1、给定二叉树，返回其节点值的后序遍历。

class Solution {

public:

vector<int> postorderTraversal(TreeNode \*root) {

if(root==NULL)

return res;

if(root->left!=NULL)

postorderTraversal(root->left);

if(root->right!=NULL)

postorderTraversal(root->right);

res.push\_back(root->val);

return res;

}

private:

vector<int>res;

};

2、给定二叉树，返回其节点值的前序遍历。

class Solution {

public:

vector<int> preorderTraversal(TreeNode \*root) {

if(root==NULL)

return res;

res.push\_back(root->val);

if(root->left!=NULL)

preorderTraversal(root->left);

if(root->right!=NULL)

preorderTraversal(root->right);

return res;

}

private:

vector<int>res;

};

3、给定包含从0到9的数字的二叉树，每个根到叶路径可以表示数字。一个例子是root-to-leaf path1-> 2-> 3，它代表数字123。找到所有根到叶数的总和。

1

/ \

2 3

root-to-leaf path1-> 2表示数字12。root-to-leaf path1-> 3表示数字13。返回总和= 12 + 13 = 25。

class Solution {

public:

int sumNumbers(TreeNode \*root) {

if(root==NULL) return 0;

int sum=0;

return preorderTraversal(root,sum);

}

int preorderTraversal(TreeNode \*root,int sum) {

if(root==NULL)

return 0;

sum=sum\*10+root->val;

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

return sum;

return preorderTraversal(root->left,sum)+preorderTraversal(root->right,sum);

}

};

4、给定二叉树，找到最大路径总和。路径可以在树中的任何节点处开始和结束。例如：给出下面的二叉树，

1

/ \

2 3

Return6。

class Solution {

public:

int maxPathSum(TreeNode \*root) {

if(root==NULL) return 0;

maxroot=-1e8;

maxsum(root);

return maxroot;

}

int maxsum(TreeNode \*root)

{

if(root==NULL) return 0;

int left=max(0,maxsum(root->left));

int right=max(0,maxsum(root->right));

maxroot=max(maxroot,left+right+root->val);

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

}

private:

int maxroot;

};

5、给定二叉树，确定它是否是高度平衡的。对于这个问题，高度平衡的二叉树被定义为二叉树，其中每个节点的两个子树的深度从不相差超过1。

class Solution {

public:

bool isBalanced(TreeNode \*root) {

if(root==NULL) return true;

if(abs(maxdepth(root->left)-maxdepth(root->right))>1)

return false;

else

return isBalanced(root->left)&&isBalanced(root->right);

}

int maxdepth(TreeNode \*root)

{

if(root==NULL) return 0;

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

}

};

6、给定二叉树，找到它的最小深度。最小深度是从根节点到最近叶节点的最短路径上的节点数。**修改min为max可求树的最大深度；**

class Solution {

public:

int run(TreeNode \*root) {

if(root==NULL)

return 0;

int left=run(root->left);

int right=run(root->right);

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

return 1+left+right;

return 1+min(left,right);

}

};

7、找出根节点到叶节点所有节点值的和等于给出的sum的所有路径

class Solution {

public:

vector<vector<int> > pathSum(TreeNode \*root, int sum) {

vector<vector<int>> res;

vector<int>temp;

pathsum(root,sum,temp,res);

return res;

}

void pathsum(TreeNode \*root, int sum,vector<int>temp,vector<vector<int> >& res)//temp不能引用，每次是复制一份，而res要引用数据存完后在上面修改。

{

if(root==NULL)

return;

temp.push\_back(root->val);

if(root->left==NULL&&root->right==NULL&&sum-root->val==0)

{

res.push\_back(temp);

}

pathsum(root->left,sum-root->val,temp,res);

pathsum(root->right,sum-root->val,temp,res);

}

};

8、是否存在根节点到叶节点所有节点值的和等于给出的sum的一条路径，若存在则返回true

**解法一：**

class Solution {

public:

bool hasPathSum(TreeNode \*root, int sum) {

vector<vector<int>> res;

vector<int>temp;

pathsum(root,sum,temp,res);

if(res.size()!=0)

return true;

else

return false;

}

void pathsum(TreeNode \*root, int sum,vector<int>temp,vector<vector<int> >& res)

{

if(root==NULL)

return;

temp.push\_back(root->val);

if(root->left==NULL&&root->right==NULL&&sum-root->val==0)

{

res.push\_back(temp);

}

pathsum(root->left,sum-root->val,temp,res);

pathsum(root->right,sum-root->val,temp,res);

}

};

解法二：

class Solution {

public:

bool hasPathSum(TreeNode \*root, int sum) {

if(root==NULL)

return false;

if(root->left==NULL&&root->right==NULL&&sum-root->val==0)

{

return true;

}

return hasPathSum(root->left,sum-root->val)||hasPathSum(root->right,sum-root->val);

}

};

9、跟进问题“ 在每个节点中填充下一个右指针 ”。如果给定的树可以是任何二叉树怎么办？您以前的解决方案是否仍然可行，注意：您可能只使用恒定的额外空间。

例如，

给定以下二叉树，

1

/ \

2 3

/ \ \

4 5 7

调用函数后，树应该如下所示：

1 - > NULL

/ \

2 - > 3 - > NULL

/ \ \

4 - > 5 - > 7 - > NULL

class Solution {

public:

void connect(TreeLinkNode \*root) {

if(root==NULL) return;

while(root!=NULL)

{

TreeLinkNode \*temp = new TreeLinkNode(0);

TreeLinkNode \*proot=temp;

while(root!=NULL)

{

if(root->left!=NULL)

{

proot->next=root->left;

proot=proot->next;

}

if(root->right!=NULL)

{

proot->next=root->right;

proot=proot->next;

}

root=root->next;

}

root=temp->next;

}

}

};

10、给定一个二叉树，返回其中序遍历的节点的值

1

\

2

/

3

return[1,3,2].

class Solution {

public:

vector<int> inorderTraversal(TreeNode \*root) {

if(root==NULL)

return res;

if(root->left!=NULL)

inorderTraversal(root->left);

res.push\_back(root->val);

if(root->right!=NULL)

inorderTraversal(root->right);

return res;

}

private:

vector<int>res;

};

11、单链表的归并排序；

class Solution{

public:

ListNode\* sortlist(ListNode \*head)

{  
 if(head==NULL || head->next== NULL)

Return head;

ListNode \* l1 = head, \* l2 = head;

While(l2-next && l2->next->next)

{

l1= l1->next;

l2= l2->next->next;

}

ListNode \* right=sortlist(l1->next);

`l1->next=NULL;

ListNode \* left=sortlist(head);

return merge(left,right);

}

Private:

ListNode\* merge(ListNode \*h1, ListNode \*h2)

{  
 if(h1 == NULL) return h2;

if(h2 == NULL) return h1;

ListNode \* res= new listNode(0);

ListNode \* tem=res;

while(h1 && h2)

{

if(h1->val < h2->val)

{

tem->next=h1;

h1=h1->next;

}

else

{

tem->next=h2;

h2=h2->next;

}

tem=tem->next;

}

if(h1)

tem->next=h1;

if(h2)

tem->next=h2;

return res->next;

}

12、一个数组中只有一个元素出现了一次，其余的各元素均出现了两次，请找出这个元素；使用异或运算要求重复元素出现的次数是偶数次，未重复元素时奇数次

class Solution {

public:

int singleNumber(int A[], int n) {

int res=0;

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

res^=A[i];

return res;

}

};

13、将一个升序的单链表转换为平衡二叉搜索树；

class Solution {

public:

TreeNode \*sortedListToBST(ListNode \*head) {

if(head==NULL)

return NULL;

if(head->next==NULL)

return new TreeNode(head->val);

ListNode \*mid=head;

ListNode \*end=head;

ListNode \*premid=NULL;

while(end!=NULL&&end->next!=NULL)

{

premid=mid;

mid=mid->next;

end=end->next->next;

}

premid->next=NULL;

TreeNode \*root= new TreeNode(mid->val);

root->left=sortedListToBST(head);

root->right=sortedListToBST(mid->next);

return root;

}

};

14、将一个升序的数组转换成一个二叉搜索树

class Solution {

public:

TreeNode \*sortedArrayToBST(vector<int> &num) {

if(num.size()==0)

return NULL;

return sortedArrayToBST(num,0,num.size()-1);

}

private:TreeNode \*sortedArrayToBST(vector<int> &num,int begin,int end)

{

if(begin>end)

return NULL;

if(begin==end)

return new TreeNode(num[begin]);

int mid=(begin+end+1)/2;

TreeNode \*root= new TreeNode(num[mid]);

root->left=sortedArrayToBST(num,begin,mid-1);

root->right=sortedArrayToBST(num,mid+1,end);

return root;

}

};

15、给定二叉树，返回其节点值的自下而上级别顺序遍历。 （即，从左到右，逐层逐层）

3

/ \

9 20

/ \

15 7

return its bottom-up level order traversal as:

[

[15,7]

[9,20],

[3],

]

class Solution {

public:

vector<vector<int> > levelOrderBottom(TreeNode \*root) {

vector<vector<int>>res;

if(root==NULL)

return res;

queue<TreeNode\*>que;

que.push(root);

while(que.size())

{

vector<int>res1;

int n=que.size();

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

{

TreeNode\* temp=que.front();

que.pop();

res1.push\_back(temp->val);

if(temp->left) que.push(temp->left);

if(temp->right) que.push(temp->right);

}

res.push\_back(res1);

}

reverse(res.begin(),res.end());//**若从上向下保存的话，就不用reverse**

return res;

}

};

16、给定二叉树，返回其节点值的Z字形级别遍历。 （即，从左到右，然后从右到左进行下一级别并在之间交替）

3

/ \

9 20

/ \

15 7

return its zigzag level order traversal as:

[

[3],

[20,9],

[15,7]

]

class Solution {

public:

vector<vector<int> > zigzagLevelOrder(TreeNode \*root) {

vector<vector<int>>res;

if(root==NULL)

return res;

int m=0;

queue<TreeNode\*>que;

que.push(root);

while(que.size())

{

vector<int>res1;

m++;

int n=que.size();

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

{

TreeNode\* temp=que.front();

que.pop();

res1.push\_back(temp->val);

if(temp->left) que.push(temp->left);

if(temp->right) que.push(temp->right);

}

if(m%2==0)

reverse(res1.begin(),res1.end());//**每隔一次翻转一次res1里面的数，达到Z字形；**

res.push\_back(res1);

}

return res;

}

};

17、Given n, how many structurally unique BST's (binary search trees) that store values 1...n? For example,  
Given *n* = 3, there are a total of 5 unique BST's。总共有左右子树数量乘积个；

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

class Solution {

public:

int numTrees(int n) {

if(n<=1)

return 1;

int count=0;

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

count+=numTrees(i-1)\*numTrees(n-i);

return count;

}

};

18、Given n, generate all structurally unique BST's (binary search trees) that store values 1...n.

For example,Given n = 3, your program should return all 5 unique BST's shown below.

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

class Solution {

public:

//递归创建

vector<TreeNode \*> creatTrees(int left,int right){

vector<TreeNode \*> res;

if(left>right){

res.push\_back(NULL);

return res;

}

for(int i=left;i<=right;i++){ //以i为根节点的树，其左子树由[1, i-1]构成， 其右子树由[i+1, n]构成。该原则建树具有唯一性

vector<TreeNode \*> left\_res = creatTrees(left,i-1);

vector<TreeNode \*> right\_res = creatTrees(i+1,right);

//每个左边的子树跟所有右边的子树匹配，而每个右边的子树也要跟所有的左边子树匹配，总共有左右子树数量的乘积种情况

int lsize = left\_res.size();

int rsize = right\_res.size();

for(int j=0;j<lsize;j++){

for(int k=0;k<rsize;k++){

TreeNode \*root = new TreeNode(i);

root->left = left\_res[j];

root->right = right\_res[k];

res.push\_back(root);

}

}

}

return res;

}

vector<TreeNode \*> generateTrees(int n) {

//本题主要考察二叉排序树的构建

//unique-binary-search-trees-i，我们可以知道结果的数目是一个卡特兰数列，当本题需要把所有可能的二叉排序

//给构建出来要求解所有的树，自然是不能多项式时间内完成的思路是每次一次选取一个结点为根，然后递归求解左右子树的所有结果，最后根据左右子树的返回的所有子树，依次选取

//然后接上（每个左边的子树跟所有右边的子树匹配，而每个右边的子树也要跟所有的左边子树匹配，总共有左右子树数量的乘积种情况），构造好之后作为当前树的

//结果返回

return creatTrees(1,n); //从1作为root开始，到n作为root结束

}

};

19、实现两个函数，分别用来序列化和反序列化二叉树

/\*

struct TreeNode {

int val;

struct TreeNode \*left;

struct TreeNode \*right;

TreeNode(int x) :

val(x), left(NULL), right(NULL) {}

};

\*/

class Solution {

public:

char\* Serialize(TreeNode \*root) {

if(!root) return "#";

string r = to\_string(root->val);

r+=',';

// r.push\_back(',');

char \*left = Serialize(root->left);

char \*right = Serialize(root->right);

char \*ret = new char[strlen(left) + strlen(right) + r.size()];

strcpy(ret, r.c\_str());

strcat(ret, left);

strcat(ret, right);

return ret;

}

TreeNode\* Deserialize(char \*str) {

return decode(str);

}

private:

TreeNode\* decode(char \*&str) {

if(\*str=='#'){

str++;

return NULL;

}

int num = 0;

while(\*str != ',')

num = num\*10 + (\*(str++)-'0');

str++;

TreeNode \*root = new TreeNode(num);

root->left = decode(str);

root->right = decode(str);

return root;

}

};

20、给定一棵二叉搜索树，请找出其中的第k小的结点。例如， （5，3，7，2，4，6，8） 中，按结点数值大小顺序第三小结点的值为4。

class Solution {

public:

void inorder(TreeNode\* pRoot,vector<TreeNode\*>&vec)

{

if(pRoot==NULL)return;

inorder(pRoot->left,vec);

vec.push\_back(pRoot);

inorder(pRoot->right,vec);

}

TreeNode\* KthNode(TreeNode\* pRoot, int k)

{

if(pRoot==NULL||k<=0)

return NULL;

vector<TreeNode\*>vec;

inorder(pRoot,vec);

if(k>vec.size())

return NULL;

return vec[k-1];

}

};

21、请实现一个函数，用来判断一颗二叉树是不是对称的。注意，如果一个二叉树同此二叉树的镜像是同样的，定义其为对称的。

class Solution {

public:

bool helper(TreeNode\* left,TreeNode\* right)

{

if(left==NULL&&right==NULL)

return true;

if(left!=NULL&&right!=NULL)

return left->val==right->val&&helper(left->left,right->right)&&

helper(left->right,right->left);

return false;

}

bool isSymmetrical(TreeNode\* pRoot)

{

if(pRoot==NULL)

return true;

return helper(pRoot->left,pRoot->right);

}

};

22、给定一个二叉树和其中的一个结点，请找出中序遍历顺序的下一个结点并且返回。注意，树中的结点不仅包含左右子结点，同时包含指向父结点的指针。

class Solution {

public:

TreeLinkNode\* GetNext(TreeLinkNode\* pNode)

{

if(pNode==NULL)

return NULL;

if(pNode->right!=NULL)

{

pNode=pNode->right;

while(pNode->left!=NULL)

pNode=pNode->left;

return pNode;

}

while(pNode->next!=NULL)

{

TreeLinkNode\* ptemp;

ptemp=pNode->next;

if(ptemp->left==pNode)

return ptemp;

pNode=pNode->next;

}

return NULL;

}

};

23、操作给定的二叉树，将其变换为源二叉树的镜像。输入描述:二叉树的镜像定义：源二叉树

8 源二叉树

/ \

6 10

/ \ / \

5 7 9 11

镜像二叉树

8

/ \

10 6

/ \ / \

11 9 7 5

class Solution {

public:

void Mirror(TreeNode \*pRoot) {

TreeNode \*temp=NULL;

if(pRoot!=NULL)

{

temp=pRoot->left;

pRoot->left=pRoot->right;

pRoot->right=temp;

if(pRoot->left!=NULL)

Mirror(pRoot->left);

if(pRoot->right!=NULL)

Mirror(pRoot->right);

}

}

};

24、输入两棵二叉树A，B，判断B是不是A的子结构。（ps：我们约定空树不是任意一个树的子结构）

class Solution {

bool issubtree(TreeNode\* pRoot1, TreeNode\* pRoot2)

{

if(pRoot2==NULL)

return true;

if(pRoot1==NULL)

return false;

if(pRoot1->val==pRoot2->val)

{

return issubtree(pRoot1->left,pRoot2->left)&&issubtree(pRoot1->right,pRoot2->right);

}

else return false;

}

public:

bool HasSubtree(TreeNode\* pRoot1, TreeNode\* pRoot2)

{

if(pRoot1==NULL||pRoot2==NULL)

return false;

return issubtree(pRoot1,pRoot2)||HasSubtree(pRoot1->left,pRoot2)||HasSubtree(pRoot1->right,pRoot2);

}

};

25、输入某二叉树的前序遍历和中序遍历的结果，请重建出该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。例如输入前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}，则重建二叉树并返回。

class Solution {

public:

TreeNode\* reConstructBinaryTree(vector<int> pre,vector<int> vin) {

int vinlength=vin.size();

if(vinlength==0)

return NULL;

TreeNode\* root=new TreeNode(pre[0]);

vector<int>pre\_left,pre\_right,vin\_left,vin\_right;

int Head=0;

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

{

if(vin[i]==pre[0])

{

Head=i;

break;

}

}

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

{

vin\_left.push\_back(vin[i]);

pre\_left.push\_back(pre[i+1]);

}

for(int i=Head+1;i<vinlength;i++)

{

vin\_right.push\_back(vin[i]);

pre\_right.push\_back(pre[i]);

}

root->left=reConstructBinaryTree(pre\_left,vin\_left);

root->right=reConstructBinaryTree(pre\_right,vin\_right);

return root;

}

};

26、