236. 二叉树的最近公共祖先

```
var lca *TreeNode
func lowestCommonAncestor(root, p, q *TreeNode) *TreeNode {
    lca = nil
    traverse(root, p, q)
    return lca
}

func traverse(root, p, q *TreeNode) int{
    if root == nil {return 0}
    leftContains := traverse(root.Left, p, q)
    if lca != nil { //提前退出
        return 2
    }
    rightContains := traverse(root.Right, p, q)
    if lca != nil { //提前退出
        return 2
    }
    rootContains := 0
    if root == p || root == q {
        rootContains = 1
    }
    if rootContains == 0 && leftContains == 1 && rightContains == 1 {
        lca = root
    }
    if rootContains == 1 && (leftContains == 1 || rightContains == 1) {
        lca = root
    }
    return leftContains + rightContains + rootContains
}
```

剑指 Offer 68 - I. 二叉搜索树的最近公共祖先

```
func lowestCommonAncestor(root, p, q *TreeNode) *TreeNode {
    x := root
    for {
        if p.Val < x.Val && q.Val < x.Val {
            x = x.Left
        } else if p.Val > x.Val && q.Val > x.Val {
            x = x.Right
        } else {
            return x
        }
    }
}

//解法 2 递归
func lowestCommonAncestor(root, p, q *TreeNode) *TreeNode {
    if p == root || q == root ||
        (p.Val < root.Val && root.Val < q.Val) ||
        (q.Val < root.Val && root.Val < p.Val) {
        return root
    }
    if p.Val < root.Val && q.Val < root.Val {
        return lowestCommonAncestor(root.Left, p, q)
    } else {
        return lowestCommonAncestor(root.Right, p, q)
```

```
    }
}
```

114. 二叉树展开为链表

```
var dummyHead *TreeNode
var tail *TreeNode
func flatten(root *TreeNode)  {
    dummyHead = &TreeNode{}
    tail = dummyHead
    preorder(root)
}

func preorder(root *TreeNode) {
    if root == nil {return}
    left := root.Left
    right := root.Right
    tail.Right = root
    tail = root
    root.Left = nil
    preorder(left)
    preorder(right)
}
```

面试题 17.12. BiNode

```
var dummyHead *TreeNode
var tail *TreeNode
func convertBiNode(root *TreeNode) *TreeNode{
    dummyHead = &TreeNode{}
    tail = dummyHead
    inorder(root)
    return dummyHead.Right
}

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

剑指 Offer 36. 二叉搜索树与双向链表

```
var dummyHead *TreeNode
var tail *TreeNode
func treeToDoublyList(root *TreeNode) *TreeNode{
    dummyHead = &TreeNode{}
    tail = dummyHead
    if root == nil {return nil}
    inorder(root)
    tail.Right = dummyHead.Right
    dummyHead.Right.Left = tail
    return dummyHead.Right
}

func inorder(root *TreeNode) {
```

```
    if root == nil {return}
    inorder(root.Left)
    root.Left = tail
    tail.Right = root
    tail = root
    inorder(root.Right)
}
```

面试题 04.03. 特定深度节点链表

```
func listOfDepth(tree *TreeNode) []*ListNode {
    if tree == nil {return []*ListNode{}}
    result := make([]*ListNode, 0)
    queue := make([]*TreeNode, 0)
    queue = append(queue, tree)
    for len(queue) > 0 {
        dummyHead := &ListNode{}
        tail := dummyHead
        curLevelNum := len(queue)
        for i := 0; i < curLevelNum; i++ {
            treeNode := queue[0]
            queue = queue[1:]
            tail.Next = &ListNode{Val: treeNode.Val}
            tail = tail.Next
            if treeNode.Left != nil {
                queue = append(queue, treeNode.Left)
            }
            if treeNode.Right != nil {
                queue = append(queue, treeNode.Right)
            }
        }
        result = append(result, dummyHead.Next)
    }
    return result
}
```

105.从前序与中序遍历序列构造二叉树

```
func buildTree(preorder []int, inorder []int) *TreeNode {
    return myBulildTree(preorder, 0, len(preorder)-1, inorder, 0, len(inorder)-1)
}

func myBulildTree(preorder []int, i, j int, inorder []int, p, r int) *TreeNode{
    if i > j {return nil}
    root := &TreeNode{Val: preorder[i]}
    q := p
    for inorder[q] != preorder[i] {
        q++
    }
    leftTreeSize := q-p
    leftNode := myBulildTree(preorder, i+1, i+leftTreeSize, inorder, p, q-1)
    rightNode := myBulildTree(preorder, i+leftTreeSize+1, j, inorder, q+1, r)
    root.Left = leftNode
    root.Right = rightNode
    return root
}
```

106. 从中序与后序遍历序列构造二叉树

```go
func buildTree(inorder []int, postorder []int) *TreeNode {
    return myBuildTree(postorder, 0, len(postorder)-1, inorder, 0, len(inorder)-1)
}

func myBuildTree(postorder []int, i, j int, inorder []int, p, r int) *TreeNode{
    if i > j {return nil}
    root := &TreeNode{Val:postorder[j]}
    q := p
    for inorder[q] != postorder[j] {
        q++
    }
    leftTreeSize := q-p
    leftNode := myBuildTree(postorder, i, i+leftTreeSize-1, inorder, p, q-1)
    rightNode := myBuildTree(postorder, i+leftTreeSize, j-1, inorder, q+1, r)
    root.Left = leftNode
    root.Right = rightNode
    return root
}
```

889.根据前序和后序遍历构造二叉树

```go
func constructFromPrePost(preorder []int, postorder []int) *TreeNode {
    return myBuildTree(preorder, 0, len(preorder)-1, postorder, 0, len(postorder)-1)
}

func myBuildTree(pre []int, i, j int, post []int, p, r int) *TreeNode{
    if i > j {return nil}
    root := &TreeNode{Val:pre[i]}
    if i == j {return root}
    q := p
    for post[q] != pre[i+1] {
        q++
    }
    leftTreeSize := q-p+1
    leftNode := myBuildTree(pre, i+1, i+leftTreeSize, post, p, q)
    rightNode := myBuildTree(pre, i+leftTreeSize+1, j, post, q+1, r-1)
    root.Left = leftNode
    root.Right = rightNode
    return root
}
```

剑指 Offer33.二叉搜索树的后序遍历序列

```go
func verifyPostorder(postorder []int) bool {
    return myVerify(postorder, 0, len(postorder)-1)
}

func myVerify(postorder []int, i, j int) bool{
    if i >= j {return true}
    //postorder[j]是根节点，先分离出左子树[i,k-1]
    k := i
    for k < j && postorder[k] < postorder[j] {
        k++
    }
    //验证[k,j-1]满足右子树的要求，都大于 postorder[j]
    p := k
    for p < j {
        if postorder[p] < postorder[j] {
            return false
```

```
        }
        p++
    }
    //递归验证左右子树是否满足 BST 的要求
    leftValid := myVerify(postorder, i, k-1)
    if leftValid == false {return false}
    rightVlid := myVerify(postorder, k, j-1)
    return rightVlid
}
```

543.二叉树的直径

```
//转化成求数的最大高度
var result int
func diameterOfBinaryTree(root *TreeNode) int {
    result = 0
    calMaxHeight(root)
    return result
}

func calMaxHeight(root *TreeNode) int{
    if root == nil {return 0}
    maxLeftHeight := calMaxHeight(root.Left)
    maxRightHeight := calMaxHeight(root.Right)
    diameter := maxLeftHeight + maxRightHeight
    if diameter > result {result = diameter}
    return int(math.Max(float64(maxLeftHeight), float64(maxRightHeight))) + 1
}
```

124. 二叉树中的最大路径和

```
var result int
func maxPathSum(root *TreeNode) int {
    result = -1001
    dfs(root)
    return result
}

func dfs(root *TreeNode) int {
    if root == nil {return 0}
    leftMaxPath := dfs(root.Left)
    rightMaxPath := dfs(root.Right)
    max := root.Val
    if leftMaxPath > 0 {max += leftMaxPath}
    if rightMaxPath > 0 {max += rightMaxPath}
    if max > result {result = max}
    ret := root.Val
    if ret < leftMaxPath+root.Val {ret = leftMaxPath+root.Val}
    if ret < rightMaxPath+root.Val {ret = rightMaxPath+root.Val}
    return ret
}
```

剑指 Offer 34. 二叉树中和为某一值的路径

```
var result [][]int
func pathSum(root *TreeNode, target int) [][]int {
    result = make([][]int, 0)
    if root == nil {return result}
```

```
    dfs(root, target, []int{}, 0)
    return result
}

func dfs(root *TreeNode, sum int, path []int, pathSum int) {
    path = append(path, root.Val)
    pathSum += root.Val
    if root.Left == nil && root.Right == nil {
        if pathSum == sum {
            pathSnapshot := make([]int, len(path))
            copy(pathSnapshot, path)
            result = append(result, pathSnapshot)
        }
        path = path[:len(path)-1]
        return
    }
    if root.Left != nil {
        dfs(root.Left, sum, path, pathSum)
    }
    if root.Right != nil {
        dfs(root.Right, sum, path, pathSum)
    }
    path = path[:len(path)-1]
    //pathSum -= root.Val 不需要这句
}
```