pParent==nullptr)

{

return;

}

while(p->pParent != NULL)

{ //Node is not root

if(p->pParent == this->root)

{ //Node is child of root, one rotation

if(p == p->pParent->pLeft) {

right\_rotate( p->pParent);

}

else {

left\_rotate(p->pParent);

}

}

else {

Node \*pa = p->pParent;

Node \*gr = pa->pParent; //grandparent

if(p->pParent->pLeft == p && pa->pParent->pLeft == pa)

{ //both are pLeft children

right\_rotate(gr);

right\_rotate(pa);

}

else if(p->pParent->pRight == p && pa->pParent->pRight == pa)

{ //both are pRight children

left\_rotate(gr);

left\_rotate( pa);

}

else if(p->pParent->pRight == p && pa->pParent->pLeft == pa)

{

left\_rotate(pa);

right\_rotate(gr);

}

else if(p->pParent->pLeft == p && pa->pParent->pRight == pa) {

right\_rotate(pa);

left\_rotate( gr);

}

}

}

}

void insert(int val) {

if(this->root==nullptr)

{

this->root=new Node(val);

return;

}

Node \*newnode=new Node(val);

Node \*y=nullptr,\*p=this->root;

while(p!=nullptr)

{

y=p;

if(p->val<=val)

{

p=p->pRight;

}

else

{

p=p->pLeft;

}

}

newnode->pParent=y;

if(y->val<=val)

{

y->pRight=newnode;

}

else

{

y->pLeft=newnode;

}

splay(newnode);

}

};

int main()

{

SplayTree tree;

int query;

cin >> query;

for(int i = 0; i < query; i++) {

string op;

int val;

cin >> op >> val;

if (op == "insert")

tree.insert(val);

}

// print preorder traversal of the tree

tree.printPreorder();

// print structure of the tree

tree.printBinaryTree();

system("pause");

}