**Axborot tizimlari va texnologiyalari yo’nalishi**

**122-20-guruh talabasi**

**Sattorova Mohiraning**

**Algoritmlar va berilganlar strukturasi**

**Fanidan 4-amaliy topshirig’i**

**10-topshiriq:** Berilgan sonlar ustida rootni aniqlab quyidagicha daraxt turlarini tuzing: Full binary tree, Complete binary tree, perfect binary tree, balanced binary tree, degenerate binary tree.

1. **variant:** 18.70,6,35,1,90,40,81,88,74,14,85,43,75,58,83,23,37

class Node:

def \_\_init\_\_(self, data):

self.left = None

self.right = None

self.data = data

# Insert method to create nodes

def insert(self, data):

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

# findval method to compare the value with nodes

def findval(self, lkpval):

if lkpval < self.data:

if self.left is None:

return str(lkpval)+" -> Topilmadi"

return self.left.findval(lkpval)

elif lkpval > self.data:

if self.right is None:

return str(lkpval)+" -> Topilmadi"

return self.right.findval(lkpval)

else:

return str(self.data) + " -> Topildi"

# Print the tree

def PrintTree(self):

if self.left:

self.left.PrintTree()

print(self.data,end=" , "),

if self.right:

self.right.PrintTree()

root = Node(7)

a=[70,6,35,1,90,40,81,88,74,14,85,43,75,58,83,23,37]

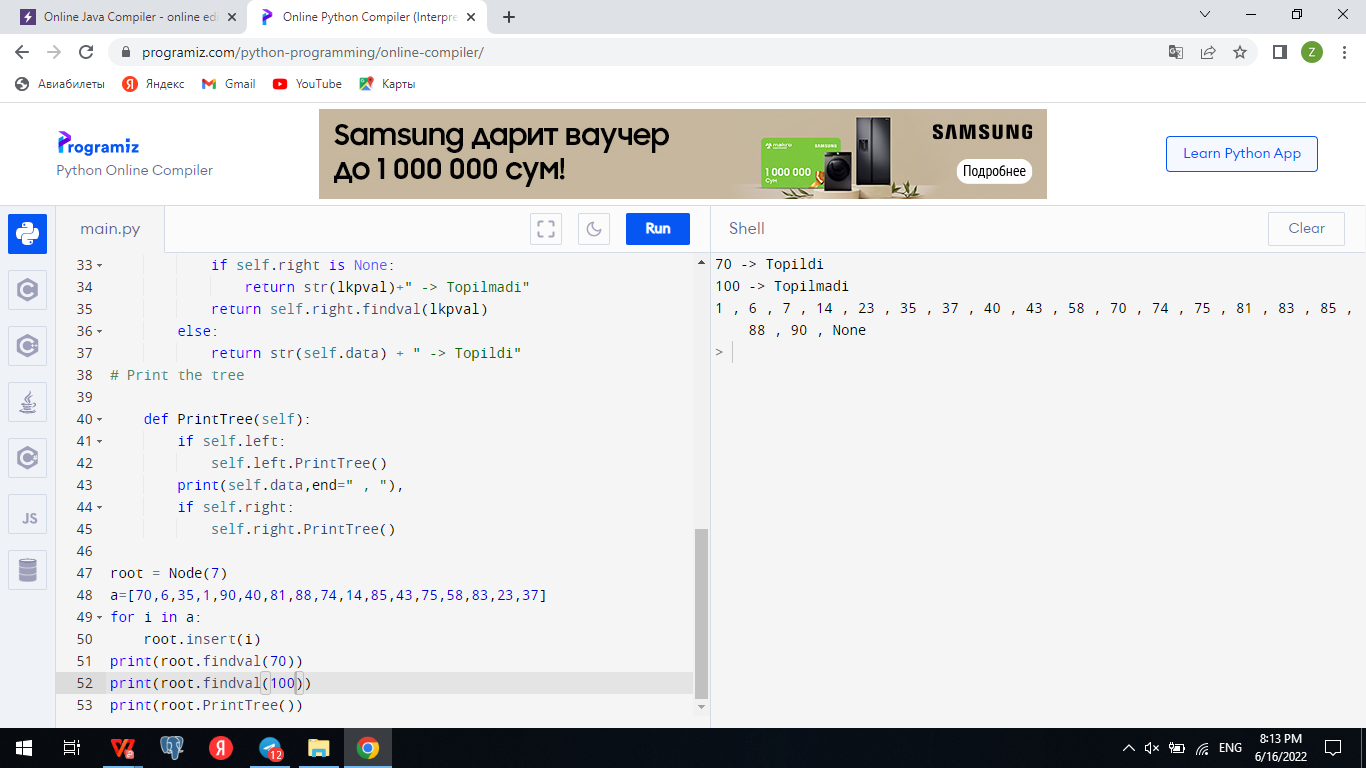
for i in a:

root.insert(i)

print(root.findval(70))

print(root.findval(100))

print(root.PrintTree())



1. **Full binary tree**

class Node {

int data;

Node leftChild, rightChild;

Node(int item) {

data = item;

leftChild = rightChild = null;

}

}

class BinaryTree {

Node root;

// Check for Full Binary Tree

boolean isFullBinaryTree(Node node) {

// Checking tree emptiness

if (node == null)

return true;

// Checking the children

if (node.leftChild == null && node.rightChild == null)

return true;

if ((node.leftChild != null) && (node.rightChild != null))

return (isFullBinaryTree(node.leftChild) && isFullBinaryTree(node.rightChild));

return false;

}

public static void main(String args[]) {

BinaryTree tree = new BinaryTree();

tree.root = new Node(1);

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

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

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

tree.root.leftChild.rightChild = new Node(5);

tree.root.rightChild.leftChild = new Node(6);

tree.root.rightChild.rightChild = new Node(7);

if (tree.isFullBinaryTree(tree.root))

System.out.print("The tree is a full binary tree");

else

System.out.print("The tree is not a full binary tree");

}

}

**2.Complate binary tree**

Node creationclass Node {

int data;

Node left, right;

Node(int item) {

data = item;

left = right = null;

}

}

class BinaryTree {

Node root;

// Count the number of nodes

int countNumNodes(Node root) {

if (root == null)

return (0);

return (1 + countNumNodes(root.left) + countNumNodes(root.right));

}

// Check for complete binary tree

boolean checkComplete(Node root, int index, int numberNodes) {

// Check if the tree is empty

if (root == null)

return true;

if (index >= numberNodes)

return false;

return (checkComplete(root.left, 2 \* index + 1, numberNodes)

&& checkComplete(root.right, 2 \* index + 2, numberNodes));

}

public static void main(String args[]) {

BinaryTree tree = new BinaryTree();

tree.root = new Node(1);

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

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

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

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

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

int node\_count = tree.countNumNodes(tree.root);

int index = 0;

if (tree.checkComplete(tree.root, index, node\_count))

System.out.println("The tree is a complete binary tree");

else

System.out.println("The tree is not a complete binary tree");

}

}

**3.Perfect binary tree**

class PerfectBinaryTree {

static class Node {

int key;

Node left, right;

}

// Calculate the depth

static int depth(Node node) {

int d = 0;

while (node != null) {

d++;

node = node.left;

}

return d;

}

// Check if the tree is perfect binary tree

static boolean is\_perfect(Node root, int d, int level) {

// Check if the tree is empty

if (root == null)

return true;

// If for children

if (root.left == null && root.right == null)

return (d == level + 1);

if (root.left == null || root.right == null)

return false;

return is\_perfect(root.left, d, level + 1) && is\_perfect(root.right, d, level + 1);

}

// Wrapper function

static boolean is\_Perfect(Node root) {

int d = depth(root);

return is\_perfect(root, d, 0);

}

// Create a new node

static Node newNode(int k) {

Node node = new Node();

node.key = k;

node.right = null;

node.left = null;

return node;

}

public static void main(String args[]) {

Node root = null;

root = newNode(1);

root.left = newNode(2);

root.right = newNode(3);

root.left.left = newNode(4);

root.left.right = newNode(5);

if (is\_Perfect(root) == true)

System.out.println("The tree is a perfect binary tree");

else

System.out.println("The tree is not a perfect binary tree");

}

}

**5.Balanced binary tree**

class Node {

int data;

Node left, right;

Node(int d) {

data = d;

left = right = null;

}

}

// Calculate heightclass Height {

int height = 0;

}

class BinaryTree {

Node root;

// Check height balance

boolean checkHeightBalance(Node root, Height height) {

// Check for emptiness

if (root == null) {

height.height = 0;

return true;

}

Height leftHeighteight = new Height(), rightHeighteight = new Height();

boolean l = checkHeightBalance(root.left, leftHeighteight);

boolean r = checkHeightBalance(root.right, rightHeighteight);

int leftHeight = leftHeighteight.height, rightHeight = rightHeighteight.height;

height.height = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

if ((leftHeight - rightHeight >= 2) || (rightHeight - leftHeight >= 2))

return false;

else

return l && r;

}

public static void main(String args[]) {

Height height = new Height();

BinaryTree tree = new BinaryTree();

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);

if (tree.checkHeightBalance(tree.root, height))

System.out.println("The tree is balanced");

else

System.out.println("The tree is not balanced");

}

}

