1.144. 二叉树的前序遍历

递归

```java
public List<Integer> preorderTraversal(TreeNode root) {
    List<Integer> list = new ArrayList<>();
    preOrder(root,list);
    return list;
}
public void preOrder(TreeNode root,List<Integer> list){
    if(root==null) return;
    list.add(root.val);
    preOrder(root.left,list);
    preOrder(root.right,list);
}
```

普通迭代

```java
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) return result;
        Deque<TreeNode> stack = new LinkedList<>();
        stack.push(root);
        while (!stack.isEmpty()) {
            TreeNode node = stack.pop();
            result.add(node.val);
            if (node.right != null) stack.push(node.right);
            if (node.left != null) stack.push(node.left);
        }
        return result;
    }
```

船长亲传法

```java
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) return result;
        Deque<TreeNode> stack = new LinkedList<>();
        Deque<Integer> statusStack = new LinkedList<>();
        stack.push(root);
        statusStack.push(2);
        while (!stack.isEmpty()) {
            switch (statusStack.pop()) {
                case 0: {
                    statusStack.push(1);
                    if (stack.peek().right != null) {
                        stack.push(stack.peek().right);
                        statusStack.push(2);
                    }
                    break;
                }
                case 1: {
                    stack.pop();
                    break;
```

```
                }
                case 2: {
                    result.add(stack.peek().val);
                    statusStack.push(0);
                    if (stack.peek().left != null) {
                        stack.push(stack.peek().left);
                        statusStack.push(2);
                    }
                    break;
                }
            }
        }
        return result;
    }
```

2.589. N 叉树的前序遍历

递归

```
    public List<Integer> preorder(Node root) {
        List<Integer> result = new ArrayList<>();
        myPreOrder(root, result);
        return result;
    }

    public void myPreOrder(Node root, List<Integer> result) {
        if (root == null) return;
        result.add(root.val);
        for (Node children : root.children) {
            myPreOrder(children, result);
        }
    }
```

迭代

```
public List<Integer> preorder(Node root) {
    Deque<Node> stack = new LinkedList<>();
    List<Integer> result = new LinkedList<>();
    if (root == null) return result;
    stack.push(root);
    while (!stack.isEmpty()) {
        Node node = stack.pop();
        result.add(node.val);
        for (int i = node.children.size() - 1; node.children != null && i >= 0;
i--) {
            stack.push(node.children.get(i));
        }
    }
    return result;
}
```

3.226. 翻转二叉树

```
    public TreeNode invertTree(TreeNode root) {
        if (root == null) return null;
        TreeNode temp = root.right;
        root.right = root.left;
        root.left = temp;
        invertTree(root.left);
        invertTree(root.right);
        return root;
    }
```

4.剑指 Offer 32 - II. 从上到下打印二叉树 II

```
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        getResult(root, 0, result);
        return result;
    }

    public void getResult(TreeNode root, int k, List<List<Integer>> result) {
        if (root == null) return;
        if (k == result.size()) result.add(new ArrayList<Integer>());
        result.get(k).add(root.val);
        getResult(root.left, k + 1, result);
        getResult(root.right, k + 1, result);
    }
```

队列

```
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> result = new ArrayList<>();
    Queue<TreeNode> queue = new LinkedList<>();
    queue.offer(root);
    myLeveLOrder(root, result, queue, 0);
    return result;
}

public void myLeveLOrder(TreeNode root, List<List<Integer>> result,
Queue<TreeNode> queue, int k) {
    if (root == null) return;
    if (k == result.size()) result.add(new ArrayList<Integer>());
    while (!queue.isEmpty()) {
        result.get(k).add(queue.poll().val);
    }
    if (root.left != null) queue.offer(root.left);
    if (root.right != null) queue.offer(root.right);
    myLeveLOrder(root.left, result, queue, k + 1);
    myLeveLOrder(root.right, result, queue, k + 1);
}
```

5.107. 二叉树的层序遍历 II

```java
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        getResult(root, 0, result);
        Collections.reverse(result);
        return result;
    }

    public void getResult(TreeNode root, int k, List<List<Integer>> result) {
        if (root == null) return;
        if (k == result.size()) result.add(new ArrayList<Integer>());
        result.get(k).add(root.val);
        getResult(root.left, k + 1, result);
        getResult(root.right, k + 1, result);
    }

```

6.103. 二叉树的锯齿形层序遍历

```java
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        getResult(root, 0, result);
        for (int i = 1; i < result.size(); i += 2) {
            Collections.reverse(result.get(i));
        }
        return result;

    }

    public void getResult(TreeNode root, int k, List<List<Integer>> result) {
        if (root == null) return;
        if (k == result.size()) result.add(new ArrayList<Integer>());
        result.get(k).add(root.val);
        getResult(root.left, k + 1, result);
        getResult(root.right, k + 1, result);
    }
```

7.110. 平衡二叉树

```java
    public boolean isBalanced(TreeNode root) {
        return getHeight(root) >= 0;
    }

    public int getHeight(TreeNode root) {
        if (root == null) return 0;
        int leftHeight = getHeight(root.left);
        int rightHeight = getHeight(root.right);
        if (leftHeight < 0 || rightHeight < 0) return -2;
        if (Math.abs(leftHeight - rightHeight) > 1) return -2;
        return Math.max(leftHeight, rightHeight) + 1;
    }
```

8.112. 路径总和

```java
    public boolean hasPathSum(TreeNode root, int targetSum) {
        if (root == null) return false;
        if (root.left == null && root.right == null) return targetSum == root.val;
        return hasPathSum(root.left, targetSum - root.val) || hasPathSum(root.right, targetSum - root.val);
    }
```

9.105. 从前序与中序遍历序列构造二叉树

```java
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        if (preorder.length == 0) return null;
        int rootIndex = 0;
        while (preorder[0] != inorder[rootIndex]) ++rootIndex;
        int[] leftPre = new int[rootIndex],
                leftIn = new int[rootIndex],
                rightPre = new int[inorder.length - rootIndex - 1],
                rightIn = new int[inorder.length - rootIndex - 1];
        for (int i = 0; i < rootIndex; i++) {
            leftIn[i] = inorder[i];
            leftPre[i] = preorder[i+1];
        }

        for (int i = rootIndex + 1, j = 0; i < preorder.length; i++, j++) {
            rightIn[j] = inorder[i];
            rightPre[j] = preorder[i];
        }
        TreeNode root = new TreeNode(preorder[0]);
        root.left = buildTree(leftPre, leftIn);
        root.right = buildTree(rightPre, rightIn);
        return root;
    }
```

10.222. 完全二叉树的节点个数

```java
    public int countNodes(TreeNode root) {
        if (root==null) return 0;
        return countNodes(root.left)+countNodes(root.right)+1;
    }
```

11.剑指 Offer 54. 二叉搜索树的第k大节点

```
    public int getCount(TreeNode root) {
        if (root == null) return 0;
        return getCount(root.left) + getCount(root.right) + 1;
    }

    public int kthLargest(TreeNode root, int k) {
        int n = getCount(root.right);
        if (n >= k) return kthLargest(root.right, k);
        if (n + 1 == k) return root.val;
        return kthLargest(root.left, k - n - 1);
    }
```

从右遍历的思想

```
int res, k;

public int kthLargest(TreeNode root, int k) {
    this.k = k;
    searchK(root);
    return res;
}

public void searchK(TreeNode root) {
    if (root == null) return;
    searchK(root.right);
    if (k == 0) return;
    if (--k == 0) res = root.val;
    searchK(root.left);
}
```

12.剑指 Offer 26. 树的子结构

```
    public boolean isSubStructure(TreeNode A, TreeNode B) {
        if (A == null || B == null) return false;
        if (A.val == B.val && isMatch(A, B)) return true;
        return isSubStructure(A.left, B) || isSubStructure(A.right, B);
    }

    public boolean isMatch(TreeNode A, TreeNode B) {
        if (B == null) return true;
        if (A == null) return false;
        return A.val == B.val && isMatch(A.left, B.left) && isMatch(A.right,
B.right);
    }
```

13.662. 二叉树最大宽度

```
    public int widthOfBinaryTree(TreeNode root) {
        Queue<Pair> queue = new LinkedList<>();
        queue.offer(new Pair(root, 0));
        int result = 0;
        while (!queue.isEmpty()) {
            int count = queue.size();
            int left = queue.peek().number, right = 0;
            for (int i = 0; i < count; i++) {
```

```
                Pair temp = queue.poll();
                right = temp.number - left;
                if (temp.treeNode.left != null) {
                    queue.offer(new Pair(temp.treeNode.left, right * 2));
                }
                if (temp.treeNode.right != null) {
                    queue.offer(new Pair(temp.treeNode.right, right * 2 + 1));
                }

            }
            result = Math.max(result, right + 1);
        }
        return result;
    }
```

彩蛋

第五周作业彩蛋（Java学科）

题目描述：将数字1-10000（包含1和10000），奇数入栈，偶数入队列。然后将栈和队列中元素分别出栈和出队，将依次出栈和出队的数字分别两两相乘，然后将乘积累加求和，最后将和除以18（取整），即为答案。