import java.sql.SQLOutput;  
import java.util.LinkedList;  
import java.util.Queue;  
  
public class PreorderTree {  
  
 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 *index*=-1;  
 public static Node buildTree(int[] nodes)  
 {  
 *index*++;  
 if(nodes[*index*]==-1)  
 {  
 return null;  
 }  
 Node newnode=new Node(nodes[*index*]);  
 newnode.left=*buildTree*(nodes);  
 newnode.right=*buildTree*(nodes);  
  
 return newnode;  
 }  
 }  
  
 public static void inorder(Node root)  
 {  
 if(root==null)  
 {  
 return;  
 }  
 *inorder*(root.left);  
 System.*out*.print(root.data + " ");  
 *inorder*(root.right);  
  
 }  
  
 public static void postorder(Node root)  
 {  
 if(root==null)  
 {  
 return;  
 }  
 *postorder*(root.left);  
 *postorder*(root.right);  
 System.*out*.print(root.data + " ");  
 }  
  
 public static void preorder(Node root) *//time complexity- O(n)* {  
 if(root==null)  
 {  
 return;  
 }  
 System.*out*.print(root.data+ " ");  
 *preorder*(root.left);  
 *preorder*(root.right);  
 }  
  
 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 currnode = q.remove();  
 if(currnode==null)  
 {  
 System.*out*.println("");  
 if(q.isEmpty())  
 {  
 break;  
 }  
 else  
 {  
 q.add(null);  
 }  
 }  
 else  
 {  
 System.*out*.print(currnode.data);  
 if(currnode.left!=null)  
 {  
 q.add(currnode.left);  
 }  
 if(currnode.right!=null)  
 {  
 q.add(currnode.right);  
 }  
  
 }  
 }  
  
 }  
  
 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;  
 }  
  
 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;  
 }  
  
 public static int height(Node root)  
 {  
 if(root==null)  
 {  
 return 0;  
 }  
 int leftHeight=*height*(root.left);  
 int rightHeight=*height*(root.right);  
 int myHeight = Math.*max*(leftHeight,rightHeight) + 1;  
 return myHeight;  
 }  
  
 public static int diameter(Node root) *//time complexity - O(n^2)* {  
 if(root==null)  
 {  
 return 0;  
 }  
 int diam1=*diameter*(root.left);  
 int diam2=*diameter*(root.right);  
 int diam3= *height*(root.left) +*height*(root.right) + 1;  
  
 return Math.*max*(diam3, Math.*max*(diam1,diam2));  
 }  
  
 *//to calculate diameter of a tree using linear time complexity* static class TreeInfo{  
 int ht;  
 int diam;  
  
 TreeInfo(int ht,int diam)  
 {  
 this.ht=ht;  
 this.diam=diam;  
 }  
 }  
  
 public static TreeInfo diameterlinear(Node root) *//time complexity - O(n)* {  
 if(root==null)  
 {  
 return new TreeInfo(0,0);  
 }  
 TreeInfo left=*diameterlinear*(root.left);  
 TreeInfo right=*diameterlinear*(root.right);  
  
 int myHeight=Math.*max*(left.ht,right.ht) +1;  
  
  
 int diam1=left.diam;  
 int diam2=right.diam;  
 int diam3= left.ht + right.ht + 1;  
  
 int mydiam=Math.*max*(Math.*max*(diam1, diam2),diam3);  
  
 TreeInfo myInfo = new TreeInfo(myHeight,mydiam);  
 return myInfo;  
 }  
   
 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*.print("Preorder Traversal -> ");  
 *preorder*(root);  
 System.*out*.println("");  
 System.*out*.print("Inorder Traversal -> ");  
 *inorder*(root);  
 System.*out*.println("");  
 System.*out*.print("Postorder Traversal -> ");  
 *postorder*(root);  
 System.*out*.println("");  
 System.*out*.print("Levelorder Traversal -> ");  
 *levelOrder*(root);  
  
 System.*out*.println("Tree count - " + *countofNodes*(root));  
 System.*out*.println("Tree Sum count - " + *sumofNodes*(root));  
 System.*out*.println("Height of Tree - " + *height*(root));  
 System.*out*.println("Diameter of Tree - " + *diameter*(root));  
 System.*out*.println("Diameter of Tree (Linear Complexity) - " + *diameterlinear*(root).diam);  
  
 }  
  
}

Output –

Preorder Traversal -> 1 2 4 5 3 6

Inorder Traversal -> 4 2 5 1 3 6

Postorder Traversal -> 4 5 2 6 3 1

Levelorder Traversal -> 1

23

456

Tree count - 6

Tree Sum count - 21

Height of Tree - 3

Diameter of Tree - 5

Diameter of Tree (Linear Complexity) - 5