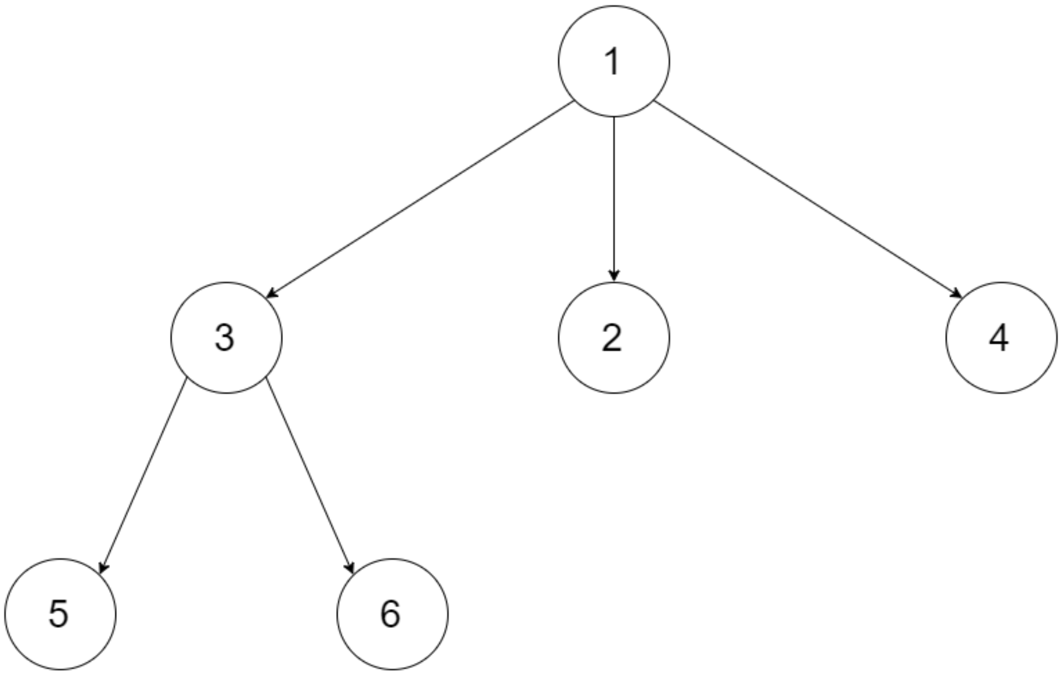
429. N-ary Tree Level Order Traversal

- https://leetcode-cn.com/problems/n-ary-tree-level-order-traversal/

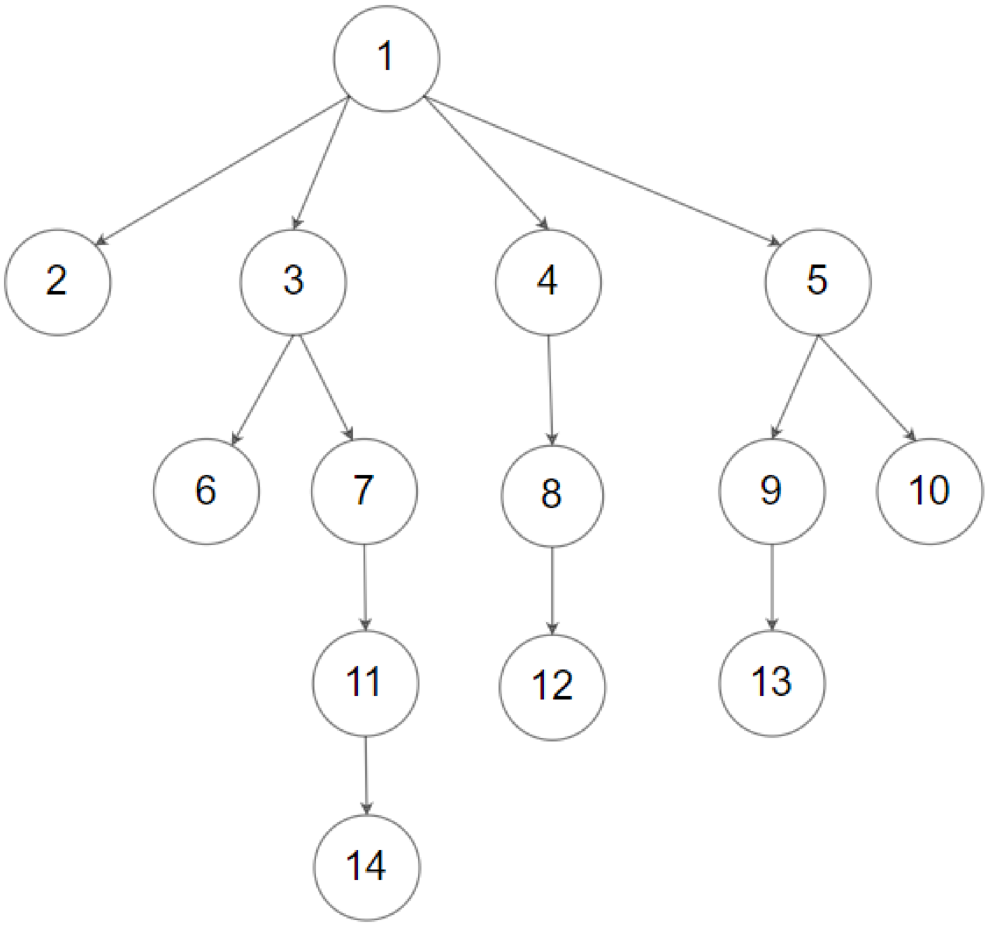
Given an n-ary tree, return the *level order* traversal of its nodes' values.
Nary-Tree input serialization is represented in their level order traversal, each group of children is separated by the null value (See examples).

Example 1:



```
Input: root = [1,null,3,2,4,null,5,6]
Output: [[1],[3,2,4],[5,6]]
```

Example 2:



```
Input: root = [1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null,null,14]
Output: [[1],[2,3,4,5],[6,7,8,9,10],[11,12,13],[14]]
```

Constraints:

- The height of the n-ary tree is less than or equal to `1000`
- The total number of nodes is between `[0, 10^4]`

Solution

1.利用队列实现广度优先搜索 O(n) – 推荐

- 利用队列作为临时容器，利用其先进先出的特性，在队列取出某节点是，将其子节点放入队列中，这实际上就是树的广度优先遍历方法(DFS)。
- 那如何将每一层的节点保存为一个列表呢，则可以在每次从队列中取数据前，读取当前队列的元素个数，这个个数就是同一层节点的个数，然后利用这个个数，来决定取节点和放入子节点的循环次数。这样就可以保存同一层节点的值到一个列表中，在这个内循环结束之后，将列表保存到返回的大列表当中

```
//using queue and node count of each level
class Solution {
public:
    vector<vector<int>> levelOrder(Node* root) {
        if(root == nullptr) { return {}; }
        vector<vector<int>> res;
        queue<Node*> container;
        container.push(root);
        while(!container.empty()) {
            vector<int> levelVal;
            int levelCount = (int)container.size();
            for(int i = 0; i < levelCount; i++) {
                Node* node = container.front();
                levelVal.push_back(node->val);
                container.pop();
                for(auto &child : node->children) {
                    container.push(child);
                }
            }
            res.push_back(levelVal);
        }
        return res;
    }
};
```

2.利用更新每一层的节点来实现层遍历 O(n) – 简洁，巧妙 – 推荐

- 利用一个全局数组来保存每一层的节点，在遍历的过程中来不断更新这个数组
- 同样是判断这个全局数组是否还有元素，因为在遍历节点的过程中一直在更新这个数组，所以也是等到所有节点都遍历完了该全局数组才会为空。
- 在这个全局数组中遍历上一层的节点时，将这些节点的子节点放入到另一个临时数组中，这一层的节点全部遍历完了之后，再将临时数组覆盖掉全局数组的节点，这样就能继续遍历下一层了。

```
//using node list and node count of each level
class Solution {
public:
    vector<vector<int>> levelOrder(Node* root) {
        if(root == nullptr) { return {}; }
        vector<vector<int>> res;
        vector<Node*> upperlever = {root};
        while (!upperlever.empty()) {
            vector<Node*> currentLevel = {};
            vector<int> levelVal = {};
            for (auto &node : upperlever) {
                levelVal.push_back(node->val);
                currentLevel.insert(currentLevel.end(), node->children.begin(), node->children.end());
            }
            res.push_back(levelVal);
            upperlever.assign(currentLevel.begin(), currentLevel.end());
        }
        return res;
    }
};
```