**Java - DSA**

**Trees**

1. **Build Tree from given Preorder Sequence**

//Build a Tree from its Preorder traversal

public class BinaryTreesYT {

static class Node {

int data;

Node left;

Node right;

Node(int data) {

this.data = data;

this.left = null;

this.right = null;

}

}

static class BinaryTree {

static int idx = -1;

public static Node buildTree(int nodes[]) {

idx++;

if(nodes[idx] == -1) {

return null;

}

Node newNode = new Node(nodes[idx]);

newNode.left = buildTree(nodes);

newNode.right = buildTree(nodes);

return newNode;

}

}

public static void main(String args[]) {

int nodes[] = {1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1};

BinaryTree tree = new BinaryTree();

Node root = tree.buildTree(nodes);

System.out.println(root.data);

}

}

1. **Tree Traversals**
2. Preorder

public static void preorder(Node root) {

if(root == null) {

System.out.print(-1+" ");

return;

}

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

preorder(root.left);

preorder(root.right);

}

1. Inorder

public static void inorder(Node root) {

if(root == null) {

System.out.print(-1+" ");

return;

}

inorder(root.left);

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

inorder(root.right);

}

1. Postorder

public static void postorder(Node root) {

if(root == null) {

System.out.print(-1+" ");

return;

}

postorder(root.left);

postorder(root.right);

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

}

1. Level Order

public static void levelOrder(Node root) {

if(root == null) {

return;

}

Queue<Node> q = new LinkedList<>();

q.add(root);

q.add(null);

while(!q.isEmpty()) {

Node curr = q.remove();

if(curr == null) {

System.out.println();

//queue empty

if(q.isEmpty()) {

break;

} else {

q.add(null);

}

} else {

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

if(curr.left != null) {

q.add(curr.left);

}

if(curr.right != null) {

q.add(curr.right);

}

}

}

}

**3. Height of Tree**

public static int height(Node root) {

if(root == null) {

return 0;

}

int leftHeight = height(root.left);

int rightHeight = height(root.right);

return Math.max(leftHeight, rightHeight) + 1;

}

**4. Count of Nodes of Tree**

public static int countOfNodes(Node root) {

if(root == null) {

return 0;

}

int leftNodes = countOfNodes(root.left);

int rightNodes = countOfNodes(root.right);

return leftNodes + rightNodes + 1;

}

**5. Sum of Nodes of Tree**

public static int sumOfNodes(Node root) {

if(root == null) {

return 0;

}

int leftSum = sumOfNodes(root.left);

int rightSum = sumOfNodes(root.right);

return leftSum + rightSum + root.data;

}

**6. Diameter of Tree - Approach1 O(N^2)**

public static int diameter(Node root) {

if(root == null) {

return 0;

}

int diam1 = height(root.left) + height(root.right) + 1;

int diam2 = diameter(root.left);

int diam3 = diameter(root.right);

return Math.max(diam1, Math.max(diam2, diam3));

}

**7. Diameter of Tree - Approach2 O(N)**

public static TreeInfo diameter(Node root) {

if(root == null) {

return new TreeInfo(0, 0);

}

TreeInfo leftTI = diameter(root.left);

TreeInfo rightTI = diameter(root.right);

int myHeight = Math.max(leftTI.height, rightTI.height) + 1;

int diam1 = leftTI.height + rightTI.height + 1;

int diam2 = leftTI.diam;

int diam3 = rightTI.diam;

int myDiam = Math.max(diam1, Math.max(diam2, diam3));

return new TreeInfo(myHeight, myDiam);

}

**8. Subtree of another tree**

public boolean isIdentical(TreeNode root,TreeNode subRoot){

if(subRoot == null && root == null){

return true;

}

if(root == null || subRoot == null){

return false;

}

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

return isIdentical(root.left, subRoot.left) && isIdentical(root.right, subRoot.right);

}

return false;

}

public boolean isSubtree(TreeNode root, TreeNode subRoot) {

if(subRoot == null){

return true;

}

if(root == null){

return false;

}

if(isIdentical(root, subRoot)){

return true;

}

return isSubtree(root.left, subRoot) || isSubtree(root.right, subRoot);

}