

```

1 // 之字形打印二叉树
2 /*
3 struct TreeNode {
4     int val;
5     struct TreeNode *left;
6     struct TreeNode *right;
7     TreeNode(int x) :
8         val(x), left(NULL), right(NULL) {
9     }
10 };
11 */
12 class Solution {
13 public:
14     vector<vector<int>> > Print(TreeNode* pRoot) {
15
16         // 存储结果
17         vector<vector<int>> > results;
18         vector<int> rlt_temp;
19
20         // 边界条件
21         if(pRoot == nullptr)
22             return results;
23
24         // 辅助容器
25         stack<TreeNode*> stk[2]; // stk[0]是奇数层, stk[1]是偶数层
26         int now = 0;
27         int next = 1;
28         TreeNode* temp=pRoot;
29
30         // 根节点入栈
31         stk[now].push(temp);
32
33         // 遍历两个栈, 当两个栈均为空时, 跳出循环
34         while(!stk[now].empty() || !stk[next].empty()){
35             // 存储遍历结果
36             temp = stk[now].top();
37             rlt_temp.push_back(temp->val);
38             stk[now].pop();
39
40             // 当前层是奇数或偶数
41             if(now==0)
42             {
43                 // 当前层是奇数时, 左子树先入栈, 右子树后入栈
44                 if(temp->left!=nullptr)
45                     stk[next].push(temp->left);
46                 if(temp->right!=nullptr)
47                     stk[next].push(temp->right);
48             }
49             else
50             {
51                 // 当前层是偶数时, 右子树先入栈, 左子树后入栈
52                 if(temp->right!=nullptr)
53                     stk[next].push(temp->right);
54                 if(temp->left!=nullptr)
55                     stk[next].push(temp->left);
56             }
57
58             // 当前层为空时, 打印下一层
59             if(stk[now].empty())
60             {
61                 results.push_back(rlt_temp);
62                 rlt_temp.clear();
63                 now=1-now;
64                 next = 1-next;
65             }
66         }
67         return results;
68     }
69 };
70

```