**Tree Assignment 2 done by N S K K K Naga Jayanth**

https://www.interviewbit.com/problems/vertical-order-traversal-of-binary-tree/

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

class TreeNode {
   int val;
   TreeNode left;
   TreeNode right;
   TreeNode(int x) {
      val = x;
      left = null;
      right = null;
   }
}

public class Solution {
   public ArrayList<ArrayList<Integer>> verticalOrderTraversal(TreeNode A) {
      ArrayList<ArrayList<Integer>> result = new ArrayList<>();
      if (A == null) {
         return result;
      }

      HashMap<Integer, List<Integer>> columnTable = new HashMap<>();
      Queue<TreeNode> queue = new LinkedList<>();
      Queue<Integer> columnQueue = new LinkedList<>();

      queue.offer(A);
```

```
columnQueue.offer(0);

int minColumn = 0;
int maxColumn = 0;

while (!queue.isEmpty()) {
    TreeNode node = queue.poll();
    int column = columnQueue.poll();

    if (!columnTable.containsKey(column)) {
        columnTable.put(column, new ArrayList<>());
    }
    columnTable.get(column).add(node.val);

    if (node.left != null) {
        queue.offer(node.left);
        columnQueue.offer(column - 1);
        minColumn = Math.min(minColumn, column - 1);
    }

    if (node.right != null) {
        queue.offer(node.right);
        columnQueue.offer(column + 1);
        maxColumn = Math.max(maxColumn, column + 1);
    }
}

for (int i = minColumn; i <= maxColumn; i++) {
    if (columnTable.containsKey(i)) {
        result.add(new ArrayList<>(columnTable.get(i)));
    }
```

```
        }


        return result;
    }
}
```

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

```
class Solution {
    int maxLevel = 0;
    List<Integer> list = new ArrayList();
    public List<Integer> rightSideView(TreeNode root) {
        if(root == null) return list;
        rightView(root,1);
        return list;
    }
    void rightView(TreeNode root,int level){
        if(root == null) return;
        if(maxLevel < level){
            list.add(root.val);
            maxLevel = level;
        }
        rightView(root.right,level+1);
        rightView(root.left,level+1);
    }
}
```

https://practice.geeksforgeeks.org/problems/left-view-of-binary-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article

```
class Tree
{
    // Function to return list containing elements of left view of binary tree
    ArrayList<Integer> leftView(Node root)
```

```
{
    ArrayList<Integer> result = new ArrayList<>();
    if (root == null) {
        return result;
    }

    Queue<Node> queue = new LinkedList<>();
    queue.add(root);

    while (!queue.isEmpty()) {
        int size = queue.size();
        boolean isFirstNode = true;

        for (int i = 0; i < size; i++) {
            Node currentNode = queue.poll();
            if (isFirstNode) {
                result.add(currentNode.data);
                isFirstNode = false;
            }

            if (currentNode.left != null) {
                queue.add(currentNode.left);
            }
            if (currentNode.right != null) {
                queue.add(currentNode.right);
            }
        }
    }

    return result;
}
```

```
}
```

https://practice.geeksforgeeks.org/problems/top-view-of-binary-tree/1

```
class Solution {
    // Function to return a list of nodes visible from the top view
    // from left to right in Binary Tree.
    static ArrayList<Integer> topView(Node root) {
        ArrayList<Integer> result = new ArrayList<>();
        if (root == null) {
            return result;
        }

        // Create a HashMap to store nodes at each horizontal distance
        HashMap<Integer, Integer> verticalOrderMap = new HashMap<>();
        Queue<NodeWithHD> queue = new LinkedList<>();

        // Initialize the queue with the root node and its horizontal distance (HD)
        queue.add(new NodeWithHD(root, 0));

        // Perform level order traversal
        while (!queue.isEmpty()) {
            NodeWithHD current = queue.poll();
            int hd = current.hd;
            Node node = current.node;

            // If the horizontal distance is not present in the map, add it
            if (!verticalOrderMap.containsKey(hd)) {
                verticalOrderMap.put(hd, node.data);
            }

            // Enqueue left child with a horizontal distance decreased by 1
            if (node.left != null) {
```

```
            queue.add(new NodeWithHD(node.left, hd - 1));
        }

        // Enqueue right child with a horizontal distance increased by 1
        if (node.right != null) {
            queue.add(new NodeWithHD(node.right, hd + 1));
        }
    }

    // Add the values from the HashMap to the result
    for (int hd : verticalOrderMap.keySet()) {
        result.add(verticalOrderMap.get(hd));
    }

    return result;
  }

  static class NodeWithHD {
    Node node;
    int hd;

    NodeWithHD(Node node, int hd) {
        this.node = node;
        this.hd = hd;
    }
  }
}
```

https://practice.geeksforgeeks.org/problems/bottom-view-of-binary-tree/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article

```
class Solution
{
```

```
//Function to return a list containing the bottom view of the given tree.
public ArrayList<Integer> bottomView(Node root)
{
    ArrayList<Integer> result = new ArrayList<>();
    if (root == null) {
        return result;
    }

    // HashMap to store nodes at each horizontal distance.
    HashMap<Integer, Integer> horizontalDistanceMap = new HashMap<>();
    Queue<NodeWithHD> queue = new LinkedList<>();

    // Initialize the queue with the root node and its horizontal distance (HD)
    queue.add(new NodeWithHD(root, 0));

    // Perform level order traversal
    while (!queue.isEmpty()) {
        NodeWithHD current = queue.poll();
        int hd = current.hd;
        Node node = current.node;

        // Update the node's value in the HashMap for the current horizontal distance.
        horizontalDistanceMap.put(hd, node.data);

        if (node.left != null) {
            queue.add(new NodeWithHD(node.left, hd - 1));
        }

        if (node.right != null) {
            queue.add(new NodeWithHD(node.right, hd + 1));
        }
```

```
        }

        // Sort the HashMap entries based on keys
        List<Map.Entry<Integer, Integer>> entryList = new
ArrayList<>(horizontalDistanceMap.entrySet());
        Collections.sort(entryList, Map.Entry.comparingByKey());

        // Add nodes from the sorted HashMap entries to the result list
        for (Map.Entry<Integer, Integer> entry : entryList) {
            result.add(entry.getValue());
        }

        return result;
    }

    // Class to represent a node with horizontal distance
    static class NodeWithHD {
        Node node;
        int hd;

        NodeWithHD(Node node, int hd) {
            this.node = node;
            this.hd = hd;
        }
    }
}
```

https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
```

```
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode() {}
*     TreeNode(int val) { this.val = val; }
*     TreeNode(int val, TreeNode left, TreeNode right) {
*         this.val = val;
*         this.left = left;
*         this.right = right;
*     }
* }
*/
class Solution {
    private int preorderIdx = 0;
    private TreeNode construct(int[] preorder, HashMap<Integer,Integer> map, int left, int right){
        if(left>right) return null;
        int pval = preorder[preorderIdx];
        int inorderIdx = map.get(pval);

        TreeNode root = new TreeNode(pval);
        preorderIdx++;
        root.left = construct(preorder, map, left, inorderIdx-1);
        root.right = construct(preorder, map, inorderIdx+1, right);
        return root;

    }
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        HashMap<Integer,Integer> map = new HashMap<>();
        int len = inorder.length;
        for(int i=0;i<len;i++){
            map.put(inorder[i],i);
```

```
        }


        return construct(preorder, map, 0, len-1);
    }
}
```

https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode buildTree(int[] inorder, int[] postorder) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < inorder.length; i++){
            map.put(inorder[i], i);
        }
        return buildTree(inorder, postorder, 0, inorder.length - 1, 0, postorder.length - 1, map);
    }
    private TreeNode buildTree(int[] inorder, int[] postorder, int inStart, int inEnd, int postStart, int
postEnd, HashMap<Integer, Integer> map){
```

```
        if (inStart > inEnd || postStart > postEnd){
            return null;
        }
        int rootVal = postorder[postEnd];
        TreeNode root = new TreeNode(rootVal);
        int rootIndex = map.get(rootVal);
        int leftSize = rootIndex - inStart;
        root.left = buildTree(inorder, postorder, inStart, rootIndex - 1, postStart, postStart + leftSize - 1,
map);
        root.right = buildTree(inorder, postorder, inStart + leftSize + 1, inEnd, postStart + leftSize,
postEnd - 1, map);
        return root;
    }
}
```