题目一 不分行

面试题 32：从上到下打印二叉树

题目一：不分行从上到下打印二叉树

从上到下打印出二叉树的每个节点，同一层的节点按照从左到右的顺序打印。例如，输入图 4.6 中的二叉树，则依次打印出 8,6,10,5,7,9,11。二叉树节点的定义如下：

```
struct BinaryTreeNode
{
    int                    m_nValue;
    BinaryTreeNode*        m_pLeft;
    BinaryTreeNode*        m_pRight;
};
```

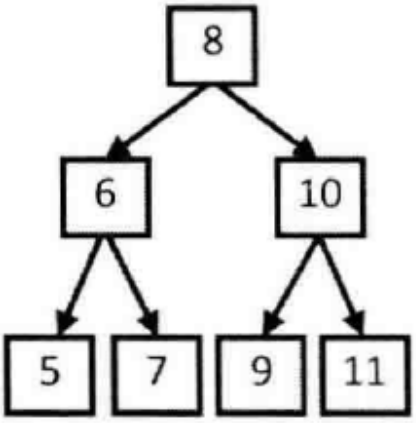


图 4.6　一棵二叉树，从上到下按层打印的顺序为 8,6,10,5,7,9,11

解

层序遍历，队列

```
vector<int> printFromToptoBottom(TreeNode* root)
{
    vector<int> res;
    if(!root)
        return res;
    queue<TreeNode*> q;
    q.push(root);
    while(!q.empty())
    {
        TreeNode* t=q.front();
```

```
        q.pop();
        res.push_back(t->val);
        if(t->left)
            q.push(t->left);
        if(t->right)
            q.push(t->right);
    }
    return res;
}
```

题目二

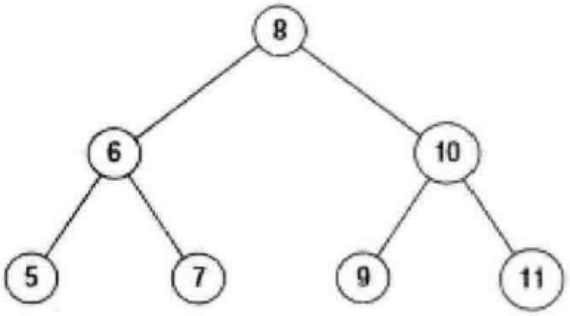
题目二：分行从上到下打印二叉树。

从上到下按层打印二叉树，同一层的节点按从左到右的顺序打印，每一层打印到一行。例如，打印图 4.7 中二叉树的结果为：

8

6 10

5 7 9 11



解

层序遍历，队列，不过要记录每一层的节点个数

```
vector<vector<int>> Print(TreeNode* root)
{
    vector<vector<int>> res;
    if(!root)
        return res;
    queue<TreeNode*> q;
    q.push(root);
    while(!q.empty())
    {
        int n=q.size();
        vector<int> tmp;
```

```
        while(n--)
        {
            TreeNode* t=q.front();
            q.pop();
            tmp.push_back(t->val);
            if(t->left)
                q.push(t->left);
            if(t->right)
                q.push(t->right);
        }
        res.push_back(tmp);
    }
    return res;
}
```

题目三 之字形打印二叉树

题目三：之字形打印二叉树。

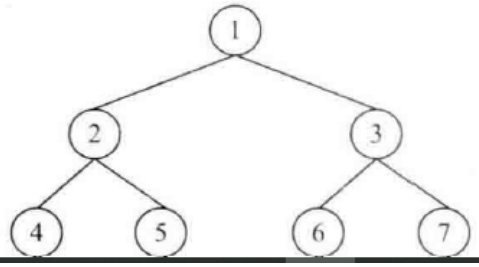
请实现一个函数按照之字形顺序打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右到左的顺序打印，第三行再按照从左到右的顺序打印，其他行以此类推。例如，按之字形顺序打印图 4.8 中二叉树的结果为：

1

3 2

4 5 6 7

15 14 13 12 11 10 9 8



解

两个栈，

```
vector<vector<int>> zigzagPrint(TreeNode* root)
{
    vector<vector<int>> res;
```

```
vector<int> tmp;
stack<TreeNode*> s1,s2;
s1.push(root);
tmp.push_back(root->val);
res.push_back(tmp);
tmp.clear();
TreeNode* node;
while(!s1.empty() || !s2.empty())
{
    while(!s1.empty())
    {
        node=s1.top();
        s1.pop();
        if(node->right)
        {
            s2.push(node->right);
            tmp.push_back(node->right->val);
        }
        if(node->left)
        {
            s2.push(node->left);
            tmp.push_back(node->left->val);
        }
    }
    if(!tmp.empty())
    {
        res.push_back(tmp);
        tmp.clear();
    }
    while(!s2.empty())
    {
        node=s2.top();
        s2.pop();
        if(node->left)
        {
            s1.push(node->left);
            tmp.push_back(node->left->val);
        }
        if(ndoe->right)
        {
            s1.push(node->right);
            tmp.push_back(node->right->val);
        }
    }
    if(!tmp.empty())
    {
        res.push_back(res);
```

```
            tmp.clear();
        }
    }
    return res;
}
```