题目 把二叉树打印成多行

考点 树 热点指数 31052 通过率 29.85%

具体题目

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。

```
剑指Offer-把二叉树打印成多行 package Tree;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
/**
 * 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：
 * 按层次输出二叉树
 * 访问根节点，并将根节点入队。
 * 当队列不空的时候，重复以下操作。
 * 1、弹出一个元素。作为当前的根节点。
 * 2、如果根节点有左孩子，访问左孩子，并将左孩子入队。
 * 3、如果根节点有右孩子，访问右孩子，并将右孩子入队。
 */
public class Solution8 {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9};
        Solution8 solution8 = new Solution8();
        TreeNode treeNode = solution8.createBinaryTreeByArray(array, 0);
        for (ArrayList list :
                solution8.Print(treeNode)) {
            System.out.println(list);
        }
    }
    /**
     * 层次遍历
     *
     * [@param pRoot 根节点
     * @return](/profile/547241) arrayLists
     */
    <ArrayList> Print(TreeNode pRoot) {
        //存放结果
        ArrayList> arrayLists = new ArrayList();
        if (pRoot == null) {
            return arrayLists;
        }
        //使用队列，先进先出
        Queue queue = new LinkedList();
        //存放每行的列表
        ArrayList arrayList = new ArrayList();
        //记录本层打印了多少个
        int start = 0;
        //记录下层打几个
        int end = 1;
        queue.add(pRoot);
        while (!queue.isEmpty()) {
            TreeNode temp = queue.remove();
            //添加到本行的arrayList
            arrayList.add(temp.val);
            start++;
            //每打印一个节点，就把此节点的下一层的左右节点加入队列，并记录下一层要打印的个数
```

```
            if (temp.left != null) {
                queue.add(temp.left);
            }
            if (temp.right != null) {
                queue.add(temp.right);
            }
            //判断本层打印是否完成
            if (start == end) {
                //此时的queue中存储的都是下一层的节点，则end即为queue大小
                end = queue.size();
                start = 0;
                //把arrayList添加到结果列表arrayLists中
                arrayLists.add(arrayList);
                //重置arrayList
                arrayList = new ArrayList();
            }
        }
        return arrayLists;
    }
    private TreeNode createBinaryTreeByArray(int[] array, int index) {
        TreeNode tn = null;
        if (index < array.length) {
            int value = array[index];
            tn = new TreeNode(value);
            tn.left = createBinaryTreeByArray(array, 2 * index + 1);
            tn.right = createBinaryTreeByArray(array, 2 * index + 2);
            return tn;
        }
        return tn;
    }
    public class TreeNode {
        int val = 0;
        TreeNode left = null;
        TreeNode right = null;
        public TreeNode(int val) {
            this.val = val;
        }
    }
}
```

```
/*
 *按行打印二叉树
 *方法1：
 * 借助两个队列按层扫描
 *
 * 方法2：
 * 其实可以用一个队列就能做，扫描完一层插入一个null
 * 然后下层扫描的时候扫描到null再插一个null
 */

    ArrayList<ArrayList<Integer> > Print(TreeNode pRoot)
    {
        ArrayList<ArrayList<Integer>> resutltList = new ArrayList<>();

        if(pRoot == null)
        {
            return resutltList;
        }
```

```
        ArrayList<TreeNode> treeQueue = new ArrayList<>();

        treeQueue.add(pRoot);
        treeQueue.add(null);

        while(treeQueue.get(0) != null)
        {//树没有扫描完
            ArrayList<Integer> currentList = new ArrayList<>();

            TreeNode currentNode = treeQueue.get(0);//获取队首元素

            while(currentNode != null)
            {
                currentList.add(currentNode.val);//将当前节点进入链

                if(currentNode.left != null)
                {
                    treeQueue.add(currentNode.left);
                }

                if(currentNode.right != null)
                {
                    treeQueue.add(currentNode.right);
                }

                treeQueue.remove(0); //队首元素出队列
                currentNode = treeQueue.get(0);//继续获取队首元素
            }

            treeQueue.remove(0);//上一层的null出队列
            treeQueue.add(null);//扫描完一层加入一个null代表当前层的结束标志
            resutltList.add(currentList); //插入当前行扫描的所有元素

        }

        return resutltList;
    }
```

```
import java.util.ArrayList;
import java.util.LinkedList; /*
public class TreeNode {
    int val = 0;
    TreeNode left = null;
    TreeNode right = null;     public TreeNode(int val) {
        this.val = val;     } }
*/
//now表示当前行剩余节点个数，next表示下一行节点总数，array存放每行的节点，当now==0,表示本行全部在array中，则加入结果集
public class Solution {
    ArrayList<ArrayList<Integer> > Print(TreeNode pRoot) {
     ArrayList<ArrayList<Integer> > result =new ArrayList<ArrayList<Integer> >();
        ArrayList<Integer> array=new ArrayList<Integer>();
        LinkedList<TreeNode> list=new LinkedList<TreeNode>();
        if(pRoot==null){
            return result;
        }
        int now=1;
        int next=0;
        list.add(pRoot);
        while(!list.isEmpty()){
```

```
            TreeNode p=list.remove();
            now--;
            array.add(p.val);
            if(p.left!=null){
                list.add(p.left);
                next++;
            }
            if(p.right!=null){
                list.add(p.right);
                next++;
            }
            if(now==0){
                result.add(new ArrayList<Integer>(array));
                array.clear();
                now=next;
                next=0;
            }
        }
        return result;
    }

}
```

```
import java.util.*;
public class Solution {
    ArrayList<ArrayList<Integer>> Print(TreeNode pRoot) {
        if (pRoot == null) {
            return new ArrayList<ArrayList<Integer>>();
        }
        ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();
        LinkedList<TreeNode> list = new LinkedList<>();
        LinkedList<TreeNode> helpList = new LinkedList<>();;
        list.add(pRoot);
        while (!list.isEmpty()) {
            if(helpList.isEmpty()) {
                while (!list.isEmpty()) {
                    helpList.add(list.poll());
                }
            }

            // 跳出上面的循环表示list已经空了
            ArrayList<Integer> tmp = new ArrayList<>();
            while (!helpList.isEmpty()) {
                TreeNode cur = helpList.poll();
                tmp.add(cur.val);
                if(cur.left != null) {
                    list.add(cur.left);
                }
                if(cur.right != null) {
                    list.add(cur.right);
                }
            }
            res.add(tmp);
        }
        return res;
    }
}
```