Day 3 Trees - BFS

ITSRUNTYM

#	Problem	Description	LeetCode Link
1	Binary Tree Level Order Traversal	Traverse tree level by level using BFS.	🔗 Link
2	Binary Tree Level Order Traversal II	Traverse tree from bottom to top using BFS.	🔗 Link
3	Average of Levels in Binary Tree	Find average value at each level using BFS.	🔗 Link
4	Minimum Depth of Binary Tree	Find shortest path from root to leaf using BFS.	🔗 Link
5	Maximum Depth of Binary Tree	Use BFS to calculate maximum depth (alternate to DFS).	🔗 Link
6	Binary Tree Right Side View	Collect rightmost node of each level using BFS.	🔗 Link
7	Populating Next Right Pointers in Each Node	Connect nodes at the same level using BFS.	🔗 Link
8	Cousins in Binary Tree	Check if two nodes are cousins using BFS.	🔗 Link
9	Symmetric Tree	Use BFS (queue of pairs) to check tree symmetry.	🔗 Link

10	Find Largest Value in Each Tree Row	BFS to find max value at each level.	🔗 Link

https://leetcode.com/problems/binary-tree-level-order-traversal/description/

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        if(root==null) return res;
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);
        while(!q.isEmpty()){
            int size = q.size();//size of curr level/ no of elements
            List<Integer> arr = new ArrayList<>();
            for(int i=0;i<size;i++){
                TreeNode node = q.poll();
                arr.add(node.val);
                if(node.left!=null) q.add(node.left);
                if(node.right!=null) q.add(node.right);
            }
            res.add(arr);
        }
        return res;
    }
}
```

https://leetcode.com/problems/find-largest-value-in-each-tree-row/description/

```
class Solution {
    public List<Integer> largestValues(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        if(root==null) return res;
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);
        while(!q.isEmpty()){
            int size = q.size();
            int max = Integer.MIN_VALUE;
```

```
            for(int i=0;i<size;i++){
                TreeNode node = q.poll();
                max = Math.max(max, node.val);
                if(node.left!=null) q.add(node.left);
                if(node.right!=null) q.add(node.right);
            }
            res.add(max);
        }
        return res;
    }
}
```

https://leetcode.com/problems/binary-tree-right-side-view/description/

```
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        if(root==null) return res;
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);
        while(!q.isEmpty()){
            int size = q.size();
            TreeNode rightMost = null;
            for(int i=0;i<size;i++){
                TreeNode node = q.poll();
                rightMost = node;
                if(node.left!=null) q.add(node.left);
                if(node.right!=null) q.add(node.right);
            }
            res.add(rightMost.val);
        }
        return res;
    }
}
```

https://leetcode.com/problems/cousins-in-binary-tree/

```
class Solution {
    public boolean isCousins(TreeNode root, int x, int y) {
        Queue<Pair<TreeNode,TreeNode>> q = new LinkedList<>();
        if(root==null) return false;
```

```
        q.add(new Pair<>(root, null));
//time O n
//spcae O n
        while(!q.isEmpty()){
            int size = q.size();
            ArrayList<TreeNode> arr = new ArrayList<>();
            for(int i=0;i<size;i++){
                Pair<TreeNode,TreeNode> currPair = q.poll();
                TreeNode node = currPair.getKey();
                TreeNode parent = currPair.getValue();
                if(node.val == x || node.val == y){
                    if(parent != null){
                        arr.add(parent);
                    }
                }
                if(node.left!=null) q.add(new Pair<>(node.left, node));
                if(node.right!=null) q.add(new Pair<>(node.right, node));
            }
            if(arr.size()==2 && arr.get(0)!=arr.get(1)) return true;
        }
        return false;
    }
}
```