public class Solution {

// Encodes a tree to a single string.

public String serialize(TreeNode root) {

if (root == null) return "";

String n = "null", sep = ",";

Queue<TreeNode> dq = new LinkedList<>();

dq.offer(root);

int size = 0;

TreeNode cur;

StringBuilder sb = new StringBuilder();

while (!dq.isEmpty()) {

size = dq.size();

for (int i = 0; i < size; ++i) {

cur = dq.poll();

if (cur != null) {

sb.append(cur.val);

dq.offer(cur.left);

dq.offer(cur.right);

} else {

sb.append(n);

}

sb.append(sep);

}

}

return sb.toString();

}

// Decodes your encoded data to tree.

public TreeNode deserialize(String data) {

if (data == null || data.length() == 0) return null;

String[] vals = data.split(",");

if (vals == null || vals.length == 0) return null;

String n = "null";

TreeNode root = new TreeNode(Integer.parseInt(vals[0]));

TreeNode cur, next;

Deque<TreeNode> dq = new ArrayDeque<>();

int size = 0, index = 1;

dq.offer(root);

while (!dq.isEmpty()) {

size = dq.size();

for (int i = 0; i < size; ++i) {

cur = dq.poll();

for (int j = index; j < index + 2 && j < vals.length; ++j) {

if (vals[j].equals(n)) {

if (j % 2 == 1) {

cur.left = null;

} else {

cur.right = null;

}

} else {

next = new TreeNode(Integer.parseInt(vals[j]));

dq.offer(next);

if (j % 2 == 1) {

cur.left = next;

} else {

cur.right = next;

}

}

}

index += 2;

}

}

return root;

}

}