TalkGo算法之美第三期

第一周线上讨论会

主持人：小林

第一周--周一题目--树的构造

108. 将有序数组转换为二叉搜索树

将一个按照升序排列的有序数组，转换为一棵高度平衡二叉搜索树。

本题中，一个高度平衡二叉树是指一个二叉树*每个节点*的左右两个子树的高度差的绝对值不超过 1。

示例:

给定有序数组: `[-10,-3,0,5,9]`,

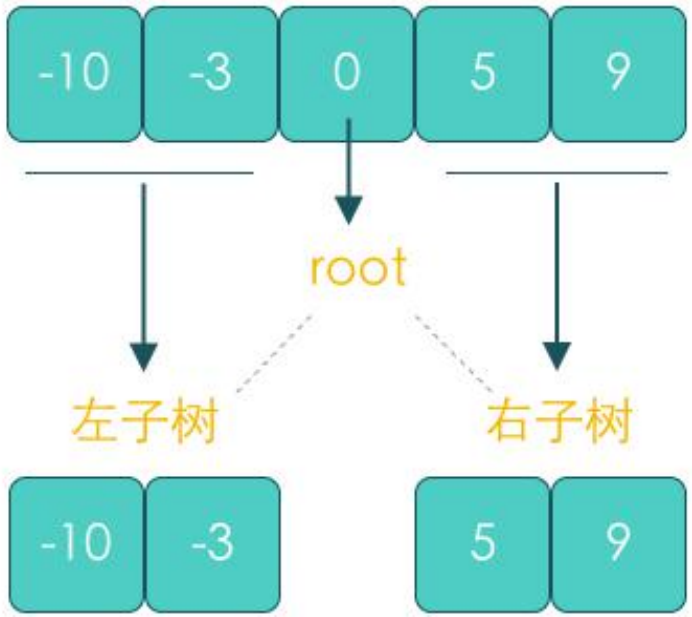
一个可能的答案是：`[0,-3,9,-10,null,5]`，它可以表示下面这个高度平衡二叉搜索树：

```
      0
     / \
   -3   9
   /   /
 -10  5
```

第一周--周一题目--树的构造

108. 将有序数组转换为二叉搜索树

思路：使用递归，从中间将数组分为两部分，左边部分作为中间节点的右子树，右边部分作为中间节点右子树。



第一周--周一题目--树的构造

108. 将有序数组转换为二叉搜索树

```go
func sortedArrayToBST(nums []int) *TreeNode {
    if len(nums) == 0 {
        return nil
    }
    mid := len(nums)/2
    node := &TreeNode{nums[mid], nil, nil}
    node.Left = sortedArrayToBST(nums[:mid])
    node.Right = sortedArrayToBST(nums[mid+1:])
    return node
}
```

第一周--周一题目--树的构造

附加题：95. 不同的二叉搜索树 II

给定一个整数 *n*，生成所有由 1 ... *n* 为节点所组成的 **二叉搜索树** 。

示例：

```
输入：3
输出：
[
  [1,null,3,2],
  [3,2,null,1],
  [3,1,null,null,2],
  [2,1,3],
  [1,null,2,null,3]
]
解释：
以上的输出对应以下 5 种不同结构的二叉搜索树：

   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

第一周--周一题目--树的构造

附加题：95. 不同的二叉搜索树 II

思路：依然使用递归，不过递归返回的是子树的列表，需要稍作处理

```go
func generateTrees(n int) []*TreeNode {
    if n == 0 {
        return nil
    }
    return dfs(1, n)
}

func dfs(left, right int) []*TreeNode {
    if left > right {
        return []*TreeNode{nil}
    }
    res := []*TreeNode{}
    for i:=left; i<=right; i++ {
        leftTrees := dfs(left, i-1)
        rightTrees := dfs(i+1, right)
        for _, lnode := range leftTrees {
            for _, rnode := range rightTrees {
                node := &TreeNode{i, lnode, rnode}
                res = append(res, node)
            }
        }
    }
    return res
}
```

第一周——周三题目——树的遍历——前中后序遍历

144. 二叉树的前序遍历
94. 二叉树的中序遍历
145. 二叉树的后序遍历

递归写法：

```
func Traversal(root *TreeNode) (res []int) {
    var helper func(node *TreeNode)
    helper = func(node *TreeNode) {
        if node == nil {
            return
        }
        res = append(res, node.Val)       // 放在这就是前序
        helper(node.Left)
        //res = append(res, node.Val)     // 放在这就是中序
        helper(node.Right)
        //res = append(res, node.Val)     // 放在这就是后序
    }
    helper(root)
    return
}
```

第一周——周三题目——树的遍历——前中后序遍历

144. 二叉树的前序遍历

非递归写法：

```go
// 前序
func preorderTraversal(root *TreeNode) (ans []int) {
    ptr := root
    stack := []*TreeNode{}
    for ptr != nil || len(stack) > 0 {
        if ptr != nil {
            ans = append(ans, ptr.Val)      // 输出值
            stack = append(stack, ptr)      // 入栈
            ptr = ptr.Left                  // 访问左子树
        } else {
            top := stack[len(stack)-1]      // 出栈
            stack = stack[:len(stack)-1]
            ptr = top.Right                 // 访问右子树
        }
    }
    return
}
```

第一周——周三题目——树的遍历——前中后序遍历

94. 二叉树的中序遍历

非递归写法：

```go
// 中序
func inorderTraversal(root *TreeNode) (ans []int) {
    ptr := root
    stack := []*TreeNode{}
    for ptr != nil || len(stack) > 0 {
        if ptr != nil {
            stack = append(stack, ptr)
            ptr = ptr.Left
        } else {
            top := stack[len(stack)-1]
            stack = stack[:len(stack)-1]
            ans = append(ans, top.Val)
            ptr = top.Right
        }
    }
    return
}
```

第一周——周三题目——树的遍历——前中后序遍历

94. 二叉树的后序遍历

非递归写法：

```go
// 后序
func postorderTraversal(root *TreeNode) (ans []int) {
    ptr := root
    stack := []*TreeNode{}
    var pre *TreeNode                          // 用于保存前一个已经出栈的节点
    for ptr != nil || len(stack) > 0 {
        if ptr != nil {
            stack = append(stack, ptr)
            ptr = ptr.Left
        } else {
            top := stack[len(stack)-1]
            if top.Right != nil && top.Right != pre {    // 没有
                ptr = top.Right
            } else {
                ans = append(ans, top.Val)
                stack = stack[:len(stack)-1]
                pre = top
            }
        }
    }
    return
}
```

第一周——周三题目——树的遍历——前中后序遍历

94. 二叉树的中序遍历

Moriis写法：

```
func preorderTraversal(root *TreeNode) []int {
    var max *TreeNode
    var res []int
    for root != nil {
        if root.Left == nil {
            res = append(res, root.Val)
            root = root.Right
        } else {
            max = root.Left
            for max.Right != nil {
                max = max.Right
            }
           root.Right, max.Right = root.Left, root.Right
           root.Left = nil
        }
    }
    return res
}
```

第一周——周三题目——树的遍历——层次遍历

102. 二叉树的层序遍历

给你一个二叉树，请你返回其按 **层序遍历** 得到的节点值。 （即逐层地，从左到右访问所有节点）。

示例：
二叉树：`[3,9,20,null,null,15,7]`，

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

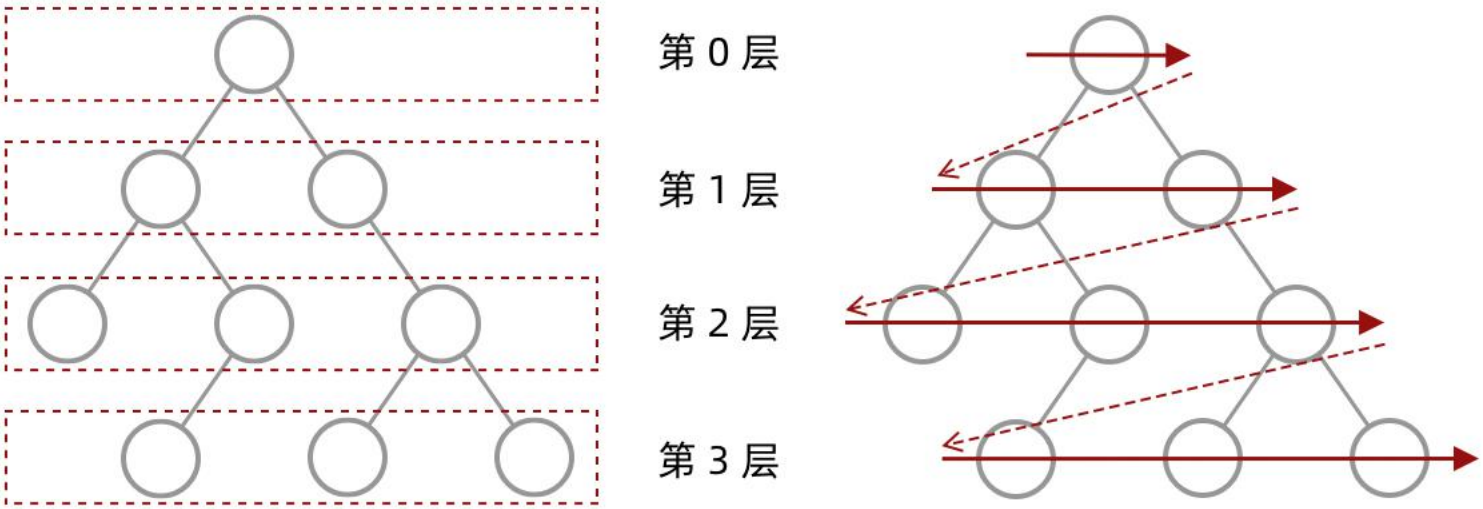
返回其层序遍历结果：

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

第一周——周三题目——树的遍历——层次遍历

102. 二叉树的层序遍历

思路：利用队列可以实现对树的遍历，每一层都保存到一个队列中，同层结点的子节点也会保存到同一个队列



第一周——周三题目——树的遍历——层次遍历

102. 二叉树的层序遍历

代码：

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

    queue, res := []*TreeNode{root}, make([][]int, 0)

    for len(queue)>0 {
        n := len(queue)
        r := make([]int, n)
        for i:=0; i<n; i++ {
            r[i] = queue[0].Val
            if queue[0].Left != nil {
                queue = append(queue, queue[0].Left)
            }
            if queue[0].Right != nil {
                queue = append(queue, queue[0].Right)
            }
            queue = queue[1:]
        }
        res = append(res, r)
    }
    return res
}
```

第一周——周三题目——树的遍历——层次遍历

102. 二叉树的层序遍历

代码：

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

    queue, res := []*TreeNode{root}, make([][]int, 0)

    for len(queue)>0 {
        n := len(queue)
        r := make([]int, n)
        for i:=0; i<n; i++ {
            r[i] = queue[0].Val
            if queue[0].Left != nil {
                queue = append(queue, queue[0].Left)
            }
            if queue[0].Right != nil {
                queue = append(queue, queue[0].Right)
            }
            queue = queue[1:]
        }
        res = append(res, r)
    }
    return res
}
```

第一周——周三题目——树的遍历——层次遍历

103. 二叉树的锯齿形层序遍历

给定一个二叉树，返回其节点值的锯齿形层序遍历。（即先从左往右，再从右往左进行下一层遍历，以此类推，层与层之间交替进行）。

例如：
给定二叉树 `[3,9,20,null,null,15,7]`，

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

返回锯齿形层序遍历如下：

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

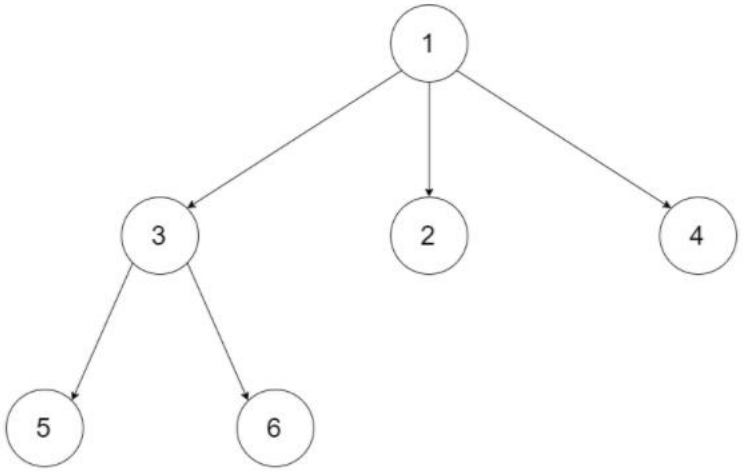
第一周——周三题目——树的遍历——层次遍历

429. N 叉树的层序遍历

给定一个 N 叉树，返回其节点值的*层序遍历*。（即从左到右，逐层遍历）。

树的序列化输入是用层序遍历，每组子节点都由 null 值分隔（参见示例）。

示例 1：




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

第一周——周三题目——树的遍历——层次遍历

429. N 叉树的层序遍历

```go
func levelOrder(root *Node) [][]int {
    if root == nil {
        return nil
    }
    queue, res := []*Node{root}, [][]int{}
    for len(queue)>0 {
        n := len(queue)
        temp := make([]int, n)
        for i:=0; i<n; i++ {
            temp[i] = queue[0].Val
            if len(queue[0].Children) > 0 {
                queue = append(queue, queue[0].Children...)
            }
            queue = queue[1:]
        }
        res = append(res, temp)
    }
    return res
}
```