Description

102. Binary Tree Level Order Traversal

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

```
For example:
Given binary tree [3,9,20,null,null,15,7],
    3
   / \
  9  20
    /  \
   15   7
return its level order traversal as:
[
  [3],
  [9,20],
  [15,7]
]
```

Idea

We want to traverse the tree level by level. When traverse the first level, we want to store the information about next level. What type of data structure can achive that the first next-level-child stored first and come out first. That is a queue. So when we traverse the nodes in this level, we want to put the left/right child of current node into queue so that we can traverse later. The solution can be using a BFS from the root. Each node is traversed queued and dequeued once, so O(n) time. The extra space is O(w) of the queue, where w is how wide the tree would be, which is the longest level of the tree.

Java

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
```

```
        List<List<Integer>> res = new ArrayList<>();
        Deque<TreeNode> q = new LinkedList<>();
        q.offer(root);
        while (!q.isEmpty()) {
            int size = q.size();
            List<Integer> level = new ArrayList<>();
            while (size-- > 0) {
                TreeNode cur = q.poll();
                if (cur == null) {
                    continue;
                }
                level.add(cur.val);
                q.offer(cur.left);
                q.offer(cur.right);
            }
            if (!level.isEmpty()) {
                res.add(level);
            }
        }
        return res;
    }
}
```

C++

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> res;
        deque<TreeNode*> q;
        q.push_back(root);
        while (!q.empty()) {
            int size = q.size();
            vector<int> level;
            while (size-- > 0) {
                TreeNode* cur = q.front();
                q.pop_front();
                if (cur == NULL) {
                    continue;
                }
                level.push_back(cur->val);
                q.push_back(cur->left);
                q.push_back(cur->right);
            }
            if (!level.empty()) {
                res.push_back(level);
            }
        }
```

```
        return res;
    }
};
```

We can also do a DFS of the tree and assign a level id to each level, and we add the node.val to its corresponding level in the result. The time complexity is still O(n), but the space complexity becomes the size of the recursion call stack, so it is O(H), where H is the height of the tree.

Java

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        dfsHelper(root, 0, res);
        return res;
    }

    private void dfsHelper(TreeNode root, int level, List<List<Integer>> res) {
        if (root == null) {
            return;
        }
        if (res.size() == level) {
            res.add(new ArrayList<>());
        }

        res.get(level).add(root.val);
        dfsHelper(root.left, level + 1, res);
        dfsHelper(root.right, level + 1, res);
    }
}
```

C++

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> res;
        dfsHelper(root, 0, res);
        return res;
    }

private:
    void dfsHelper(TreeNode* root, int level, vector<vector<int>>& res) {
        if (root == NULL) {
```

```
            return;
        }
        if (res.size() == level) {
            res.push_back(vector<int>());
        }

        res[level].push_back(root->val);
        dfsHelper(root->left, level + 1, res);
        dfsHelper(root->right, level + 1, res);
    }
}
```

Summary

- Looking for the order of the nodes that has been traversed, from left to right, and in next level we want to traverse left child to right child, so we want to use FIFO data structure queue to save the next level nodes that needs to be traversed.
- Basically we can have a lot of ways to implement the tree traveral, and we find out where the operation needs to be (add root.val into result) and place it in any correct location would solve the problem.
- BFS/DFS