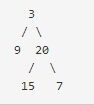
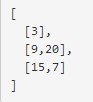
Binary Tree Level Order Traversal

# 102Binary Tree Level Order Traversal 二叉树的层遍历

Given **a binary tree**, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example: Given binary tree [3,9,20,null,null,15,7],

return its level order traversal as:

## 算法1:递归方法。(必须掌握)

思路：在先序遍历的递归算法的基础上，增加一个level的变量，标识当前的层数，然后将节点值存到List中。层数对应List中的索引。

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

public List<List<Integer>> levelOrder(TreeNode root) {

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

if(root == null) return lists;

int level = 0;//表示当前层

**helperRecur(root,level,lists);**

return lists;

}

//递归方法

private void helperRecur(TreeNode root,int level,List<List<Integer>> lists){

if(root == null) return ;

if(lists.size() < level +1)

lists.add(new ArrayList<Integer>());

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

**helperRecur(root.left,level+1,lists);**

**helperRecur(root.right,level+1,lists);**

}

}

## 算法2：比较容易理解

思路：一层一层的往下走，利用List<TreeNode>存储当前层的所有节点，然后获取当前层所有节点的值存到List<List<Integer>>中，再获取下一层的所有的节点…，依次类推，直到下一层的不存在节点为止。

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

public List<List<Integer>> levelOrder(TreeNode root) {

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

if(root == null) return lists;

List<TreeNode> rootList = new ArrayList<TreeNode>();

rootList.add(root);

while(rootList.size() != 0){//下一层没有节点了，就跳出循环

**lists.add(getListVals(rootList));//获取当前层的所有值**

**rootList = getListNodes(rootList);//**获取当前层的所有子节点

}

return lists;

}

//获取List中节点的所有的值

private ArrayList<Integer> getListVals(List<TreeNode> nodes){

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

for(TreeNode node : nodes){

vals.add(node.val);

}

return vals;

}

//获取List中节点的所有的子节点

private ArrayList<TreeNode> getListNodes(List<TreeNode> nodes){

ArrayList<TreeNode> childrenNodes = new ArrayList<TreeNode>();

for(TreeNode node : nodes){

if(node.left != null) childrenNodes.add(node.left);

if(node.right != null) childrenNodes.add(node.right);

}

return childrenNodes;

}

\*/

}

## 算法3：与算法2类似，原理相同。

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

public List<List<Integer>> levelOrder(TreeNode root) {

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

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

if(root == null)

return al;

**LinkedList<TreeNode> current = new LinkedList<TreeNode>();**

**LinkedList<TreeNode> next = new LinkedList<TreeNode>();**

current.add(root);

while(!current.isEmpty()){

TreeNode node = current.remove();

if(node.left != null)

next.add(node.left);

if(node.right != null)

next.add(node.right);

nodeValues.add(node.val);

if(current.isEmpty()){

current = next;

**next = new LinkedList<TreeNode>();**

al.add(nodeValues);

nodeValues = new ArrayList();

}

}

return al;

}

}

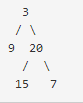
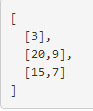
# 二叉树锯齿形层级遍历Binary Tree Zigzag Level Order Traversal

103. Binary Tree Zigzag Level Order Traversal

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example:

Given binary tree [3,9,20,null,null,15,7],

 return its **zigzag** level order traversal as: 

**算法：递归方法实现。（必须掌握）**在普通层遍历基础上添加层的奇偶判断。

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

public List<List<Integer>> zigzagLevelOrder(TreeNode root) {

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

if(lists == null) return lists;

int level = 0;

helperRecur(root,level,lists);

return lists;

}

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

if(root == null) return;

if(lists.size() < level +1){

lists.add(new LinkedList<Integer>());

}

**if(level%2 == 0){ //仅仅这里需要改变**

**lists.get(level).add(root.val);**

**}else{**

**// ((LinkedList)lists.get(level)).addFirst(root.val);**

**lists.get(level).add(0,root.val);**

**}**

helperRecur(root.left,level+1,lists);

helperRecur(root.right,level+1,lists);

}

}

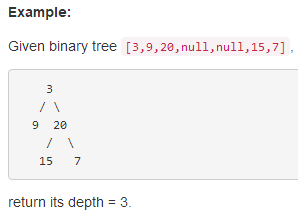
经过测试：发现若使用LinkedList的话，性能比较差，改成ArrayList就是对优秀的。

# 104. Maximum Depth of Binary Tree求二叉树的最大深度

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Note: A leaf is a node with no children.



算法：递归方法。**只需要两行代码**。

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

public int maxDepth(TreeNode root) {

**if(root == null) return 0;**

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

}

}

# 107 Binary Tree Level Order Traversal II二叉树层遍历II

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

For example: Given binary tree [3,9,20,null,null,15,7],

 return its bottom-up level order traversal as: 

算法：

思路：整体与1的层遍历相同。区别是：首先求出二叉树的最大深度，然后将二叉树的第1层当做depth-1层，第2层作为depth-11层，第depth层作为0层即可。

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

class Solution {

/\*

public List<List<Integer>> levelOrderBottom(TreeNode root) {

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

if(root == null) return lists;

**int depth = maxDepth(root);//get the max depth**

int level = 0;

for(int i = 0;i < depth;i++)

lists.add(new ArrayList<Integer>());

helperRecur(root,lists,level,depth);

return lists;

}

private void helperRecur(TreeNode root,List<List<Integer>> lists,int level,int depth){

if(root == null) return;

lists.get(depth - 1 - level).add(root.val);

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

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

}

//get max depth

private int maxDepth(TreeNode root){

if(root == null) return 0;

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

}\*/

//上面的解决方法已经很不错，再优化一点点：上面需要depth-1-level，可以直接认为把来level逆过来，从depth-1递减

public List<List<Integer>> levelOrderBottom(TreeNode root) {

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

if(root == null) return lists;

int depth = maxDepth(root);//get the max depth

int level = 0;

for(int i = 0;i < depth;i++)

lists.add(new ArrayList<Integer>());

helperRecur(root,lists,depth-1);

return lists;

}

private void helperRecur(TreeNode root,List<List<Integer>> lists,int level){

if(root == null) return;

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

helperRecur(root.left,lists,level-1);

helperRecur(root.right,lists,level-1);

}

//get max depth

private int maxDepth(TreeNode root){

if(root == null) return 0;

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

}

}