124. Binary Tree Maximum Path Sum

Given a binary tree, find the maximum path sum. The path may start and end at any node in the tree.

For each node there can be four ways that the max path goes through the node:  
1. Node only  
2. Max path through Left Child + Node  
3. Max path through Right Child + Node  
4. Max path through Left Child + Node + Max path through Right Child

public class Solution {

int maxValue;

public int maxPathSum(TreeNode root) {

maxValue = Integer.MIN\_VALUE;

maxPathDown(root);

return maxValue;

}

private int maxPathDown(TreeNode node) {

if (node == null) return 0;

int left = Math.max(0, maxPathDown(node.left));

int right = Math.max(0, maxPathDown(node.right));

maxValue = Math.max(maxValue, left + right + node.val);

return Math.max(left, right) + node.val;

}

}

95. Unique Binary Search Trees II

public class Solution {

public List<TreeNode> generateTrees(int n) {

return genTrees(1,n);

}

public List<TreeNode> genTrees (int start, int end)

{

List<TreeNode> list = new ArrayList<TreeNode>();

if(start>end)

{

list.add(null);

return list;

}

if(start == end){

list.add(new TreeNode(start));

return list;

}

List<TreeNode> left,right;

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

{

left = genTrees(start, i-1);

right = genTrees(i+1,end);

for(TreeNode lnode: left)

{

for(TreeNode rnode: right)

{

TreeNode root = new TreeNode(i);

root.left = lnode;

root.right = rnode;

list.add(root);

}

}

}

return list;

}

}

98. Validate Binary Search Tree

迭代函數里定好下個節點的val範圍，如果超出範圍就false

public class Solution {

public boolean isValidBST(TreeNode root) {

return isValidBST(root, Long.MIN\_VALUE, Long.MAX\_VALUE);

}

public boolean isValidBST(TreeNode root, long minVal, long maxVal) {

if (root == null) return true;

if (root.val >= maxVal || root.val <= minVal) return false;

return isValidBST(root.left, minVal, root.val) && isValidBST(root.right, root.val, maxVal);

}

}

236. Lowest Common Ancestor of a Binary Tree

迭代，如果當前節點是P或者q 返回當前節點。如果是中間某個節點，見到左右都不為null，這個節點就是lowest common ancestor。如果這個節點發現左右一邊null,一邊not null 返回not null的點

public class Solution {

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if(root == null || root == p || root == q) return root;

TreeNode left = lowestCommonAncestor(root.left, p, q);

TreeNode right = lowestCommonAncestor(root.right, p, q);

if(left != null && right != null) return root;

return left != null ? left : right;

}

}

114. Flatten Binary Tree to Linked List

遇到root左右都有子樹時，在root.left中找到最大的節點K，把root.right接到K的右邊。然後把root.left放到root.right, root.left=null

**class** **Solution** {

**public**:

void flatten(TreeNode \*root) {

TreeNode\*now = root;

**while** (now)

{

**if**(now->left)

{

*//Find current node's prenode that links to current node's right subtree*

TreeNode\* pre = now->left;

**while**(pre->right)

{

pre = pre->right;

}

pre->right = now->right;

*//Use current node's left subtree to replace its right subtree(original right*

*//subtree is already linked by current node's prenode*

now->right = now->left;

now->left = **NULL**;

}

now = now->right;

}

}

};

106. Construct Binary Tree from Inorder and Postorder Traversal

public class Solution {

public TreeNode buildTree(int[] preorder, int[] inorder) {

if (preorder == null || inorder == null || preorder.length == 0 || inorder.length == 0)

return null;

int len = preorder.length;

return buildTree(preorder, 0, len - 1, inorder, 0, len - 1);

}

private TreeNode buildTree(int[] preorder, int preStart, int preEnd, int[] inorder, int inStart, int inEnd) {

if (inStart > inEnd) {

return null;

}

int rootVal = preorder[preStart];

int rootIdx = 0;

for (int i = inStart; i <= inEnd; i++) {

if (inorder[i] == rootVal) {

rootIdx = i;

break;

}

}

int len = rootIdx - inStart;

TreeNode root = new TreeNode(rootVal);

// recursively call its left and right subtree

root.left = buildTree(preorder, preStart + 1, preStart + len, inorder, inStart, rootIdx - 1);

root.right = buildTree(preorder, preStart + len + 1, preEnd, inorder, rootIdx + 1, inEnd);

return root;

}

}

Find the inorder successor in BST

Treenode沒有parent指針

思路：當前node如果有右子樹，就在右子樹里找最左邊最底的node，就是successor

如果沒有右子樹。需要從root開始找，因為是BST，從root往下找，一直找到node。在尋找的路徑中，在比node大的當中找到最小的即可。如果一個都找不到，說明沒有successor

129. Sum Root to Leaf Numbers

**public** **class** Solution {

**public** **int** sumNumbers(TreeNode root) {

**return** dfs(root, 0);

}

**private** **int** dfs(TreeNode root, **int** prev){

**if**(root == **null**) {

**return** 0;

}

**int** sum = root.val + prev \* 10;

**if**(root.left == **null** && root.right == **null**) {

**return** sum;

}

**return** dfs(root.left, sum) + dfs(root.right, sum);

}

}