**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **数据结构与算法** |
| **学生姓名：** | **林墨馨** |
| **学生学号：** | **201730682485** |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2018-2019学年第1学期** |

**软件学院**

**2018年9月**

# 二叉树的建立及相关算法的实现

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **地 点：** | | B7 楼B座 | | 房 |  |
| **实验日期与时间：** | | 2018年10月20日 上午第1-4节 | | |
| **实验教师：** | 黄敏、齐海涛 | |

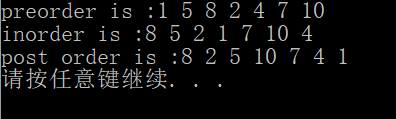
**实验内容：**

a.编程实现建立一棵二叉树，并对其进行先序、中序和后序遍历。

代码以及截图part

1. 题 //用例子建了一棵树！

运行结果图：



实验代码

#include<iostream>

using std::endl;

using std::cout;

using std::cin;

//need test

//preorder postorder inorder

template<typename T>

class binNode

{

public:

binNode();//root

binNode(const T &);//root

binNode(const T &, binNode<T> \* f);

binNode(binNode \* f);

~binNode();

binNode<T> \* setLeft(const T &);

binNode<T> \* setRight(const T &);

binNode \* LeftChild() { return leftChild; };

binNode \* RightChild() { return rightChild; };

T \* left();

T \* right();

T& element() { return Element; };

bool ifLeaf();//判断是否叶子节点

void visit();

void preorder(binNode <T> \*);

void postorder(binNode <T> \*);

void inorder(binNode <T> \*);

binNode \* leftChild, \*rightChild, \*father;

T Element;

int level;//root is level 0

};

template<typename T>

binNode<T>::binNode()

{

leftChild = rightChild = nullptr;

Element = 0;

level = 0;

father = nullptr;

}

template<typename T>

binNode<T>::binNode(const T & e)

{

level = 0;

father = nullptr;

Element = e;

leftChild = rightChild = nullptr;

}

template<typename T>

binNode<T>::binNode(const T & e, binNode<T> \* f)

{

leftChild = rightChild = nullptr;

Element = e;

father = f;

}

template<typename T>

binNode<T>::binNode(binNode \* f)

{

leftChild = rightChild = nullptr;

Element = 0;

father = f;

}

template<typename T>

binNode<T>::~binNode()//delete tree

{

if (leftChild != nullptr)delete leftChild;

if (rightChild != nullptr)delete rightChild;

}

template<typename T>

binNode<T> \* binNode<T>::setLeft(const T & e)

{

if (leftChild == nullptr)

leftChild = new binNode(e, this);

//else leftChild->setLeft(e);

else leftChild->Element = e;

leftChild->level = level + 1;

return leftChild;

}

template<typename T>

binNode<T> \* binNode<T>::setRight(const T & e)

{

if (rightChild == nullptr)

rightChild = new binNode(e, this);

//else rightChild->setRight(e);

else rightChild->Element = e;

rightChild->level = level + 1;

return rightChild;

}

template<typename T>

T \* binNode<T>::left()

{

return leftChild;

}

template<typename T>

T \* binNode<T>::right()

{

return rightChild;

}

template<typename T>

bool binNode<T>::ifLeaf()

{

if (leftChild == nullptr && rightChild == nullptr)

return true;

return false;

}

template<typename T>

void binNode<T>::visit()

{

std::cout << element() << " ";

}

template<typename T>

void binNode<T>::preorder(binNode <T> \* r)//递归的方式 访问节点

{

if (r == nullptr) return;

r->visit();

preorder(r->leftChild);

preorder(r->rightChild);

}

template<typename T>

void binNode<T>::postorder(binNode <T> \* r)

{

if (r == nullptr) return;

postorder(r->leftChild);

postorder(r->rightChild);

r->visit();

}

template<typename T>

void binNode<T>::inorder(binNode <T> \* r)

{

if (r == nullptr) return;

inorder(r->leftChild);

r->visit();

inorder(r->rightChild);

}

template<typename T>

binNode<T> \* buildTree( binNode<T> \* root)//用例子建了一棵树

{

binNode<T> \* right = root->setRight(4);

binNode<T> \* left = root->setLeft(5);

left->setRight(2);

left->setLeft(8);

right = right->setLeft(7);

right->setRight(10);

return root;

}

int main()

{

binNode<int> tree(1);

buildTree<int>(&tree);

cout << "preorder is :";

tree.preorder(&tree);

cout << endl;

cout << "inorder is :";

tree.inorder(&tree);

cout << endl;

cout << "post order is :";

tree.postorder(&tree);

cout << endl;

system("pause");

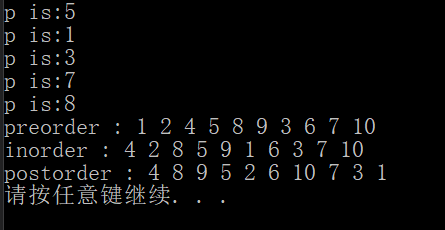
return 0;

}

b.给出二叉树的前序遍历：1 2 4 5 8 9 3 6 7 10及中序遍历：4 2 8 5 9 1 6 3 7 10，请编程实现二叉树的重构，并输出该二叉树的后序遍历。

1. 题

结果截图：



其中 p表示的是子树的根节点

代码part:

#define DEBUG

#include<iostream>

using namespace std;

template<typename T>

class binNode//树的节点类

{

public:

binNode()//root

{

Element = -1;

};

binNode(T e)

{

Element = e;

};

~binNode() {};

void visit()const

{

if (this == nullptr) return;

cout << element() << " ";

};

//三种遍历方式

void preorder(binNode<T> \* r)

{

if (r == nullptr)return;

r->visit();

preorder(r->left());

preorder(r->right());

}

void inorder(binNode<T> \* r)

{

if (r == nullptr) return;

inorder(r->left());

r->visit();

inorder(r->right());

}

void postorder(binNode <T> \* r)

{

if (r == nullptr)return;

postorder(r->left());

postorder(r->right());

r->visit();

};

binNode \* left()

{

return lc;

};

binNode \* right()

{

return rc;

};

binNode \* setLeft(const T & t)

{

lc = new binNode(t);

return lc;

}

binNode \* setLeft( binNode<T> \* ll)

{

lc = ll;

return lc;

}

binNode \* setRight(const T & t)

{

rc = new binNode(t);

return rc;

}

binNode \* setRight( binNode<T> \* rr)

{

rc = rr;

return rc;

}

T element()const

{

return Element;

}

//private:

binNode \* lc, \*rc;

T Element;

};

//封装成 binTree类

template<typename T>

class binTree

{

public:

void buildTree(T \* pre, T \* in, int len)//建树

{

root = buildhelp(root ,pre, in, len, 0, 0, len);

};

void preorder()const//前序遍历

{

root->preorder(root);

}

void postorder()const

{

root->postorder(root);

}

void inorder()const

{

root->inorder(root);

}

private:

binNode<T> \* root;

int findPosition(T \* seq,int start,int end,const T & targe)//寻找targe在seq中的位置(seq的起始位置为 start 结束位置为end [start,end))

{

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

{

if (targe == seq[i])

return i;

}

return -1;//没有找到则返回 -1

};

// r是当前树（也有可能是子树）的起始节点 pre是前序遍历的数组 in是中序遍历的数组 ,len是元素的个数

//pos是当前pre[]数组index的位置，left right表示的是中序遍历数组index的范围

binNode<T> \* buildhelp(binNode<T> \* r,T \* pre, T \* in, int len, int pos, int left, int right)//pos is the position of preorder sequence

{

if (right - left <= 1)

{

if (right - left == 1)//如果只有一个元素 那就直接new进去就行了

{

r = new binNode<T>(in[left]);

return r;

}

//如果没有元素 说明是空子树

else return nullptr;

}

//p表示的是pre[pos] 这个元素 在in数组中的位置

//以此为中界限 把in的数组划分为两个子数组 in[left->p) in[p + 1->right)

//然后再递归的建树

int p = findPosition(in, left, right, pre[pos]);

while (p == -1 && p < len && pos < len - 1)

{

p = findPosition(in, left, right,pre[++pos]);

}

//used to debug

#ifdef DEBUG

cout << "p is:" << p << endl;

#endif

//pos is the position of pre

//p is the position of in

//注意 此时in[p](也就是pre[pos])就是当前节点的值 这里要new

r = new binNode<T> (in[p]);

//递归到左孩子 和右孩子

r->setLeft(buildhelp(r->lc,pre,in,len,pos + 1,left,p));

r->setRight(buildhelp(r->rc,pre,in,len,pos + 1,p + 1,right));

return r;

};

};

int main()

{

int size = 10;

int pre[] = { 1, 2, 4, 5, 8, 9, 3, 6, 7, 10 };

int in[] = { 4, 2, 8, 5, 9, 1, 6, 3, 7, 10 };

binTree<int> tree;

tree.buildTree(pre, in, size);

cout << "preorder : ";

tree.preorder();

cout << endl;

cout << "inorder : ";

tree.inorder();

cout << endl;

cout << "postorder : ";

tree.postorder();

cout << endl;

system("pause");

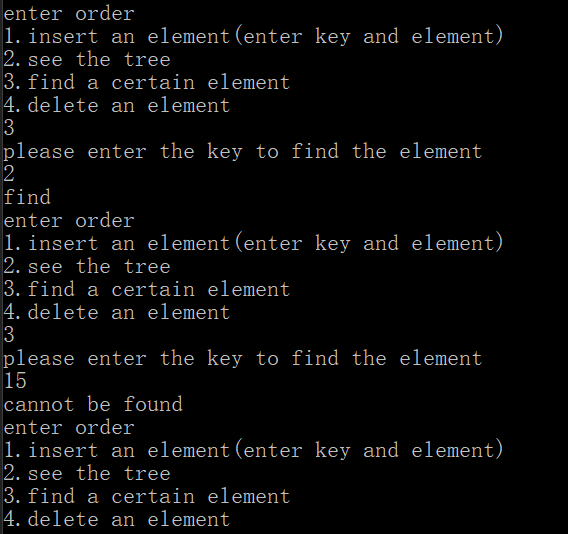
return 0;

}

c.使用你的二叉树将给定序列构建成二叉搜索树，实现查找函数。

1. 解释:通过输入命令实现功能，如图 找到则显示find 没有找到则显示cannot be found

程序运行截图:



程序片段

//need to be test

#include<stdio.h>

template <typename E>

class BinNode

{

public:

virtual ~BinNode() {};

virtual E element()const = 0;

virtual void setElement(const E &) = 0;

virtual bool isLeaf()const = 0;

};

template<typename Key,typename E>

class BSTNode : public BinNode<E>

{

public:

BSTNode()

{

lc = rc = nullptr;

};

BSTNode(Key kk, E e, BSTNode \* l = nullptr, BSTNode \* r = nullptr)

{

k = kk;

it = e;

lc = l;

rc = r;

};

~BSTNode()

{

};

//E & element() const////////////////////////////////why wrong??

E element() const

{

return it;

};

void setElement(const E & e)

{

it = e;

};

Key & key()

{

return k;

};

void setKey(const Key & K)

{

k = K;

};

inline BSTNode \* left()const

{

return lc;

}

inline BSTNode \* right()const

{

return rc;

}

void setLeft(BinNode<E> \* b)

{

//lc = (BSTNode \*)b;

lc = (BSTNode \*)b;

}

void setRight(BinNode<E> \* b)

{

rc = (BSTNode \*)b;

}

bool isLeaf()const

{

return(lc == nullptr && rc == nullptr);

}

void visit()const

{

cout << k << " " << it << endl;

}

//用递归的方式 实现前序中序后序遍历

void preorder(BSTNode \* r)const

{

if (r == nullptr)return ;

r->visit();

preorder(r->left());

preorder(r->right());

}

void inorder(BSTNode \* r)const

{

if (r == nullptr)return;

inorder(r->left());

r->visit();

inorder(r->right());

}

void postorder(BSTNode \* r)const

{

if (r == nullptr)return;

postorder(r->left());

postorder(r->right());

r->visit();

}

private:

Key k;

E it;

BSTNode \* lc;

BSTNode \* rc;

};

template<typename Key,typename E>

class BST //:public Dictionary<Key,E>

{

//public 部分 封装了insert remove find 遍历的函数

public:

BST()

{

root = nullptr;

}

~BST()

{

clearhelp(root);

}

BST(const Key & k,const E & e)

{

root = new BSTNode<Key, E>(k, e);

}

void insert(const Key & k, const E & e)

{

root = inserthelp(root, k, e);

nodecount++;

}

E remove(const Key & k)

{

E temp = findhelp(root, k);

if (temp != NULL) //???

{

root = removehelp(root, k);

nodecount--;

}

return temp;

}

E removeAny()

{

if (root != nullptr)

{

E temp = root->element();

root = removehelp(root, root->key());

nodecount--;

return temp;

}

else return nullptr;

}

E find(const Key & k)const

{

return findhelp(root, k);

}

int size()

{

return nodecount;

}

//void print()const

//{

// if(root == nullptr)cout<<"The BST tree is empty";

// else printhelp(root,0);

//}

void preorder()const

{

root->preorder(root);

}

void inorder()const

{

root->inorder(root);

}

void postprder()const

{

root->postorder(root);

}

private:

BSTNode<Key, E>\*root;

int nodecount;

void clearhelp(BSTNode<Key, E> \* r)//delete所有的节点

{

if (r == nullptr)return;

clearhelp(r->left());

clearhelp(r->right());

delete r;

};

BSTNode<Key, E>\* inserthelp(BSTNode<Key, E> \* root\_, const Key & k, const E & it)

{

if (root\_ == nullptr)//如果当前节点为空 new

{

BSTNode<Key, E> \* temp = new BSTNode<Key, E>(k, it, nullptr, nullptr);

return temp;

//return new BSTNode<Key, E>(k, it, nullptr, nullptr);

}

if (k < root\_->key())//根据关键值 选择进入左孩子 还是右孩子

{

root\_->setLeft(inserthelp(root\_->left(), k, it));

}

else root\_->setRight(inserthelp(root\_->right(), k, it));

return root\_;

// do not understand

}

BSTNode<Key, E> \* deletemin(BSTNode<Key, E> \* rt)

{

if (rt->left() == nullptr)return rt->right();

else

{

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

return rt;

}

};

BSTNode<Key, E> \* getmin(BSTNode<Key, E> \* rt)

{

if (rt->left() == nullptr)return rt;

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

};

BSTNode<Key, E> \* removehelp(BSTNode<Key, E> \* rt,const Key & k)

{

if (rt == nullptr) return nullptr;

else if (k < rt->key())rt->setLeft(removehelp(rt->left(), k));

else if (k > rt->key())rt->setRight(removehelp(rt->right(), k));

else

{

BSTNode<Key, E> \* temp = rt;

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

{

rt = rt->right();

delete temp;

}

else if (rt->right() == nullptr)

{

rt = rt->left();

delete temp;

}

else

{

BSTNode<Key, E> \* temp = getmin(rt->right());

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

rt->setKey(temp->key());

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

delete temp;

}

}

return rt;

};

E findhelp(BSTNode<Key, E>\* root\_,const Key & k)const

{

if (root\_ == nullptr)return NULL;

if (k < root\_->key())return findhelp(root\_->left(), k);

else if (k > root\_->key())return findhelp(root\_->right(), k);

else return root\_->element();

};

void printhelp(BSTNode<Key, E> \* rt)

{

cout << "preorder is \n";

if (rt == nullptr)return;

cout << rt->element() << " ";

printhelp(rt -> left());

printhelp(rt->right());

};

};

#include<iostream>

using namespace std;

int main()

{

int order, element, key;

BST<int, int> tree;

while (true)

{

cout << "enter order\n";

cout << "1.insert an element(enter key and element)\n";

cout << "2.see the tree\n";

cout << "3.find a certain element\n";

cout << "4.delete an element\n";

cin >> order;

switch (order)

{

case 1:

cout << "enter the key and element:";

cin >> key >> element;

tree.insert(key, element);

break;

case 2:

cout << "preorder:\n";

tree.preorder();

cout << "\ninorder:\n";

tree.inorder();

cout << "\npostorder:\n";

tree.postprder();

cout << endl;

break;

case 3:

cout << "please enter the key to find the element\n";

cin >> key;

tree.find(key);

break;

case 4:

cout << "enter the key of the element\n";

cin >> key;

tree.remove(key);

}

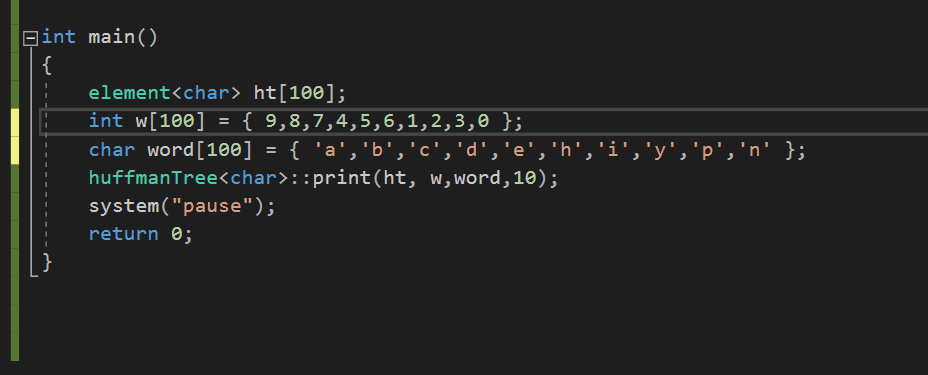
}

}

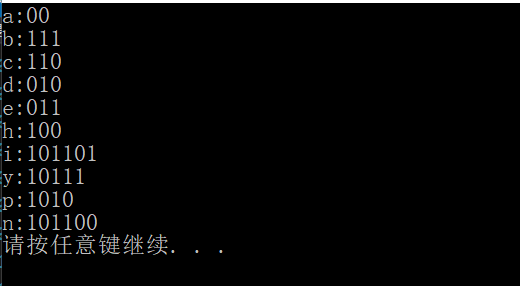
d.给出一组权值，将它们构建成Huffman树，并输出Huffman编码。

程序运行截图

样例:



输出:



程序片段

#pragma once

#include<iostream>

using namespace std;

template<typename T>

struct element

{

T it;

int w;

int lc, rc, parent;

//parent = -1 表示没有父亲节点

};

template<typename T>

class huffmanTree

{

public:

static void print(element<T> ht[], int w[],T e[], int n)//外部接口

{

buildTree(ht, w, e,n);

int c, p, i, start;

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

{

int code[100];

start = 0;

c = i;

p = ht[c].parent;

while (p>0)

{

if (ht[p].lc == c)

{

code[start++] = 0;

}

else

{

code[start++] = 1;

}

c = p;

p = ht[c].parent;

}

cout << ht[i].it << ":";

for (int j = start - 1; j >= 0; j--)

cout << code[j];

cout << endl;

}

}

private:

static void buildTree(element<T> huffmantree[], int w[], T e[] ,int n)

{

for (int i = 0; i < 2 \* n - 1; i++)

{

huffmantree[i].parent = -1;

huffmantree[i].lc = -1;

huffmantree[i].rc = -1;

}

for (int i = 0; i < n; i++)//构造

{

huffmantree[i].w = w[i];

huffmantree[i].it = e[i];

}

for (int k = n; k < 2 \* n - 1; k++)

{

int s1, s2;

//查找并合并当前最小的两个节点

selectMin(huffmantree, k, s1, s2);

huffmantree[s1].parent = k;

huffmantree[s2].parent = k;

huffmantree[k].lc = s1;

huffmantree[k].rc = s2;

huffmantree[k].w = huffmantree[s1].w + huffmantree[s2].w;

}

}

static void selectMin(element<T> a[], int n, int &s1, int &s2)

{

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

{

//初始化 将所有节点设置为没有parent

if (a[i].parent == -1)

{

s1 = i;

break;

}

}

for (int i = 0; i < n; i++)//s1为权值最小的下标

{

if (a[i].parent == -1 && a[s1].w > a[i].w)s1 = i;

}

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

{

if (a[j].parent == -1 && j != s1)//

{

s2 = j;

break;

}

}

for (int j = 0; j < n; j++)//s2为另一个权值最小的结点

{

if (a[j].parent == -1 && a[s2].w > a[j].w &&j != s1)s2 = j;

}

}

};

int main()

{

element<char> ht[100];

int w[100] = { 9,8,7,4,5,6,1,2,3,0 };

char word[100] = { 'a','b','c','d','e','h','i','y','p','n' };

huffmanTree<char>::print(ht, w,word,10);

system("pause");

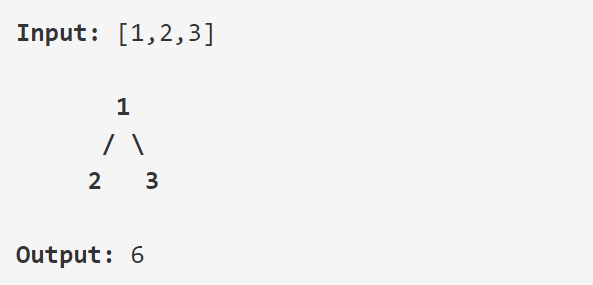
return 0;

}

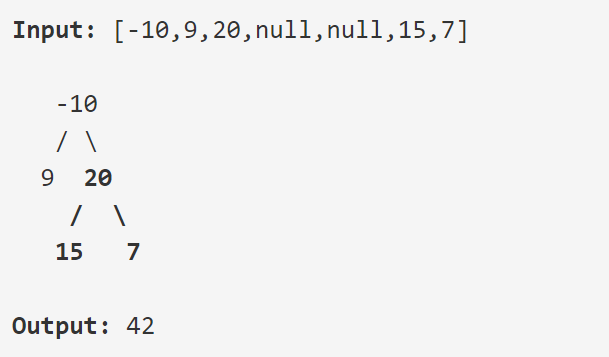
e.附加题：Given a **non-empty** binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain **at least one node** and does not need to go through the root.

**Example 1:**



**Example 2:**



（通过链接<https://leetcode.com/problems/binary-tree-maximum-path-sum/description/>测试你的代码）

附加题

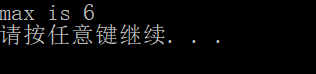
程序运行截图

e)

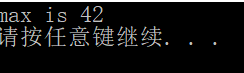
程序运行截图：

样例：

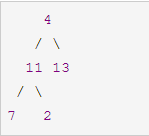
Example 1:



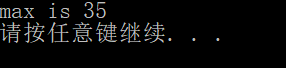
Example 2:



Example 3:

可以看出来 选择路径7->11->4->13可以得到结果的最大值 35

结果：



题解:首先递归到叶子节点，用深搜的思想做，递归函数返回的是 以当前节点为根节点的子树，到某个叶节点的最大值，答案作为参数res传进去。

完整程序片段:

#include<iostream>

using namespace std;

//思路 先递归到叶子节点 再回溯

class TreeNode

{

public:

TreeNode(int ww)

{

val = ww;

}

TreeNode \* left, \*right;

int val;

};

class Solution

{

public:

int maxPathSum(TreeNode \* root)

{

int res = INT\_MIN;

solve(root, res);

return res;

}

//要得到的答案是res

//因为如果

int solve(TreeNode\* node, int & res)// 返回的 是 以当前结点为根结点，到叶节点的最大路径之和(也就是说只能取到一边的值)

{

if (node == nullptr) return 0;

//max\_ 取的是 0 left\_ right\_三者之间的最大值

int left\_ = solve(node->left, res);

int right\_ = solve(node->right, res);

if (left\_ < 0)left\_ = 0;

if (right\_ < 0)right\_ = 0;

int max\_ = left\_ > right\_ ? left\_ : right\_;//get the max of left\_ right\_ 0

//max加上当前的节点的值

max\_ += node->val;

if (res < (left\_ + right\_ + node->val))res = left\_ + right\_ + node->val;

return max\_;

}

};

int main()

{

TreeNode \* root = new TreeNode(4);

TreeNode \* temp = new TreeNode(11);

root->left = temp;

temp = new TreeNode(7);

root->left->left = temp;

temp = new TreeNode(2);

root->left->right = temp;

temp = new TreeNode(13);

root->right = temp;

Solution s;

cout << "max is " << s.maxPathSum(root) << endl;

system("pause");

return 0;

}

Leetcode AC代码

C:\Users\lmx\AppData\Local\Temp\1540211339(1).png

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

int maxPathSum(TreeNode \* root)

{

int res = INT\_MIN;

solve(root, res);

return res;

}

int solve(TreeNode\* node, int & res)

{

if (node == nullptr) return 0;

int left\_ = solve(node->left, res);

int right\_ = solve(node->right, res);

if (left\_ < 0)left\_ = 0;

if (right\_ < 0)right\_ = 0;

int max\_ = left\_ > right\_ ? left\_ : right\_;//get the max of left\_ right\_ 0

max\_ += node->val;

if (res < (left\_ + right\_ + node->val))res = left\_ + right\_ + node->val;

return max\_;

}

};