链滴

日刷 leetcode-- 简单版（四）

作者：InkDP
原文链接：https://ld246.com/article/1568444798713
来源网站：链滴

返回总目录

日刷leetcode–简单版

---

88. 合并两个有序数组

题目描述

给定两个有序整数数组 *nums1* 和 *nums2*，将 *nums2* 合并到 *nums1* 中，使得 *num1* 成为一个有序数组。

说明:

- 初始化 *nums1* 和 *nums2* 的元素数量分别为 *m* 和 *n*。
- 你可以假设 *nums1* 有足够的空间（空间大小大于或等于 *m* + *n*）来保存 *nums2* 中的元素。

示例:

输入:
nums1 = [1,2,3,0,0,0], m = 3
nums2 = [2,5,6], n = 3

输出: [1,2,2,3,5,6]

解题思路

- 采用双指针，从前往后，如果nums2中当前值小于nums1中的值，这插入并后移

示例代码

```
func merge(nums1 []int, m int, nums2 []int, n int) []int {
    if n == 0 {
        return nums1
    }
    i, j := 0, 0
    for ; i < len(nums1) && j < n; i++ {
        if nums1[i] > nums2[j] {
            reverses(nums1[i:])
            nums1[i] = nums2[j]
            j++
        } else {
            continue
        }
    }
    for j < n {
        nums1[m+j] = nums2[j]
        j++
    }
    return nums1
}

func reverses(nums []int) {
    for i := len(nums) - 1; i > 0; i-- {
        nums[i] = nums[i-1]
    }
}
```

运行结果

执行用时 :4 ms, 在所有 Go 提交中击败了73.14%的用户

内存消耗 :3.6 MB, 在所有 Go 提交中击败了84.42%的用户

100. 相同的树

题目描述

给定两个二叉树，编写一个函数来检验它们是否相同。

如果两个树在结构上相同，并且节点具有相同的值，则认为它们是相同的。

示例 1:

```
输入:   1     1
      /\    /\
     2 3   2 3

    [1,2,3],  [1,2,3]

输出: true
```

示例 2:

```
输入:   1     1
      /       \
     2         2

    [1,2],   [1,null,2]

输出: false
```

示例 3:

```
输入:   1     1
      /\    /\
     2 1   1 2

    [1,2,1],  [1,1,2]

输出: false
```

解题思路

- 有两中情况下可直接返回
  1. p与q同时为空时为 True
  2. p或q只有一个为空时 False
- p与q同时不为空是，判断其Val，同时递归判断两个的Left与Right

示例代码

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

运行结果

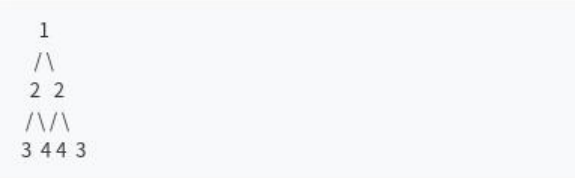
执行用时 :0 ms, 在所有 Go 提交中击败了100.00%的用户
内存消耗 :2.1 MB, 在所有 Go 提交中击败了87.07%的用户

101. 对称二叉树

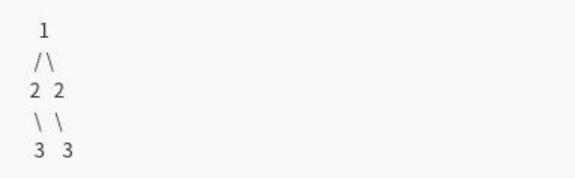
题目描述

给定一个二叉树，检查它是否是镜像对称的。

例如，二叉树 [1,2,2,3,4,4,3] 是对称的。



但是下面这个 [1,2,2,null,3,null,3] 则不是镜像对称的:



说明:

如果你可以运用递归和迭代两种方法解决这个问题，会很加分。

解题思路--递归

- 这道题与上一题很像，上一题是要求树完全相等，此题是要求对称
- 两树相等时为 Left == Left,Right == Right，而对称则是Left == Right，Right == Left

示例代码

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

func isLeftRight(L *TreeNode, R *TreeNode) bool {
    if L == nil && R == nil {
        return true
    }

    if L == nil || R == nil {
        return false
    }
```

```
    if L.Val != R.Val{
        return false
    }
    return isLeftRight(L.Left, R.Right) && isLeftRight(L.Right, R.Left)
}
```

运行结果

执行用时 :0 ms, 在所有 Go 提交中击败了100.00%的用户

内存消耗 :3 MB, 在所有 Go 提交中击败了60.14%的用户

解题思路--迭代

- 迭代我也不是很懂，就摘抄了一个，配合注释看一看

示例代码

```
func isSymmetric(root *TreeNode) bool { // 迭代方法，bfs
    L := []*TreeNode{}       // 从左向右遍历顺序的队列
    R := []*TreeNode{}       // 从右向左遍历顺序的队列
    L = append(L, root)      // 加入初始节点
    R = append(R, root)
    for len(L)!=0 && len(R)!=0 {   // bfs
        Lv,rv := L[0], R[0]    // 不同遍历顺序的队列，队首出队
        L, R = L[1:], R[1:]      // 删除队首元素
        if Lv==nil && rv==nil {  // 空节点，不添加节点
            continue
        } else if Lv!=nil && rv!=nil && Lv.Val == rv.Val {    // 比较两种遍历顺序的出队节点，如果
同，继续搜索
            L = append(L, Lv.Left, Lv.Right)
            R = append(R, rv.Right, rv.Left)
        } else {                    // 如果不同，证明不是镜像二叉树
            return false
        }
    }
    if len(L)==0 && len(R)==0 {
        return true
    } else {
        return false
    }
}
```

运行结果

执行用时 :0 ms, 在所有 Go 提交中击败了100.00%的用户

内存消耗 :3.2 MB, 在所有 Go 提交中击败了5.80%的用户

104. 二叉树的最大深度

题目描述

给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

说明: 叶子节点是指没有子节点的节点。

示例:
给定二叉树 `[3,9,20,null,null,15,7]`，

```
    3
   / \
  9  20
    /  \
   15   7
```

返回它的最大深度 3 。

解题思路

- 使用递归查看当前值是否为空，为空则返回0，接受到递归返回的0就加1
- 毕竟树的左右，看那边更大，返回大的一边。

示例代码

```go
func maxDepth(root *TreeNode) int {
    if root == nil {
        return 0
    }
    Ll := maxDepth(root.Left) + 1
    Lr := maxDepth(root.Right) + 1
    if Ll > Lr {
        return Ll
    }
    return Lr
}
```

运行结果

执行用时 :8 ms, 在所有 Go 提交中击败了79.37%的用户

内存消耗 :4.4 MB, 在所有 Go 提交中击败了61.74%的用户

107. 二叉树的层次遍历 II

题目描述

给定一个二叉树，返回其节点值自底向上的层次遍历。 （即按从叶子节点所在层到根节点所在的层，逐层从左向右遍历）

例如：
给定二叉树 `[3,9,20,null,null,15,7]`，

```
    3
   / \
  9  20
    /  \
   15   7
```

返回其自底向上的层次遍历为：

```
[
  [15,7],
  [9,20],
  [3]
]
```

解题思路

- 看下面代码注释

示例代码

```
func levelOrderBottom(root *TreeNode) (res [][]int) {
    if root == nil {
        return
    }

    var node []*TreeNode
    node = append(node, root)
    for len(node) > 0 { // node不为0就一直循环
        len := len(node)
        var tmp []int
        for i:= 0; i < len; i++ {
        // 循环次数等于每一层的节点数，每次都取第一个node，因为下面会将第一个node删除
            indexNode := node[0]
            node = node[1:]  // 将第一个node删除
            tmp = append(tmp, indexNode.Val) // 记录节点值
            if indexNode.Left != nil { // 如果左子树不为空，将左子树添加node
                node = append(node, indexNode.Left)
            }
            if indexNode.Right != nil { // 如果右边树不为空，将右子树添加node
                node = append(node, indexNode.Right)
            }
        }
        res = append(res, tmp)
    }
    var res1 [][]int
    for i:= len(res)-1; i >= 0; i-- {
        res1 = append(res1, res[i])
    }
    return res1
```

```
}
```

运行结果

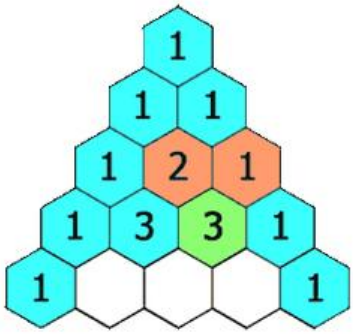
执行用时 :4 ms, 在所有 Go 提交中击败了85.64%的用户
内存消耗 :6 MB, 在所有 Go 提交中击败了87.36%的用户

对树不是很了解，树相关题目后面再做

118. 杨辉三角

题目描述

给定一个非负整数 *numRows*，生成杨辉三角的前 *numRows* 行。

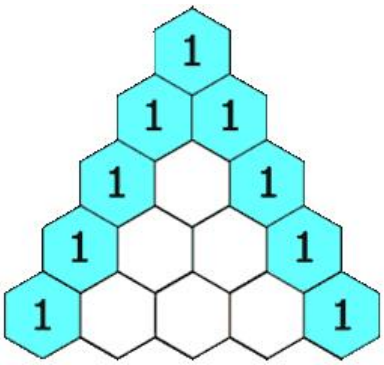


在杨辉三角中，每个数是它左上方和右上方的数的和。

示例:

```
输入: 5
输出:
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
```

解题思路

- 

示例代码

```
func generate(numRows int) [][]int {
    var arr [][]int
    if numRows == 0 {
        return arr
    }

    arr = append(arr, []int{1})
    if numRows == 1 {
        return arr
    }

    for i := 1; i < numRows; i++ {
        var row []int
        row = append(row, 1)
        for j := 1; j < i; j++ {
            tmp := arr[i-1][j-1] + arr[i-1][j]
            fmt.Println(tmp)
            row = append(row, tmp)
        }
        row = append(row, 1)
        arr = append(arr, row)
    }
    return arr
}
```

运行结果

执行用时 :0 ms, 在所有 Go 提交中击败了100.00%的用户

内存消耗 :2.3 MB, 在所有 Go 提交中击败了47.54%的用户