**Assignment21**

**1.**

class TreeNode {

constructor(value) {

this.val = value;

this.left = null;

this.right = null;

}

}

function binaryTreeToBST(root) {

if (root === null) {

return null;

}

const values = [];

function inOrderTraversal(node) {

if (node === null) {

return;

}

inOrderTraversal(node.left);

values.push(node.val);

inOrderTraversal(node.right);

}

inOrderTraversal(root);

values.sort((a, b) => a - b);

let index = 0;

function inOrderReplace(node) {

if (node === null) {

return;

}

inOrderReplace(node.left);

node.val = values[index++];

inOrderReplace(node.right);

}

inOrderReplace(root);

return root;

}

// Example:

const root = new TreeNode(10);

root.left = new TreeNode(2);

root.right = new TreeNode(7);

root.left.left = new TreeNode(8);

root.left.right = new TreeNode(4);

const result = binaryTreeToBST(root);

function inOrderTraversal(node) {

if (node === null) {

return;

}

inOrderTraversal(node.left);

console.log(node.val);

inOrderTraversal(node.right);

}

inOrderTraversal(result);

**2.**

class TreeNode {

constructor(value) {

this.val = value;

this.left = null;

this.right = null;

}

}

function findLCA(root, node1, node2) {

if (root === null) {

return null;

}

if (root.val > node1.val && root.val > node2.val) {

return findLCA(root.left, node1, node2);

} else if (root.val < node1.val && root.val < node2.val) {

return findLCA(root.right, node1, node2);

} else {

return root;

}

}

function findDistance(root, node, distance) {

if (root === null) {

return -1;

}

if (root.val === node.val) {

return distance;

}

if (root.val > node.val) {

return findDistance(root.left, node, distance + 1);

} else {

return findDistance(root.right, node, distance + 1);

}

}

function distanceBetweenNodes(root, node1, node2) {

const lca = findLCA(root, node1, node2);

const distance1 = findDistance(lca, node1, 0);

const distance2 = findDistance(lca, node2, 0);

return distance1 + distance2;

}

// Example:

const root = new TreeNode(8);

root.left = new TreeNode(3);

root.right = new TreeNode(10);

root.left.left = new TreeNode(1);

root.left.right = new TreeNode(6);

root.left.right.left = new TreeNode(4);

root.left.right.right = new TreeNode(7);

root.right.right = new TreeNode(14);

root.right.right.left = new TreeNode(13);

const node1 = new TreeNode(6);

const node2 = new TreeNode(14);

console.log("The distance between the two keys =", distanceBetweenNodes(root, node1, node2));

Output: 4

const node3 = new TreeNode(3);

const node4 = new TreeNode(4);

console.log("The distance between the two keys =", distanceBetweenNodes(root, node3, node4));

Output: 2

**3.**

class Node {

constructor(value) {

this.val = value;

this.left = null;

this.right = null;

}

}

function convertBinaryTreeToDLL(root) {

let head = null;

let prev = null;

function convertToDLL(node) {

if (node === null) {

return;

}

convertToDLL(node.left);

if (prev === null) {

head = node;

} else {

prev.right = node;

node.left = prev;

}

prev = node;

convertToDLL(node.right);

}

convertToDLL(root);

return head;

}

// Example:

const root = new Node(10);

root.left = new Node(5);

root.right = new Node(20);

root.right.left = new Node(30);

root.right.right = new Node(35);

const head = convertBinaryTreeToDLL(root);

function traverseForward(head) {

let current = head;

while (current !== null) {

console.log(current.val);

current = current.right;

}

}

traverseForward(head);

Output: 5 10 30 20 35

**4.**

class Node {

constructor(value) {

this.val = value;

this.left = null;

this.right = null;

this.next = null;

}

}

function connectNodesAtSameLevel(root) {

if (root === null) {

return;

}

const LevelMarker = new Node(-1);

const queue = [];

queue.push(root);

queue.push(LevelMarker);

while (queue.length > 0) {

const node = queue.shift();

if (node === LevelMarker) {

if (queue.length > 0) {

queue.push(LevelMarker);

}

} else {

node.next = queue[0];

if (node.left !== null) {

queue.push(node.left);

}

if (node.right !== null) {

queue.push(node.right);

}

}

}

return root;

}

// Example:

const root = new Node(1);

root.left = new Node(2);

root.right = new Node(3);

root.left.left = new Node(4);

root.left.right = new Node(5);

root.right.left = new Node(6);

root.right.right = new Node(7);

const result = connectNodesAtSameLevel(root);

function printConnectedNodes(root) {

let current = root;

while (current !== null) {

let node = current;

while (node !== null) {

let nextNode = node.next ? node.next.val : -1;

process.stdout.write(`${node.val} → ${nextNode} `);

node = node.next;

}

console.log();

current = current.left;

}

}

printConnectedNodes(result);