二叉树的按层遍历和锯齿形遍历

* 按层遍历

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*按层遍历\*\*\*\*\*\*\*\*\*\*/

\* 1. 按层遍历，存放到一个List中；

\* 2. 按层遍历，每一层存放到一个List中，返回一个大**List<List>**；

# 从上往下从左往右遍历二叉树

## 题目描述

从上往下打印出二叉树的每个节点，同层节点从左至右打印。

\* 题目要求:按层遍历，存到一个List中；

\* 题目描述：从上往下按层打印出二叉树的每个节点，同层节点从左至右打印。

\*

## 思路分析

思路介绍：借助**双端队列Deque**即可。

\*首先将根节点放入队列中，终止条件队列为空；

\*从队列头部取节点，同时将该节点的左、右子节点分别从队列后端添加进去。

## Java代码

public ArrayList<Integer> traversalByLevel(TreeNode root) { //PrintFromTopToBottom

ArrayList<Integer> list = new ArrayList<Integer>();

if(root == null) return list;

**ArrayDeque<TreeNode> deque = new ArrayDeque<>();**

**deque.addLast(root);**

while(!deque.isEmpty()){//队列为空表示结束

TreeNode pollNode = deque.pollFirst();

list.add(pollNode.val);

**if(pollNode.left!=null) deque.addLast(pollNode.left);**

**if(pollNode.right!=null) deque.addLast(pollNode.right);**

}

return list;

}

# 按层遍历每一层—剑指Offer60\_把二叉树打印出多行

## 题目介绍

\* 按层遍历：每一层放到一个List中，返回List<List>

从上到下按层打印二叉树，同一层结点从左至右输出。每一层输出一行。

## 思路分析

\* 思路分析：利用双端队列Deque解决。

\*增加两个统计数，一个是currentLevelCount一个是nextLevelCount；

\*用于标识当前层的范围，其他思路与按层遍历整个二叉树类似。

## Java代码

public ArrayList<ArrayList<Integer>> traversalByEveryLevel(TreeNode root){

ArrayList<ArrayList<Integer>> lists = new ArrayList<ArrayList<Integer>>();

if(root == null) return lists;

ArrayList<Integer> list = new ArrayList<Integer>();

int currentLevelCount = 1,nextLevelCount = 0;

ArrayDeque<TreeNode> deque = new ArrayDeque<TreeNode>();

deque.addLast(root);

while(!deque.isEmpty()){//一次循环只能取出一个元素

TreeNode pollNode = deque.pollFirst();

currentLevelCount--;

list.add(pollNode.val);

if(pollNode.left!= null){

deque.addLast(pollNode.left);

nextLevelCount++;

}

if(pollNode.right!= null){

deque.addLast(pollNode.right);

nextLevelCount++;

}

if(currentLevelCount == 0){//当前层没有了

lists.add(list);

if(!deque.isEmpty()){

list = new ArrayList<Integer>();

currentLevelCount = nextLevelCount;

nextLevelCount = 0;

}

}

}

return lists;

}

# 从下往上遍历二叉树

## 题目介绍

\* 题目：LeetCode\_107\_BinaryTreeLevelOrderTraversal\_II

\* 改进：从下往上按层遍历二叉树；首先打印最后一层，最后打印第一层。也就是要求List<List>的第一子List为最后一层。

## 思路分析

\* 递归方法解决：首先获取二叉树的深度，初始化lists结果集合；

\* 然后在递归的时候传入当前层的层数，利用层数标识往lists的第几个list中添加元素；

\* 按照前序遍历的顺序递归，递归终止条件是node==null。

## Java代码

public List<List<Integer>> **binaryTreeLevelOrderTraversal**\_II(TreeNode root){

int maxDepth = getMaxDepth(root);//获取二叉树深度

List<List<Integer>> lists = new ArrayList<List<Integer>>();

for(int i = 0;i < maxDepth;i++) lists.add(new ArrayList<Integer>());//初始化lists

perLevelTraversal(root,1,lists,maxDepth);

return lists;

}

//递归方法

public void perLevelTraversal(TreeNode root,int level,List<List<Integer>> lists,int depth){

if(root == null) return;

lists.get(depth-level).add(root.val);

perLevelTraversal(root.left,level+1,lists,depth);

perLevelTraversal(root.right,level+1,lists,depth);

}

public int getMaxDepth(TreeNode root){

//获取二叉树深度

if(root == null) return 0;

return Math.max(getMaxDepth(root.left),getMaxDepth(root.right))+1;

}

# 按照锯齿形或“之”字形遍历二叉树—剑指Offer59\_按之字形顺序打印二叉树

## 题目介绍

\*按照锯齿形或之字形遍历打印二叉树

题目描述

请实现一个函数按照**之**字形打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右至左的顺序打印，第三行按照从左到右的顺序打印，其他行以此类推。

## 思路分析

\* 思路分析：利用两个栈，一个存储当前层的节点引用，另一个存储下一层的节点引用。

\* 奇数层时先存左节点，再存右节点；偶数层时先存右节点，再存左节点。

\*因此设置一个flag作为0或1的标识；0与1交换：flag=(flag+1)&1;

## Java代码

public ArrayList<ArrayList<Integer>> traversalByZigZagLevel(TreeNode root){//锯齿形

ArrayList<ArrayList<Integer>> lists = new ArrayList<ArrayList<Integer>>();

if(root == null) return lists;

ArrayDeque<TreeNode>[] stacks = new ArrayDeque[2];

stacks[0] = new ArrayDeque<TreeNode>();

stacks[1] = new ArrayDeque<TreeNode>();

ArrayList<Integer> list = new ArrayList<Integer>();

int flag = 1;

stacks[flag].push(root);

// while(!stacks[0].isEmpty()||!stacks[1].isEmpty()){//都空了才结束；只要有一个没空就继续

**while(!stacks[flag].isEmpty()){//**只要当前要取元素的栈非空即可，若空了在循环体结束部分更换flag

TreeNode popNode = stacks[flag].pop();

list.add(popNode.val);

if(flag == 1) {

if (popNode.left != null) stacks[0].push(popNode.left);

if (popNode.right != null) stacks[0].push(popNode.right);

}else{

if (popNode.right != null) stacks[1].push(popNode.right);

if (popNode.left != null) stacks[1].push(popNode.left);

}

**if(stacks[flag].isEmpty()){**

**flag = (flag+1)&1;//0变成1，1变成0**

**lists.add(list);**

**if(!stacks[flag].isEmpty()) list = new ArrayList<Integer>();**

**}**

}

return lists;

}