Code : 1  
package Topic\_15\_Trees;

import java.io.\*;

import java.util.\*;

public class F\_GenericTree {

private static class Node {

int data;

ArrayList<Node> children = new ArrayList<>();

}

public static void display(Node node) {

String str = node.data + " -> ";

for (Node child : node.children) {

str += child.data + ", ";

}

str += ".";

System.out.println(str);

for (Node child : node.children) {

display(child);

}

}

public static int size(Node node) {

int s = 0;

for (Node child : node.children) {

s += size(child);

}

s += 1;

return s;

}

public static int max(Node node) {

int m = Integer.MIN\_VALUE;

for (Node child : node.children) {

int cm = max(child);

m = Math.max(m, cm);

}

m = Math.max(m, node.data);

return m;

}

public static int height(Node node) {

int h = -1;

for (Node child : node.children) {

int ch = height(child);

h = Math.max(h, ch);

}

h += 1;

return h;

}

//Generic Tree - Traversals (pre-order, Post-order)

public static void traversals(Node node) {

System.out.println("Node Pre " + node.data);

for (Node child : node.children) {

System.out.println("Edge Pre " + node.data + "--" + child.data);

traversals(child);

System.out.println("Edge Post " + node.data + "--" + child.data);

}

System.out.println("Node Post " + node.data);

}

public static void levelOrder(Node root) {

Queue<Node> queue = new ArrayDeque<Node>();

queue.add(root);

while (queue.size() > 0) {

// r,p,a

Node temp = queue.remove();

System.out.print(temp.data + " ");

for (Node child : temp.children) {

queue.add(child);

}

}

System.out.println(".");

}

public static void levelOrderLinewise(Node root) {

Queue<Node> queue = new ArrayDeque<Node>();

Queue<Node> cqueue = new ArrayDeque<Node>();

queue.add(root);

while (queue.size() > 0) {

Node temp = queue.remove();

System.out.print(temp.data + " ");

for (Node child : temp.children) {

cqueue.add(child);

}

if (queue.size() == 0) {

queue = cqueue;

cqueue = new ArrayDeque<>();

System.out.println("");

}

}

}

//Extra question

public static void levelOrderLinewiseZZ(Node node) {

Stack<Node> stack = new Stack<>();

stack.add(node);

Stack<Node> cstack = new Stack<>();

int level = 0;

while (stack.size() > 0) {

node = stack.pop();

System.out.print(node.data + " ");

if (level % 2 == 0) {

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

Node child = node.children.get(i);

cstack.push(child);

}

} else {

for (int i = node.children.size() - 1; i >= 0; i--) {

Node child = node.children.get(i);

cstack.push(child);

}

}

if (stack.size() == 0) {

stack = cstack;

cstack = new Stack<>();

level++;

System.out.println();

}

}

}

public static void mirror(Node node) {

for (Node child : node.children) {

mirror(child);

}

Collections.reverse(node.children);

}

public static void removeLeaves(Node node) {

for (int i = node.children.size() - 1; i >= 0; i--) {

Node child = node.children.get(i);

if (child.children.size() == 0) {

node.children.remove(i);

}

}

for (Node child : node.children) {

removeLeaves(child);

}

}

private static Node getTail(Node node) {

while (node.children.size() == 1) {

node = node.children.get(0);

}

return node;

}

public static void linearize(Node node) {

for (Node child : node.children) {

linearize(child);

}

while (node.children.size() > 1) {

Node lc = node.children.remove(node.children.size() - 1);

Node sl = node.children.get(node.children.size() - 1);

Node slt = getTail(sl);

slt.children.add(lc);

}

}

public static Node linearizeEfficient(Node node) {

if (node.children.size() == 0) {

return node;

}

Node lastChild = node.children.get(node.children.size() - 1);

Node lastKiTail = linearizeEfficient(lastChild);

while (node.children.size() > 1) {

Node slastChild = node.children.get(node.children.size() - 2);

Node slastKiTail = linearizeEfficient(slastChild);

slastKiTail.children.add(lastChild);

node.children.remove(node.children.size() - 1);

lastChild = slastChild;

}

return lastKiTail;

}

public static boolean findANodeInTree(Node node, int data) {

if (node.data == data) {

return true;

}

for (Node child : node.children) {

boolean fic = findANodeInTree(child, data);

if (fic == true) {

return true;

}

}

return false;

}

public static ArrayList<Integer> nodeToRootPath(Node node, int data) {

if (node.data == data) {

ArrayList<Integer> bres = new ArrayList<>();

bres.add(node.data);

return bres;

}

for (Node child : node.children) {

ArrayList<Integer> nodeToChildPath = nodeToRootPath(child, data);

if (nodeToChildPath.size() > 0) {

nodeToChildPath.add(node.data);

return nodeToChildPath;

}

}

return new ArrayList<>();

}

public static int LowestCommonAncestor(Node node, int d1, int d2) {

ArrayList<Integer> path1 = nodeToRootPath(node, d1);

ArrayList<Integer> path2 = nodeToRootPath(node, d2);

int i = path1.size() - 1;

int j = path2.size() - 1;

while (i >= 0 && j >= 0) {

if (path1.get(i) == path2.get(j)) {

i--;

j--;

} else {

break;

}

}

int lca = path1.get(i + 1);

return lca;

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int[] arr = new int[n];

String[] values = br.readLine().split(" ");

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

arr[i] = Integer.parseInt(values[i]);

}

Node root = construct(arr);

linearize(root);

display(root);

}

public static Node construct(int[] arr) {

Node root = null;

Stack<Node> st = new Stack<>();

for (int i = 0; i < arr.length; i++) {

if (arr[i] == -1) {

st.pop();

} else {

Node t = new Node();

t.data = arr[i];

if (st.size() > 0) {

st.peek().children.add(t);

} else {

root = t;

}

st.push(t);

}

}

return root;

}

}