春节7天练讲Day5：二叉树和堆

你好，我是王争。春节假期进入尾声了。你现在是否已经准备返回工作岗位了呢？今天更新的是测试题的第五篇，我们继续来复习。

关于二叉树和堆的7个必知必会的代码实现

二叉树

- 实现一个二叉查找树，并且支持插入、删除、查找操作
- 实现查找二叉查找树中某个节点的后继、前驱节点
- 实现二叉树前、中、后序以及按层遍历

堆

- 实现一个小顶堆、大顶堆、优先级队列
- 实现堆排序
- 利用优先级队列合并K个有序数组
- 求一组动态数据集合的最大Top K

对应的LeetCode练习题（@Smallfly 整理）

- Invert Binary Tree（翻转二叉树）

英文版：https://leetcode.com/problems/invert-binary-tree/

中文版：https://leetcode-cn.com/problems/invert-binary-tree/

- Maximum Depth of Binary Tree（二叉树的最大深度）

英文版：https://leetcode.com/problems/maximum-depth-of-binary-tree/

中文版：https://leetcode-cn.com/problems/maximum-depth-of-binary-tree/

- Validate Binary Search Tree（验证二叉查找树）

英文版：https://leetcode.com/problems/validate-binary-search-tree/

中文版：https://leetcode-cn.com/problems/validate-binary-search-tree/

- Path Sum（路径总和）

英文版：https://leetcode.com/problems/path-sum/

中文版：https://leetcode-cn.com/problems/path-sum/

做完题目之后，你可以点击“请朋友读”，把测试题分享给你的朋友。

祝你取得好成绩！明天见！

精选留言

李皮皮皮皮皮

平衡树的各种操作太烧脑了，左旋右旋，红黑树就更别提了。过段时间就忘。

2019-02-09 09:00

kai

树的前中后序遍历-非递归实现：

```
import java.util.Stack;


public class TreeTraversal {
public static class Node {
public int value;
public Node left;
public Node right;
public Node(int value) {
this.value = value;
}
}
// 二叉树的非递归遍历
public static void preOrder(Node head) {
System.out.print("pre-order: ");
if (head == null) {
return;
}
Stack<Node> s = new Stack<>();
s.push(head);
while (!s.isEmpty()) {
head = s.pop();
System.out.print(head.value + " ");
if (head.right != null) {
s.push(head.right);
}

if (head.left != null) {
s.push(head.left);
}
}
System.out.println();
}

public static void inOrder(Node head) {
System.out.print("in-order: ");
if (head == null) {
return;
}
Stack<Node> s = new Stack<>();
while (!s.isEmpty() || head != null) {
if (head != null) {
s.push(head);
head = head.left;
} else {
head = s.pop();
System.out.print(head.value + " ");
head = head.right;
}
}
```

```
System.out.println();
}

public static void postOrder(Node head) {
System.out.print("pos-order: ");
if (head == null) {
return;
}

Stack<Node> tmp = new Stack<>();
Stack<Node> s = new Stack<>();

tmp.push(head);
while(!tmp.isEmpty()) {
head = tmp.pop();
s.push(head);

if (head.left != null) {
tmp.push(head.left);
}

if (head.right != null) {
tmp.push(head.right);
}
}

while (!s.isEmpty()) {
System.out.print(s.pop().value + " ");
}

System.out.println();
}
}
```

2019-02-11 11:06

kai

树的前中后序遍历-递归实现：

```
public class TreeTraversal {

public static class Node {
public int value;
public Node left;
public Node right;

public Node(int value) {
this.value = value;
}
}

// 二叉树的递归遍历
```

```
public static void preOrderRecursive(Node head) {
if (head == null) {
return;
}

System.out.print(head.value + " ");
preOrderRecursive(head.left);
preOrderRecursive(head.right);
}

public static void inOrderRecursive(Node head) {
if (head == null) {
return;
}

inOrderRecursive(head.left);
System.out.print(head.value + " ");
inOrderRecursive(head.right);
}

public static void postOrderRecursive(Node head) {
if (head == null) {
return;
}

postOrderRecursive(head.left);
postOrderRecursive(head.right);
System.out.print(head.value + " ");
}

}
```

2019-02-11 11:05

kai

今天看了一下这一节的题目，发现校招面试的时候都考过，今天又刷了一下，总结了一波，相应的知识点也总结了一下~

2019-02-10 02:29

纯洁的憎恶

今天的题目很适合递归实现，当然递归公式离代码实现还是存在一定距离。

1.翻转二叉树（T）｛

当T为Null时则返回；

翻转二叉树（T的左子树）；

翻转二叉树（T的右子树）；

若T不为叶节点，则交换T的左右子树位置；

}

2.最大深度（T）｛

当T为Null时，return 0；

return Max（最大深度（T左子树）+1，最大深度（T右子树）+1）；

}

函数返回值即为最大深度。

3.验证二叉查找树（T，&最大值，&最小值） {
当T为Null时，return true；
当T为叶节点时，最小值=最大值=当前节点，返回true；
左最大值=左最小值=T的值；
验证二叉查找树（T的左子树，&左最大值，&左最小值）；
右最大值=右最小值=T的值；
验证（T的右子树，&右最大值，&右最小值）；
T的值小于等于右最小值，并且大于等于左最大值时，最大值=右最大值，最小值=左最小值，之后返回true，否则返回false并结束。
}
函数最终返回true则验证成功。

4.计算路径和（T，sum） {
若T为Null返回false；
若T是叶节点，如果sum+T的值=目标值则返回true并结束，否则返回false；
计算路径和（T的左子树，sum+T的值）；
计算路径和（T的右子树，sum+T的值）；
}
计算路径和（T，0）返回true时则存在于目标值相同的路径之和；
2019-02-10 00:32

abner
java实现二叉树前序、中序、后序和层次遍历
代码如下：

```
package tree;

import java.util.LinkedList;
import java.util.Queue;

public class BinaryTree {

private Node root = null;

public static class Node {

private String data;
private Node left;
private Node right;

public Node(String data, Node left, Node right) {
this.data = data;
this.left = left;
this.right = right;
}
}

public void preOrder(Node root) {
if (null == root) {
return ;
}
System.out.print(root.data + " ");
preOrder(root.left);
```

```
preOrder(root.right);
}

public void inOrder(Node root) {
if (null == root) {
return ;
}
inOrder(root.left);
System.out.print(root.data + " ");
inOrder(root.right);
}

public void postOrder(Node root) {
if (null == root) {
return ;
}
postOrder(root.left);
postOrder(root.right);
System.out.print(root.data + " ");
}

public void traverseByLayer(Node root) {
if (null == root) {
return ;
}
Queue<Node> queue = new LinkedList<Node>();
queue.add(root);
while (!queue.isEmpty()) {
Node pNode = queue.peek();
System.out.print(pNode.data + " ");
queue.poll();
if (root.left != null) {
queue.add(root.left);
}
if (root.right != null) {
queue.add(root.right);
}
}
}
}
```

2019-02-14 23:13

拉欧

Path Sum（路径总和）go 语言实现

```
func hasPathSum(root *TreeNode, sum int) bool {

if root==nil{
return false
}
if root.Left==nil && root.Right==nil{
if root.Val==sum{
```

```
return true
}else{
return false
}

}
left:=false
if root.Left!=nil{
left=hasPathSum(root.Left,sum-root.Val)
}
right:=false
if root.Right!=nil{
right=hasPathSum(root.Right,sum-root.Val)
}
return left || right
}
```

2019-02-14 15:17

拉欧

Validate Binary Search Tree（验证二叉查找数） go语言实现

```
func isValidBST(root *TreeNode) bool {

if root==nil{
return true
}
less:=true
more:=true
if root.Left!=nil{
less=JudgeLess(root.Left,root.Val)
}
if root.Right!=nil{
more=JudgeMore(root.Right,root.Val)
}
if ! (less && more){
return false
}else{
return isValidBST(root.Left) && isValidBST(root.Right)
}
}

func JudgeLess(root *TreeNode,num int) bool{

if root.Val>=num{
return false
}
if root.Left!=nil && root.Right!=nil{
return JudgeLess(root.Left,num) && JudgeLess(root.Right,num)
}else if root.Left!=nil{
return JudgeLess(root.Left,num)
}else if root.Right!=nil{
return JudgeLess(root.Right,num)
```

```
}else{
return true
}
}

func JudgeMore(root *TreeNode,num int) bool{
if root.Val<=num{
return false
}
if root.Left!=nil && root.Right!=nil{
return JudgeMore(root.Left,num) && JudgeMore(root.Right,num)
}else if root.Left!=nil{
return JudgeMore(root.Left,num)
}else if root.Right!=nil{
return JudgeMore(root.Right,num)
}else{
return true
}
}
```

2019-02-14 14:57

拉欧

Invert Binary Tree（翻转二叉树） go 语言实现

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

2019-02-14 11:20

你看起来很好吃

路径之和python实现：

```
# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, x):
# self.val = x
# self.left = None
# self.right = None

class Solution:
def hasPathSum(self, root: 'TreeNode', sum: 'int') -> 'bool':
if not root:
return False

if not root.left and not root.right and root.val == sum:
```

return True

sum -= root.val

return self.hasPathSum(root.left, sum) or self.hasPathSum(root.right, sum)
2019-02-10 15:38

你看起来很好吃
二叉树最大深度python实现，使用递归
class Solution:

def maxDepth(self, root: 'TreeNode') -> 'int':
return self.depth_of_node(root)

def depth_of_node(self, node : TreeNode):
dep_left, dep_right = 0, 0

if not node:
return 0

dep_left = 0 if not node.left else self.depth_of_node(node.left)
dep_right =0 if not node.right else self.depth_of_node(node.right)

depth = max(dep_left, dep_right) + 1

return depth
2019-02-10 15:19

虎虎
Golang max depth
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func maxDepth(root *TreeNode) int {

if root == nil {
return 0
}

if root.Left == nil && root.Right == nil {
return 1
}

return int(math.Max(float64(maxDepth(root.Left)), float64(maxDepth(root.Right)))) + 1

}
2019-02-09 22:38

黄丹

王争老师新年的第五天快乐！
放上今天LeetCode四题的代码和思路
解题思路：对于树，这个结构很特殊，树是由根节点，根节点的左子树，根节点的右子树组成的，定义的时候就是一个递归的定义。因此在解决与树相关的问题的时候，经常会用到递归。今天的四题都不例外。
翻转二叉树：就是递归的让节点的左子树指向右子树，右子树指向左子树。
二叉树的最大深度：当前深度=1+Max(左子树深度，右子树深度)，递归的结束条件为节点为null，或者是一个叶节点。
验证二叉查找树：一颗树是二叉查找树必须满足：当前的节点>=左子树&&当前的节点<=右子树，左子树是二叉查找树，右子树是二叉查找树，也是递归的定义。
路径总和：遍历树的路径，看是否和为sum值（树的遍历也是递归的哦）
四道题的代码在：https://github.com/yyxd/leetcode/tree/master/src/leetcode/tree

2019-02-09 20:04

峰
path sum
public boolean hasPathSum(TreeNode root, int sum) {
if(root == null){
return false;
}

int remainSum = sum - root.val;

if(root.left == null && root.right == null){
if(remainSum == 0) return true;
}

return hasPathSum(root.left,remainSum) || hasPathSum(root.right,remainSum);

}
2019-02-09 18:31

molybdenum
老师新年好~今天我会把所有作业都补齐的
https://blog.csdn.net/github_38313296/article/details/86817926
2019-02-09 16:06

ext4
二叉树最大深度
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode(int x) : val(x), left(NULL), right(NULL) {}
* };
*/
class Solution {
public:
int maxDepth(TreeNode* root) {
if (root == NULL) {
return 0;
}

```
int leftDepth = maxDepth(root -> left);
int rightDepth = maxDepth(root -> right);
return 1 + (leftDepth > rightDepth ? leftDepth : rightDepth);
}
};
```

2019-02-09 14:41

_CountingStars

二叉树的最大深度 go 语言实现

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func maxDepth(root *TreeNode) int {
if root == nil {
return 0
}

leftDepth :=0
rightDepth :=0
if root.Left != nil {
leftDepth = maxDepth(root.Left)
}

if root.Right != nil {
rightDepth = maxDepth(root.Right)
}

if leftDepth >= rightDepth {
return leftDepth + 1
} else {
return rightDepth + 1
}
}
```

2019-02-09 13:10

_CountingStars

翻转二叉树 go 语言实现

```
/**
* Definition for a binary tree node.
* type TreeNode struct {
* Val int
* Left *TreeNode
* Right *TreeNode
* }
*/
func invertTree(root *TreeNode) *TreeNode {
if root == nil {
return nil
```

```
}

if root.Left != nil {
root.Left = invertTree(root.Left)
}

if root.Right != nil {
root.Right = invertTree(root.Right)
}

root.Left, root.Right = root.Right, root.Left

return root
}
```

2019-02-09 12:55

C_love

Path Sum

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode(int x) { val = x; }
* }
* Time and space complexity: O(n)
*/
class Solution {
public boolean hasPathSum(TreeNode root, int sum) {
if (root == null) {
return false;
}
if (root.left == null && root.right == null) {
return sum - root.val == 0;
}
return hasPathSum(root.left, sum - root.val) || hasPathSum(root.right, sum - root.val);
}
}
```

2019-02-09 09:41

失火的夏天

```
// 翻转二叉树
public TreeNode invertTree(TreeNode root) {
if(root == null){
return root;
}
TreeNode node = root;
Queue<TreeNode> queue = new LinkedList<>();
queue.add(node);
while(!queue.isEmpty()){
node = queue.poll();
```

```
TreeNode tempNode = node.left;
node.left = node.right;
node.right = tempNode;
if(node.left != null){
queue.offer(node.left);
}
if(node.right != null){
queue.offer(node.right);
}
}
return root;
}
// 二叉树的最大深度
public int maxDepth(TreeNode root) {
if(root == null) return 0;
return Math.max(maxDepth(root.left), maxDepth(root.right))+1;
}
// 验证二叉查找树
public boolean isValidBST(TreeNode root) {
if (root == null) {
return true;
}
Stack<TreeNode> stack = new Stack<>();
TreeNode node = root;
TreeNode preNode = null;
while(node != null || !stack.isEmpty()){
stack.push(node);
node = node.left;
while(node == null && !stack.isEmpty()){
node = stack.pop();
if(preNode != null){
if(preNode.val >= node.val){
return false;
}
}
preNode = node;
node = node.right;
}
}
return true;
}
// 路径总和
public boolean hasPathSum(TreeNode root, int sum) {
if (root == null) {
return false;
}
return hasPathSum(root, root.val, sum);
}

public boolean hasPathSum(TreeNode root, int tmp, int sum) {
if (root == null) {
```

```
return false;
}
if (root.left == null && root.right == null) {
return tmp == sum;
}
if (root.left == null) {
return hasPathSum(root.right, root.right.val + tmp, sum);
}
if (root.right == null) {
return hasPathSum(root.left, root.left.val + tmp, sum);
}
return hasPathSum(root.left, root.left.val + tmp, sum) ||
hasPathSum(root.right, root.right.val + tmp, sum);
}
```

2019-02-09 00:11