Lab 7

Binary tree

1)144

class Solution {

    public List<Integer> preorderTraversal(TreeNode root) {

         List<Integer> ans = new ArrayList<>();

        helper(root, ans);

        return ans;

    }

    private void helper(TreeNode node, List<Integer> ans) {

        if(node == null) return;

        ans.add(node.val);

        helper(node.left, ans);

        helper(node.right, ans);

    }

}

2)94

class Solution {

    public List<Integer> inorderTraversal(TreeNode root) {

      List<Integer> result=new ArrayList<>();

        tree(root,result);

        return(result);

    }

    public void tree(TreeNode root ,List<Integer> result ){

        if(root==null){

            return;

        }

        tree(root.left,result);

        result.add(root.val);

        tree(root.right,result);

    }

}

3)145

class Solution {

    ArrayList<Integer> list;

    public List<Integer> postorderTraversal(TreeNode root) {

        list = new ArrayList<>();

        postorder(root);

        return list;

    }

    public void postorder(TreeNode root){

        if(root==null) return;

        postorder(root.left);

        postorder(root.right);

        list.add(root.val);

    }

}

4)102

class Solution {

    public List<List<Integer>> levelOrder(TreeNode root) {

       Queue<TreeNode> queue = new LinkedList<>();

        List<List<Integer>> wrapList = new LinkedList<List<Integer>>();

        if(root==null)

        return wrapList;

        queue.offer(root);

        while(!queue.isEmpty()){

            int size = queue.size();

            List<Integer> subList = new LinkedList<>();

            for(int i=0;i<size;i++){

                if(queue.peek().left!=null)  queue.offer(queue.peek().left);

                if(queue.peek().right!=null) queue.offer(queue.peek().right);

                subList.add(queue.poll().val);

            }

            wrapList.add(subList);

        }

        return wrapList;

    }

}

5)102

class Solution {

    public int maxDepth(TreeNode root) {

       if(root == null){

            return 0;

        }

        int lh = maxDepth(root.left);

        int rh = maxDepth(root.right);

        return  Math.max(lh,rh)+1;

    }

}

6)110

class Solution {

    public boolean isBalanced(TreeNode root) {

      return height(root)!=-1;

    }

    public int height(TreeNode node){

        if(node==null){

            return 0;

        }

        int leftHeight=height(node.left);

        if(leftHeight==-1) return -1;

        int rightHeight=height(node.right);

        if(rightHeight==-1) return -1;

        if(Math.abs(leftHeight-rightHeight)>1) return -1;

        return 1+Math.max(leftHeight,rightHeight);

    }

}

7)543

class Solution {

    public int diameterOfBinaryTree(TreeNode root) {

       int[] ans = new int[1];

        height(root, ans);

        return ans[0];

    }

    private int height(TreeNode node, int[] ans) {

        if (node == null) return 0;

        int lh = height(node.left, ans);

        int rh = height(node.right, ans);

        ans[0] = Math.max(ans[0], lh + rh);

        return 1 + Math.max(lh, rh);

    }

}

8)124

class Solution {

    public int maxPathSum(TreeNode root) {

     int []maxValue = new int[1];

        maxValue[0] = Integer.MIN\_VALUE;

        maxPathDown(root, maxValue);

        return maxValue[0];

    }

    public static int maxPathDown(TreeNode root, int []maxValue){

        if(root == null)

         return 0;

        int lh = Math.max(0, maxPathDown(root.left, maxValue));

        int rh = Math.max(0, maxPathDown(root.right, maxValue));

        maxValue[0] = Math.max(maxValue[0], root.val + lh + rh);

        return root.val + Math.max(lh, rh);

    }

}

9)100 class Solution {

    public boolean isSameTree(TreeNode p, TreeNode q) {

     if(p==null && q==null)

        {

        return true;

        }

        if(p==null || q==null)

        {

        return false;

        }

        if(p.val!=q.val)

        {

        return false;

        }

        if(isSameTree(p.left,q.left)&& isSameTree(p.right,q.right))

        {

        return true;

        }

        return false;

    }

}

10)103

public class Solution {

    public List<List<Integer>> zigzagLevelOrder(TreeNode root)

    {

        List<List<Integer>> sol = new ArrayList<>();

        travel(root, sol, 0);

        return sol;

    }

    private void travel(TreeNode curr, List<List<Integer>> sol, int level)

    {

        if(curr == null) return;

        if(sol.size() <= level)

        {

            List<Integer> newLevel = new LinkedList<>();

            sol.add(newLevel);

        }

        List<Integer> collection  = sol.get(level);

        if(level % 2 == 0) collection.add(curr.val);

        else collection.add(0, curr.val);

        travel(curr.left, sol, level + 1);

        travel(curr.right, sol, level + 1);

    }

}

11)987

class Solution {

    Map< Integer,Map<Integer,List<Integer>> > map;

    public List<List<Integer>> verticalTraversal(TreeNode root) {

        map=new TreeMap<>();

        preorder(root,0,0);

        List<List<Integer>> ans=new ArrayList<>();

        for(Integer k:map.keySet()){

            List<Integer> temp=new ArrayList<>();

            for(Integer lk:map.get(k).keySet()){

                Collections.sort(map.get(k).get(lk));

                temp.addAll(map.get(k).get(lk));

            }

            ans.add(temp);

        }

        return ans;

    }

    void preorder(TreeNode root,int r,int c){

        if(root==null)  return;

        map.computeIfAbsent(c,k->new TreeMap<>()).

            computeIfAbsent(r,k->new ArrayList<>()).

            add(root.val);

        preorder(root.left,r+1,c-1);

        preorder(root.right,r+1,c+1);

    }

}

12)199

class Solution {

    int maxLevel = 0;

    List<Integer> list = new ArrayList();

    public List<Integer> rightSideView(TreeNode root) {

        if(root == null) return list;

        rightView(root,1);

        return list;

    }

    void rightView(TreeNode root,int level){

        if(root == null) return;

        if(maxLevel < level){

            list.add(root.val);

            maxLevel = level;

        }

        rightView(root.right,level+1);

        rightView(root.left,level+1);

    }

}

13)101

class Solution {

    public boolean find(TreeNode root1,TreeNode root2){

        if(root1==null && root2==null) return true;

        if(root2==null || root1==null) return false;

        if(root1.val != root2.val) return false;

        return find(root1.left,root2.right)&find(root1.right,root2.left);

    }

    public boolean isSymmetric(TreeNode root) {

        return find(root.left,root.right);

    }

}

14)105

class Solution {

    private int i = 0;

    private int p = 0;

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

        return build(preorder, inorder, Integer.MIN\_VALUE);

    }

    private TreeNode build(int[] preorder, int[] inorder, int stop) {

        if (p >= preorder.length) {

            return null;

        }

        if (inorder[i] == stop) {

            ++i;

            return null;

        }

        TreeNode node = new TreeNode(preorder[p++]);

        node.left = build(preorder, inorder, node.val);

        node.right = build(preorder, inorder, stop);

        return node;

    }

}