\*\*\*\*\*\*\*\*\*\*\*\*BST的实现与动态查找\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. **BSTNode.h:**

#include <iostream>

typedef int E;

class BSTNode{

private:

E data; //二叉树节点存储的值

BSTNode\* lc;

BSTNode\* rc; //左右孩子指针

public:

BSTNode() //初始化节点

{

lc=rc=NULL;

}

BSTNode(E e,BSTNode\* l=NULL,BSTNode\* r=NULL) //构建一个非空节点

{

data=e;lc=l;rc=r;

}

E getData(){ return data;}

BSTNode\* left(){ return lc; }

BSTNode\* right(){ return rc; }

void setLeft(BSTNode\* b){lc=b;}

void setRight(BSTNode\* b){rc=b;}

};

1. **BST.h：**

#include "BSTNode.h"

#include <assert.h>

#include <stdlib.h>

using namespace std;

int count=0;

class BST:public BSTNode{

private:

BSTNode\* root; // Root of the BST BST树的根节点

int nodeCount;

public:

BST() {root = NULL; nodeCount = 0;} //初始化一棵空树

BSTNode\* getRoot() {return root;}

BSTNode\* inserthelp(BSTNode\*, const E&);

BSTNode\* removehelp(BSTNode\*, const E&);

BSTNode\* deletemin(BSTNode\*);

BSTNode\* getmin(BSTNode\*);

void printhelp(BSTNode\*, int);

bool findhelp(BSTNode\* ,const E& );

int nodecount(){return nodeCount;}

void clearhelp(BSTNode\* root);

void clear() // Reinitialize tree 清空树

{

clearhelp(root);

root = NULL;

nodeCount = 0;

}

void insert(E& e)

{

root = inserthelp(root,e); //调用inserthelp插入节点

nodeCount++;

}

E remove(E& e)

{

E temp = findhelp(root,e); // First find it 第一次找到该节点

if (temp != NULL)

{

root = removehelp(root,e);

nodeCount--;

}

count=0;

return temp;

}

bool find( E& e)

{

return findhelp(root,e); //调用findhelp查找

}

};

void BST::clearhelp(BSTNode\* root)

{

if (root == NULL) return;

clearhelp(root->left());

clearhelp(root->right());

delete root;

}

bool BST::findhelp(BSTNode\* root,const E& val)

{

if(root==NULL) return false;

count++;

if(val<root->getData())

return findhelp(root->left(),val);

else if(val>root->getData())

return findhelp(root->right(),val);

else

return true;

}

BSTNode\* BST::inserthelp(BSTNode\* root,const E& val)

{

if(root==NULL)

return new BSTNode(val,NULL,NULL);

if(val<root->getData())

root->setLeft(inserthelp(root->left(),val));

else

root->setRight(inserthelp(root->right(),val));

return root;

}

BSTNode\* BST::removehelp(BSTNode\* rt, const E& val) {

if (rt == NULL) return NULL; // val is not in tree 该节点不在树中

else if (val < rt->getData())

rt->setLeft(removehelp(rt->left(), val));

else if (val > rt->getData())

rt->setRight(removehelp(rt->right(), val));

else { // Found: remove it 找到了并删除它

BSTNode\* temp = rt;

if (rt->left() == NULL) { // Only a right chil 只有右孩子

rt = rt->right(); // so point to right 指向右孩子

delete temp;

}

else if (rt->right() == NULL) { // Only a left child 只有左孩子

rt = rt->left(); //so point to left 指向左孩子

delete temp;

}

else { // Both children are non-empty 左右孩子非空时

BSTNode\* temp = getmin(rt->right());

rt->setElement(temp->getData());

rt->setRight(deletemin(rt->right()));

delete temp;

}

}

return rt;

}

// Delete the minimum value from the BST 删除BST树中最小的元素

BSTNode\* BST::getmin(BSTNode\* rt)

{

if (rt->left() == NULL)

return rt;

else return getmin(rt->left());

}

BSTNode\* BST::deletemin(BSTNode\* rt)

{

if (rt->left() == NULL) // Found min 找到最小的数

return rt->right();

else { // Continue left 继续查找左子树

rt->setLeft(deletemin(rt->left()));

return rt;

}

}

1. **主函数测试部分：**

#include "BST.h"

#include <iostream>

#include <math.h>

#include <vector>

#include <queue>

using namespace std;

void creat(BST& tree,int n,vector<int> &A);

void print(int f);

int height(BSTNode \*root);

void PrintNodeByLevel(BSTNode \*root,int n);

int main()

{

int i,n,nC,val,flag,temp;

BST tree;

vector<int> A;

cout<<"本程序对BST树的基本操作进行演示"<<endl;

cout<<"输入元素的总数n，以回车结束:"<<endl;

cin>>n;

if(n<=0) {

cout<<"输入不合法"<<endl; exit(0);

}

cout<<"输入n个元素，中间以空格隔开"<<endl;

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

cin>>val;

A.push\_back(val);

}

creat(tree,n,A); //构建BST

nC=tree.nodecount();

cout<<"层次遍历输出BST树："<<endl;

PrintNodeByLevel(tree.getRoot(),nC);

cout<<"请输入要插入的值"<<endl;

cin>>val;

tree.insert(val);

nC=tree.nodecount();

cout<<"新树如下："<<endl;

PrintNodeByLevel(tree.getRoot(),nC);

cout<<"请输入要删除的值"<<endl;

cin>>val;

temp=tree.remove(val);

if(temp==1){

nC=tree.nodecount();

cout<<"新树如下："<<endl;

PrintNodeByLevel(tree.getRoot(),nC);}

else cout<<"操作失败！"<<endl;

cout<<"请输入要查找的值，ctrl+Z结束程序"<<endl;

while(cin>>val){

flag=tree.find(val); //查找函数

print(flag); //输出函数

count=0; //每次查找完毕将count复原

}

return 0;

}

void creat(BST& tree,int n,vector<int> &A){

int i;

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

tree.insert(A[i]);

}

}

void print(int f){

if(f)

cout<<"查找成功 "<<count<<endl;

else

cout<<"查找不成功 "<<count<<endl;

}

int height(BSTNode \*root){ //求树的高度

int lh,rh,h;

if(root==NULL) return -1;

lh=height(root->left());

rh=height(root->right());

h=lh>rh?lh:rh;

return (h+1);

}

void PrintNodeByLevel(BSTNode \*root,int nC){

int parentSize=1, childSize=0;

int ncount=0,level=0,i=0,j=0,k=0;

int \*A=new int[1000000]; //新建数组

BSTNode \*temp;

BSTNode \*temp1=new BSTNode(-1,NULL,NULL); //用-1的节点表示空节点入队

queue<BSTNode \*> q;

q.push(root);

do //利用队列进行层次遍历

{

temp=q.front();

if(temp->getData()!=-1)

ncount++; //记录已输出的有效节点个数

A[i++]=temp->getData(); //将节点信息存入数组中

q.pop();

if(temp->left()!=NULL){ //检查temp指针所指的父节点的左孩子

q.push(temp->left());

childSize ++;

}

else q.push(temp1);

if(temp->right()!=NULL){ //检查temp指针所指的父节点的右孩子

q.push(temp->right());

childSize ++;

}

else q.push(temp1);

parentSize--;

if(parentSize==0)

{

parentSize=childSize;

childSize=0;

}

}while (ncount!=nC);

level=height(root)+1;

for(i=0,j=0;i<level;i++) //层次地输出节点

{

k=0;

while(k<pow(2,i))

{

if(A[j]==-1)

cout<<"\* "; // ‘\*’号表示空节点

else if(A[j]==0) ; //A[j]未存储节点值时值为0，不输出

else cout<<A[j]<<" ";

k++;

j++;

}

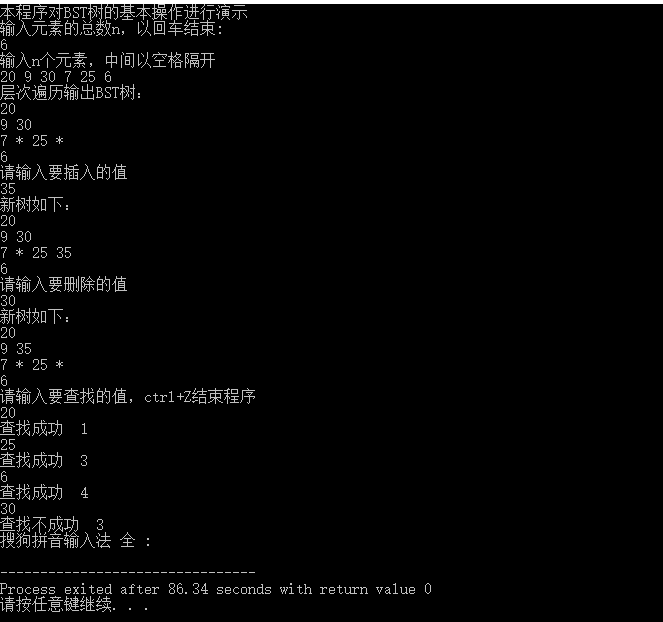
cout<<endl;

}

delete [] A; //释放数组空间

}

1. **测试结果示例：**



【附录】

版本信息声明：

Dev-C++ 5.11

TDM-GCC 4.9.2 64-bit Release

部分代码来源：

<http://people.cs.vt.edu/~shaffer/Book/>

资料整理人：  
信息科学与工程学院 物联1402班 201408080203 宁静仪

信息科学与工程学院 物联1402班 201408080222吴彦彦