Lowest Common Ancestor of a Binary Search Tree

思路分析：该树是一颗二叉查找树，即左子树比根小，右子树比根大。所以，若给定的p、q节点值都比根节点的值小，则一定在其左子树上；反之，若都大，则在右子树上；若一大一小或有一个与根相等，则返回根节点。代码如下：

class Solution {

public:

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

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

int a,b,r;

r=root->val;

a=p->val;

b=q->val;

if(a==r||b==r||(a>r&&b<r)||(a<r&&b>r))

{

return root;

}

if(a<r&&b<r)

{

return lowestCommonAncestor(root->left,p,q);

}

if(a>r&&b>r)

{

return lowestCommonAncestor(root->right,p,q);

}

**return NULL; //防止没有考虑到的情况**

}

};

依旧是递归的思想。

第二次没有注意到是二叉查找树，采用另一种方法进行实现：

/\*\*

\* 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:

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

bool flag1,flag2,flag3,flag4;

int l1,l2,l3,l4;

flag1=isSameSide(root->left,p);

flag2=isSameSide(root->left,q);

if(flag1&flag2==1){

l1=MaxTreeDeep(p);

l2=MaxTreeDeep(q);

if(l1>l2){

return p;

}

else{

return q;

}

}

flag3=isSameSide(root->right,p);

flag4=isSameSide(root->right,q);

if(flag3&flag4==1){

l3=MaxTreeDeep(p);

l4=MaxTreeDeep(q);

if(l3>l4){

return p;

}

else{

return q;

}

}

else{

return root;

}

}

int MaxTreeDeep(TreeNode\* root){ //计算深度

if(root==NULL){

return 0;

}

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

}

bool isSameSide(TreeNode\* root, TreeNode\* p){ //判断是否在树的一边

if(root==NULL){

return false;

}

if(root==p){

return true;

}

isSameSide(root->left,p);

isSameSide(root->right,p);

}

};

尽管没有通过，但这种方法可以考虑实现

在第一种方法上尝试，发现wrong answear。这里我想当然的以为在同一侧只需考虑谁的深度大就是谁，测试案例为[5,3,6,2,4,null,null,1] node with value 1 node with value 4

时就出错了！！！

/\*\*

\* 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:

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

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

return root;

}

int r,a,b;

int lp,lq;

int temp;

r=root->val;

a=p->val;

b=q->val;

if(a<r&&b>r||a>r&&b<r){

return root;

}

if(a<r&&b<r){

lp=MaxTreeDeep(p);

lq=MaxTreeDeep(q);

if(lp>lq){

temp=0;

}

else{

temp=1;

}

}

else if(a>r&&b>r){

lp=MaxTreeDeep(p);

lq=MaxTreeDeep(q);

if(lp>lq){

temp=0;

}

else{

temp=1;

}

}

if(temp==0){

return p;

}

else{

return q;

}

}

int MaxTreeDeep(TreeNode\* root){

if(root==NULL){

return 0;

}

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

}

};