题目 按之字形顺序打印二叉树

考点 树 热点指数 29055 通过率 23.42%

具体题目

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。

剑指offer的思路，按之字型顺序打印二叉树需要两个栈，在打印某一行节点时，把下一层的子节点保存到相应的栈里，如果当前打印的是奇数层（第一层，第三层等），则先保存左子树结点，再保存右子树节点得到第一个栈里，如果打印的是偶数层，则先保存右子树节点再保存左子树节点到第二个栈里。

```
     * 题目：Z字形打印二叉树
     *
     * 思路:
     * 借助两个栈stack1,stack2.
     * 先让首节点接入stack1，然后奇数行时stack1中节点的孩子出队列加入stack2（按先左孩子再右孩子的顺序），并加入链中
     * 偶数行时出stack2中节点的孩子加入stack1(按先右孩子后左孩子的顺序)，并加入链
     */
    public ArrayList<ArrayList<Integer> > Print(TreeNode pRoot)
    {
        ArrayList<ArrayList<Integer>> zTreeList = new ArrayList<>();

        if(pRoot == null)
        {
            return zTreeList;
        }

        Stack<TreeNode> oddStack = new Stack<>();//奇数行栈
        Stack<TreeNode> evenStack = new Stack<>();//偶数行栈

        boolean isOdd = true;

        oddStack.add(pRoot);

        while(!(oddStack.isEmpty() && evenStack.isEmpty()))
        {//树没有遍历完
            ArrayList<Integer> currentList = new ArrayList<>();

            if(isOdd == true)
            {//奇数行，stack1中节点的孩子节点按先左孩子后右孩子的顺序入栈2

                while(!oddStack.isEmpty())
                {
                    TreeNode currentNode = oddStack.peek(); //取队栈顶元素

                    currentList.add(currentNode.val); //添加当前列表

                    if(currentNode.left != null)
                    {
                        evenStack.push(currentNode.left);
                    }

                    if(currentNode.right != null)
                    {
                        evenStack.push(currentNode.right);
```

```
                }

                oddStack.pop(); //栈顶元素出栈

            }

            zTreeList.add(currentList);//加入当前行

            isOdd = false; //更新下一层扫描树为偶数行
        }
        else
        {//偶数行,stack2中元素节点的孩子按先右孩子孩子后左孩子顺序入stack1
            while(!evenStack.isEmpty())
            {
                TreeNode currentNode = evenStack.peek(); //获取栈顶元素

                currentList.add(currentNode.val);

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

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

                evenStack.pop();
            }

            zTreeList.add(currentList);

            isOdd = true;
        }
    }

    return zTreeList;
}
```

用两个栈实现，栈s1与栈s2交替入栈出栈。reverse方法时间复杂度比较高，两个栈以空间换时间

```
public class Solution {
    public ArrayList<ArrayList<Integer> > Print(TreeNode pRoot) {
        ArrayList<ArrayList<Integer> > listAll = new ArrayList<>();
        if(pRoot==null)return listAll;
        Stack<TreeNode> s1 = new Stack<>();
        Stack<TreeNode> s2 = new Stack<>();
        int level = 1;
        s1.push(pRoot);
        while(!s1.isEmpty()||!s2.isEmpty()){
            ArrayList<Integer> list = new ArrayList<>();
            if(level++%2!=0){
                while(!s1.isEmpty()){
                    TreeNode node = s1.pop();
                    list.add(node.val);
                    if(node.left!=null)s2.push(node.left);
                    if(node.right!=null)s2.push(node.right);
                }
            }
```

```
            else{
                while(!s2.isEmpty()){
                    TreeNode node = s2.pop();
                    list.add(node.val);
                    if(node.right!=null)s1.push(node.right);
                    if(node.left!=null)s1.push(node.left);
                }
            }
            listAll.add(list);
        }
        return listAll;
    }
}
```