1. Mark the following statements as true or false.

a. A binary tree must be nonempty. True

b. The level of the root node is 0. True

c. If a tree has only one node, the height of this tree is 0 because the number of levels is 0. True

d. The inorder traversal of a binary tree always outputs the data in ascending order. false

1. The binary tree of the following Figure is to be used for Exercises 1 through 6.



1. Find LA, the node in the left subtree of A.

2. Find RA, the node in the right subtree of A.

3. Find RB, the node in the right subtree of B.

4. List the nodes of this binary tree in an inorder sequence.

5. List the nodes of this binary tree in a preorder sequence.

6. List the nodes of this binary tree in a postorder sequence.

1. The binary search tree of the following Figure is to be used for Exercises 1through 4.



1. List the path from the node with info 80 to the node

with info 79.

1. A node with info 35 is to be inserted in the tree.

List the nodes that are visited by the function

insert to insert 35. Redraw the tree after inserting 35.

1. Delete node 52 and redraw the binary tree.
2. Delete node 40 and redraw the binary tree.
3. Delete nodes 80 and 58 in that order. Redraw the

binary tree after each deletion.

1. Write the definition of the function, **nodeCount**, that returns the number of nodes in a binary tree.

public class BinaryTree {

Node root;

// Definition of the Node class

static class Node {

int data;

Node left, right;

Node(int data) {

this.data = data;

left = right = null;

}

}

// Function to count the number of nodes in a binary tree

public int nodeCount(Node node) {

if (node == null) {

return 0;

}

int leftCount = nodeCount(node.left);

int rightCount = nodeCount(node.right);

return 1 + leftCount + rightCount;

}

public static void main(String[] args) {

BinaryTree tree = new BinaryTree();

// Creating a sample binary tree

tree.root = new Node(1);

tree.root.left = new Node(2);

tree.root.right = new Node(3);

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

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

// Counting the number of nodes

int count = tree.nodeCount(tree.root);

System.out.println("Number of nodes in the binary tree: " + count);

}

}

In this code, the nodeCount function takes a Node as input and recursively counts the number of nodes in the binary tree. If the input node is null, it returns 0. Otherwise, it recursively counts the nodes in the left subtree and the right subtree, and adds 1 to account for the current node. The main function demonstrates an example usage of the nodeCount function by creating a sample binary tree and counting its nodes.

1. Write the definition of the function, **leavesCount**, that takes as a parameter a reference to the root node of a binary tree and returns the number of leaves in a binary tree.

public class BinaryTree {

Node root;

// Definition of the Node class

static class Node {

int data;

Node left, right;

Node(int data) {

this.data = data;

left = right = null;

}

}

// Function to count the number of leaves in a binary tree

public int leavesCount(Node node) {

if (node == null) {

return 0;

}

if (node.left == null && node.right == null) {

// Node is a leaf

return 1;

}

int leftCount = leavesCount(node.left);

int rightCount = leavesCount(node.right);

return leftCount + rightCount;

}

public static void main(String[] args) {

BinaryTree tree = new BinaryTree();

// Creating a sample binary tree

tree.root = new Node(1);

tree.root.left = new Node(2);

tree.root.right = new Node(3);

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

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

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

// Counting the number of leaves

int count = tree.leavesCount(tree.root);

System.out.println("Number of leaves in the binary tree: " + count);

}

}

In this code, the leavesCount function takes a Node as input and recursively counts the number of leaves in the binary tree. If the input node is null, it returns 0. If the node has no children (left and right pointers are null), it is considered a leaf and the function returns 1. Otherwise, it recursively counts the leaves in the left subtree and the right subtree and returns their sum. The main function demonstrates an example usage of the leavesCount function by creating a sample binary

1. Draw the binary tree representation of the following arithmetic expression:

“(((5+2) ∗ (2−1))/((2+9)+((7−2)−1)) ∗8)”.

public class BinaryTree {

Node root;

// Definition of the Node class

static class Node {

String data;

Node left, right;

Node(String data) {

this.data = data;

left = right = null;

}

}

// Function to build the binary expression tree

public Node buildExpressionTree(String[] expression, int startIndex) {

if (startIndex >= expression.length) {

return null;

}

Node node = new Node(expression[startIndex]);

if (isOperator(expression[startIndex])) {

node.left = buildExpressionTree(expression, startIndex + 1);

node.right = buildExpressionTree(expression, startIndex + 2);

}

return node;

}

// Function to check if a string is an operator

public boolean isOperator(String str) {

return str.equals("+") || str.equals("-") || str.equals("\*") || str.equals("/");

}

// Function to print the binary expression tree (inorder traversal)

public void printExpressionTree(Node node) {

if (node != null) {

printExpressionTree(node.left);

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

printExpressionTree(node.right);

}

}

public static void main(String[] args) {

BinaryTree tree = new BinaryTree();

// Arithmetic expression

String expression = "(((5+2)\* (2-1))/((2+9)+((7-2)-1)) \*8)";

// Splitting the expression into individual tokens

String[] tokens = expression.split("\\s+");

// Building the binary expression tree

tree.root = tree.buildExpressionTree(tokens, 0);

// Printing the binary expression tree

tree.printExpressionTree(tree.root);

}

}

1. Insert, into an empty binary search tree, entries with keys 30, 40, 24, 58, 48, 26, 11, 13 (in this order). Draw the tree after each insertion.

Step 1: Insert 30

30

Step 2: Insert 40

30

\

40

Step 3: Insert 24

30

/ \

24 40

Step 4: Insert 58

30

/ \

24 40

\

58

Step 5: Insert 48

30

/ \

24 40

\

58

/

48

Step 6: Insert 26

30

/ \

24 40

\ \

26 58

/

48

Step 7: Insert 11

30

/ \

24 40

\ \

26 58

/ /

11 48

Step 8: Insert 13

30

/ \

24 40

\ \

26 58

/ /

11 48

\

13

يمكن حل أسئلة من الكتاب , بالإضافة الى المحاضرات

**Good Luck**