144. 二叉树的前序遍历

```
//解法 1 递归
var result []int
func preorderTraversal(root *TreeNode) []int {
    result = make([]int, 0)
    preorder(root)
    return result
}

func preorder(root *TreeNode) {
    if root == nil {return}
    result = append(result, root.Val)
    preorder(root.Left)
    preorder(root.Right)
}

//解法 2 非递归
type SFrame struct {
    status int
    node   *TreeNode
}

var result []int
func preorderTraversal(root *TreeNode) []int{
    result = make([]int, 0)
    if root == nil {return result}
    stack := make([]SFrame, 0)
    stack = append(stack, SFrame{status: 1, node: root})
    for len(stack) > 0 {
        if stack[len(stack)-1].status == 1 {
            result = append(result, stack[len(stack)-1].node.Val)
            stack[len(stack)-1].status = 2
            if stack[len(stack)-1].node.Left != nil {
                stack = append(stack, SFrame{status: 1, node: stack[len(stack)-
1].node.Left})
            }
            continue
        }
        if stack[len(stack)-1].status == 2 {
            stack[len(stack)-1].status = 3
            if stack[len(stack)-1].node.Right != nil {
                stack = append(stack, SFrame{status:1, node:stack[len(stack)-
1].node.Right})
            }
            continue
        }
        if stack[len(stack)-1].status == 3 {
            stack = stack[:len(stack)-1]
        }
    }
    return result
}
```

94. 二叉树的中序遍历

```
var result []int
func inorderTraversal(root *TreeNode) []int {
    result = make([]int, 0)
    inorder(root)
    return result
}

func inorder(root *TreeNode) {
    if root == nil {return}
    inorder(root.Left)
    result = append(result, root.Val)
    inorder(root.Right)
}
```

145. 二叉树的后序遍历

```
var result []int
func postorderTraversal(root *TreeNode) []int {
    result = make([]int, 0)
    postorder(root)
    return result
}

func postorder(root *TreeNode) {
    if root == nil {return}
    postorder(root.Left)
    postorder(root.Right)
    result = append(result, root.Val)
}
```

589. N 叉树的前序遍历

```
var result []int
func preorder(root *Node) []int {
    result = make([]int, 0)
    r_preorder(root)
    return result
}

func r_preorder(root *Node,) {
    if root == nil {return }
    result = append(result, root.Val)
    for i, _ := range root.Children {
        r_preorder(root.Children[i])
    }
}
```

590. N 叉树的后序遍历

```
var result []int
func postorder(root *Node) []int {
    result = make([]int, 0)
    r_postorder(root)
    return result
}

func r_postorder(root *Node,) {
    if root == nil {return }
    for i, _ := range root.Children {
        r_postorder(root.Children[i])
    }
    result = append(result, root.Val)
}
```

剑指 Offer 32 - I. 从上到下打印二叉树

```
//解法 1 slice 实现 queue
func levelOrder(root *TreeNode) []int {
    if root == nil {return []int{}}
    result := make([]int, 0)
    queue := make([]*TreeNode, 0)
    queue = append(queue, root)
    for len(queue) > 0 {
        node := queue[0]
        queue = queue[1:]
        result = append(result, node.Val)
        if node.Left != nil {
            queue = append(queue, node.Left)
        }
        if node.Right != nil {
            queue = append(queue, node.Right)
        }
    }
    return result
}

//解法 2 list 实现 queue
func levelOrder(root *TreeNode) []int {
    if root == nil {return []int{}}
    result := make([]int, 0)
    queue := list.List{}
    queue.PushBack(root)
    for  queue.Len() > 0  {
        front := queue.Front()
        node := front.Value.(*TreeNode)
        result = append(result, node.Val)
        queue.Remove(front)
        if node.Left != nil {
            queue.PushBack(node.Left)
        }
        if node.Right != nil {
            queue.PushBack(node.Right)
        }
    }
    return result
}
```

102. 二叉树的层序遍历

```
func levelOrder(root *TreeNode) [][]int {
    result := make([][]int, 0)
    if root == nil {return result}
    queue := make([]*TreeNode, 0)
    queue = append(queue, root)
    for len(queue) > 0 {
        curLevelNodes := make([]int, 0)
        curLevelNum := len(queue)
        for i := 0; i < curLevelNum; i++ {
            treeNode := queue[0]
            queue = queue[1:]
```

```
                curLevelNodes = append(curLevelNodes, treeNode.Val)
                if treeNode.Left != nil {
                    queue = append(queue, treeNode.Left)
                }
                if treeNode.Right != nil {
                    queue = append(queue, treeNode.Right)
                }
            }
            result = append(result, curLevelNodes)

        }
        return result
}

//解法 2 dfs
var result  [][]int
func levelOrder(root *TreeNode) [][]int {
    result = make([][]int, 0)
    dfs(root, 0)
    return result
}

func dfs (root *TreeNode, level int) {
    if root == nil {return}
    if level > len(result)-1 {
        result = append(result, make([]int, 0))
    }
    result[level] = append(result[level], root.Val)
    dfs(root.Left, level+1)
    dfs(root.Right, level+1)
}
```

剑指 Offer 32 - III. 从上到下打印二叉树 III

```
var result  [][]int
func levelOrder(root *TreeNode) [][]int {
    result = make([][]int, 0)
    if root == nil {return result}
    stacks := make([][]*TreeNode, 2)
    for i := 0; i < 2; i++ {
        stacks[i] = make([]*TreeNode, 0)
    }
    turn := 0
    stacks[turn] = append(stacks[turn], root)
    for len(stacks[turn]) > 0 {
        curLevelNodes := make([]int, 0)
        for len(stacks[turn]) > 0 {
            treeNode := stacks[turn][len(stacks[turn])-1]
            stacks[turn] = stacks[turn][:len(stacks[turn])-1]
            curLevelNodes = append(curLevelNodes, treeNode.Val)
            if turn == 0 {
                if treeNode.Left != nil {
                    stacks[1] = append(stacks[1], treeNode.Left)
                }
                if treeNode.Right != nil {
                    stacks[1] = append(stacks[1], treeNode.Right)
                }
            } else {
                if treeNode.Right != nil {
                    stacks[0] = append(stacks[0], treeNode.Right)
                }
                if treeNode.Left != nil {
                    stacks[0] = append(stacks[0], treeNode.Left)
                }
            }
        }
        result = append(result, curLevelNodes)
        turn = (turn+1)%2
    }
    return result
}
```

429. N 叉树的层序遍历

```
var result  [][]int
func levelOrder(root *Node) [][]int {
    result = make([][]int, 0)
    if root == nil {return result}
    queue := make([]*Node, 0)
    queue = append(queue, root)
    for len(queue) > 0 {
        size := len(queue)
        level := make([]int, 0)
        for i := 0; i < size; i++ {
            node := queue[0]
            queue = queue[1:]
            level = append(level, node.Val)
            for j := 0; j < len(node.Children); j++ {
                queue = append(queue, node.Children[j])
            }
        }
        result = append(result, level)
    }
    return result
}
```

513. 找树左下角的值

```
func findBottomLeftValue(root *TreeNode) int {
    queue := make([]*TreeNode, 0)
    queue = append(queue, root)
    result := -1
    for len(queue) > 0 {
        node := queue[0]
        queue = queue[1:]
        result = node.Val
        if node.Right != nil {
            queue = append(queue, node.Right)
        }
        if node.Left != nil {
            queue = append(queue, node.Left)
        }
    }
    return result
}
```

104. 二叉树的最大深度

```
func maxDepth(root *TreeNode) int {
    if root == nil {
        return 0
    }
    return int(math.Max(float64(maxDepth(root.Left)), float64(maxDepth(root.Right))))
+ 1
}
```

559. N 叉树的最大深度

```
func maxDepth(root *Node) int {
    if root == nil {
        return 0
    }
    childrenMaxDepth := 0
    for i := 0; i < len(root.Children); i++ {
        depth := maxDepth(root.Children[i])
        if depth > childrenMaxDepth {
            childrenMaxDepth = depth
        }
    }
    return childrenMaxDepth+1
}
```

剑指 Offer 55 - II. 平衡二叉树

```
var balanced bool
func isBalanced(root *TreeNode) bool {
    balanced = true
    height(root)
    return balanced
}

func height(root *TreeNode) int{
    if root == nil {return 0}
    if balanced == false {return 0}
    leftHeight := height(root.Left)
    rightHeight := height(root.Right)
    if math.Abs(float64(leftHeight-rightHeight)) > 1 {
        balanced = false
    }
    return int(math.Max(float64(leftHeight), float64(rightHeight)))+1
}
```

617. 合并二叉树

```
func mergeTrees(root1 *TreeNode, root2 *TreeNode) *TreeNode {
    if root1 == nil {return root2}
    if root2 == nil {return root1}
    newNode := &TreeNode{Val:root1.Val+root2.Val}
    leftRoot := mergeTrees(root1.Left, root2.Left)
    rightRoot := mergeTrees(root1.Right, root2.Right)
    newNode.Left = leftRoot
    newNode.Right = rightRoot
    return newNode
}
```

226. 翻转二叉树

```
func invertTree(root *TreeNode) *TreeNode {
    if root == nil {return root}
    leftNode := invertTree(root.Left)
    rightNode := invertTree(root.Right)
    root.Right = leftNode
    root.Left = rightNode
    return root
}
```

101. 对称二叉树

```
func isSymmetric(root *TreeNode) bool {
    if root == nil {return true}
    return r_isSymmetric(root.Left, root.Right)
}

func r_isSymmetric(p, q *TreeNode) bool{
    if p == nil && q == nil {return true}
    if p != nil && q != nil && p.Val == q.Val {
        return r_isSymmetric(p.Right, q.Left) && r_isSymmetric(p.Left, q.Right)
    }
    return false
}
```

98. 验证二叉搜索树

```
var isValid bool
func isValidBST(root *TreeNode) bool {
    isValid = true
    if root == nil {return true}
    dfs(root)
    return isValid
}

func dfs(root *TreeNode) []int{
    min := root.Val
    max := root.Val
    if root.Left != nil {
        leftMinMax := dfs(root.Left)
        if isValid == false {return nil}
        if leftMinMax[1] >= root.Val {
            isValid = false
            return nil
        }
        min = leftMinMax[0]
    }
    if root.Right != nil {
        rightMinMax := dfs(root.Right)
        if isValid == false {return nil}
        if rightMinMax[0] <= root.Val {
            isValid = false
            return nil
        }
        max = rightMinMax[1]
    }
    return []int{min, max}
}
```

剑指 Offer 54. 二叉搜索树的第 k 大节点

```
var count int
var result int
func kthLargest (root *TreeNode, k int) int {
    count = 0
    result = 0
    inorder(root, k)
    return result
}

func inorder(root *TreeNode, k int) {
    if root == nil {return}
    inorder(root.Right, k)
    if count >= k {return}
    count++
    if count == k {
        result = root.Val
        return
    }
    inorder(root.Left, k)
}
```

[538. 把二叉搜索树转换为累加树]

```
var sum int
func convertBST(root *TreeNode) *TreeNode {
    sum = 0
    inorder(root)
    return root
}

func inorder(root *TreeNode) {
    if root == nil {return}
    inorder(root.Right)
    sum += root.Val
    root.Val = sum
    inorder(root.Left)
}
```

[面试题 04.06. 后继者]

```
var coming bool
var successor *TreeNode
func inorderSuccessor(root *TreeNode, p *TreeNode) *TreeNode {
    coming = false
    successor = nil
    inorder(root, p)
    return successor
}

func inorder(root, p *TreeNode) {
    if root == nil {return}
    inorder(root.Left, p)
    if successor != nil {return}
    if coming == true {
        successor = root
        coming = false
        return
    }
    if root == p {
        coming = true}
    inorder(root.Right, p)
}
```