

DATA STRUCTURES & ALGORITHMS

#10

Tree basics

|  |
| --- |
| Student Name: SAAD MAQSOOD |
| Roll Number: 22SP-017-SE Section: A |
| Work submitted on: MS-Teams |

|  |  |  |  |
| --- | --- | --- | --- |
| **Maximum Marks** | **Performance** | **Viva** | **Total** |
| **Marks Obtained** |  |  |  |
| **Remarks (if any)** |  | | |
|  | | | |
| **Experiment evaluated by** | | | |
| Instructor Name: | | | |
| Signature: | | | |

|  |
| --- |
| Tree Related Tasks |

Question: Write a function to calculate the height of a binary tree.

Example:

Input:

5

/

3 7

/

2 4

Output: 2

**Code:**

public static int levelOrderTraversal(Node root)

        {

            if(root == null)

            {

                return -1;

            }

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

            q.add(root);

            q.add(null);

            int height = 0;

            while(!q.isEmpty())

            {

                Node currNode = q.remove();

                if(currNode == null)

                {

                    // We use null for marking it as the end of level

                    // System.out.println();  // and so we change line

                    // End of level we can also calculate height

                    height++;

                    if(q.isEmpty())

                    {  // if it is last null node then it means

                        break;  // whole tree has been traversed

                    }

                    else // removing the null and adding it again in the queue

                    {  // When the null is removed another is added at the

                        q.add(null);   // end of new added children

                    }

                }

                else

                {   //Printing values

                    //System.out.print(currNode.data + " ");

                    //Only checking the elements not the null node

                    if(currNode.left != null)

                    {

                        q.add(currNode.left);

                    }

                    if(currNode.right != null)

                    {

                        q.add(currNode.right);

                    }

                }

            }

            return height;

        }

Question: Given a binary tree, write a function to check if it is a binary search tree (BST).

Example:

Input:

4

/

2 5

/

1 3

Output: True

**Code:**

public static void InorderTraversal(Node root)

        {

            if(root == null)

            {

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

                return;

            }

// We can use in order traversal to check whether it

// is a binary search tree

            InorderTraversal(root.left); //First left

            System.out.print(root.data + ", "); //Second root node

            InorderTraversal(root.right);//Third right node

        }

Question: Write a function to find the maximum element in a binary tree.

Example:

Input:

5

/

3 7

/

2 4

Output: 7

**Code:**

 public static int maxNum(Node root, int max)

        {

            if(root == null)

            {   // if there is no num max remains same

                return max;

            }

            if(root.data > max)

            {   // if root's element is bigger than max

                max = root.data; // set max = root data

            }

            // Calling left, right nodes and saving their return values

            max = maxNum(root.left, max);  // Calling left node

            max = maxNum(root.right, max); // Calling right node

            return max;  // Returning the max num in the end

        }

Question: Write a function to check if two binary trees are identical (have the same structure and values).

Example:

Input:

Tree 1:

5

/

3 7

/

2 4

Tree 2:

5

/

3 7

/

2 4

Output: True

**Code:**

        public boolean identical\_tree(Node root1, Node root2)

        {

            if(root1 == null && root2 == null)

            { // if both nodes are empty return true

                return true;

            }

            if(root1 == null || root2 == null)

            { // if only one is empty means they are not equal

                return false;

            }

            if(root1 != root2)

            { // not equal simply means return false

                return false;

            }

            // Calling the left and right nodes and storing their

            // return

            boolean left\_equal = identical\_tree(root1, root2);

            boolean right\_equal = identical\_tree(root1, root2);

            // In the end result will be the combination of both

            // left and right node, if both are true then return

            // value will be true otherwise it will be false

            return left\_equal && right\_equal;

        }

Question: Given a binary tree, write a function to check if it is a balanced tree (the heights of the two subtrees of any node differ by at most one).

Example:

Input:

3

/

9 20

/

15 7

Output: True

**Code:**

        public boolean isBalanced(Node root)

        {

            if(root == null)

            {  // because empty tree would have no branches it

                // will be balanced;

                return true;

            }

            // getting the height of left and right subtree

            int leftNodeHeight = levelOrderTraversal(root.left);

            int rightNodeHeight = levelOrderTraversal(root.right);

            //if the height of left node minus right node is less than or equal to 1

            // and left and right subtrees are also true then the tree is balanced

            // otherwise it is not

            if(Math.abs(leftNodeHeight - rightNodeHeight) <= 1 && isBalanced(root.left) && isBalanced(root.right))

            {

                return true;

            }

            return false;

        }