```
public class Codec {
    String NN="X";
    String spliter=",";
    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        StringBuilder sb=new StringBuilder();
        buildString(root,sb);
        return sb.toString();
    }
    private void buildString(TreeNode node, StringBuilder sb){
        if(node==null){
            sb.append(NN);
            sb.append(spliter);
        }else{
            sb.append(node.val);
            sb.append(spliter);
            buildString(node.left,sb);
            buildString(node.right,sb);
        }
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        Deque<String> deque=new ArrayDeque<>(Arrays.asList(data.split(spliter)));
        return buildTree(deque);

    }

    private TreeNode buildTree(Deque<String> deque){
        String s=deque.removeFirst();
        if(s.equals(NN)){
            return null;
        }else{
            int val=Integer.valueOf(s);
            TreeNode node=new TreeNode(val);
            node.left=buildTree(deque);
            node.right=buildTree(deque);
            return node;
        }

    }
}
```

```
class Codec {
    String NN="X";
    String spliter=",";
    // Encodes a tree to a single string.
    public String serialize(Node root) {
        StringBuilder sb=new StringBuilder();
        buildString(root,sb);
        return sb.toString();
    }
    private void buildString(Node node, StringBuilder sb){
        if(node==null){
            sb.append(NN);
            sb.append(spliter);
        }else{
            sb.append(node.val);
            sb.append(spliter);
            sb.append(node.children.size());
            sb.append(spliter);
            for (Node child:node.children){
                buildString(child,sb);
            }
        }
    }

    // Decodes your encoded data to tree.
    public Node deserialize(String data) {
        Deque<String> deque=new ArrayDeque<>(Arrays.asList(data.split(spliter)));
        return buildTree(deque);
    }
    private Node buildTree(Deque<String> deque){
        String s1=deque.removeFirst();
        if(s1.equals(NN)) return null;

        int rootVal=Integer.valueOf(s1);
        int childrenNumber=Integer.valueOf(deque.removeFirst());

        Node root=new Node(rootVal);
        root.children=new ArrayList<>();
        for (int i=0;i<childrenNumber;i++){
            root.children.add(buildTree(deque));
        }
        return root;
    }
}
```

```
public String serialize(Node root) {
        List<String> list = new ArrayList<>();
        serializeHelper(root, list);
        StringBuilder sb = new StringBuilder();
        for (String str: list) {
            sb.append(str);
            sb.append(",");
        }
        if(sb.length() > 0) sb.deleteCharAt(sb.length()-1); // Delete last ","

        return sb.toString();
    }

    private void serializeHelper(Node root, List<String> list) {
        if(root == null) {
            list.add("#");
        }
        else {
            list.add(String.valueOf(root.val));
            int size = root.children.size();
            if(root.children.size() > 0) {
                list.add(String.valueOf(size));
                for (Node child : root.children) {
                    serializeHelper(child, list);
                }
            }
            else {
                list.add("#");
            }
        }
    }

    // Decodes your encoded data to tree.
    public Node deserialize(String data) {
        ArrayDeque<String> queue = new ArrayDeque<>();
        String[] split = data.split(",");
        for (String str : split) queue.add(str);

        return deserializeHelper(queue);

    }

    private Node deserializeHelper(ArrayDeque<String> queue) {
        String s = queue.poll();
        if(s.equals("#")) {
            return null;
```

```
    }
    // number of children，can be NN or a number
    String numStr = queue.poll();


    Node root = new Node(Integer.valueOf(s), new ArrayList<>());
    if(numStr.equals("#")) {
        return root;
    }
    else {
        int num = Integer.valueOf(numStr);
        for (int i = 0; i < num; i++) {
            root.children.add(deserializeHelper(queue));
        }
    }

    return root;
}
```