从上到下打印二叉树 III

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:

给定二叉树: [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

返回其层次遍历结果：

```
[
  [3],
  [20,9],
  [15,7]
]
```

解题思路：

利用 deque 这个 STL 数据容器，维护一个双端队列。如果是偶数层，我们先 pop_back 底层的元素，然后从右往左（从头向尾输出元素）地向顶部 push_front 下一层的元素；如果是奇数层，我们先 pop_front 顶层的元素，然后从左往右（从尾向头输出元素）地向底部 push_back 下一层的元素。

我们这里一共用到了 deque 的这几个函数：

奇数层是：front(), pop_front(), push_back()

偶数层是：back(), pop_back(), push_front()

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
    class Solution {
public:
```

```
    vector<vector<int>> levelOrder(TreeNode* root) {
        if(!root) return {};
        vector<vector<int> > res;
        vector<int> temp;
        deque<TreeNode *> store; // 使用 deque
        store.push_back(root);
        TreeNode *curr;
        bool odd = true; // odd 用来判断是否是奇数层，一开始是第一层，所以 odd 是 true
        // remaining 来记录当前层剩余的元素个数；nextLevel 来记录下一层的元素个数；row 记录行数
        int remaining = 1, nextLevel = 0; // 这两个变量与方法一意义相同
        while(!store.empty())
        {
            if(odd) // 奇数层
            {
                curr = store.front();
                temp.push_back(curr -> val);
                store.pop_front(); // 弹出头部元素
                remaining --;
                if(curr -> left)
                {
                    store.push_back(curr -> left); // 从左往右向底部添加下一层的元素
                    nextLevel ++;
                }
                if(curr -> right)
                {
                    store.push_back(curr -> right); // 从左往右向底部添加下一层的元素
                    nextLevel ++;
                }
            }
            else // 偶数层
            {
                curr = store.back();
                temp.push_back(curr -> val);
                store.pop_back(); // 弹出尾部元素
                remaining --;
                if(curr -> right)
                {
                    store.push_front(curr -> right); // 从右往左向顶部添加下一层的元素
                    nextLevel ++;
```

```
                }
                if(curr -> left)
                {
                    store.push_front(curr -> left); // 从右往左向顶部添加下一
层的元素
                    nextLevel ++;
                }
            }
            if(remaining == 0)//逻辑控制
            {
                res.push_back(temp);
                temp.clear();//清空维护下一层元素
                remaining = nextLevel;//根据上一层遍历左右子树的结果找到下一层
节点个数
                nextLevel = 0;//置零维护下一层
                odd = !odd; // 当前层已处理完，正式进入下一层，odd 要变成 !odd
            }
        }
        return res;
    }
};
```