Vikas Jindal 1900290120128

GITHUB SUBMISSION WEEK-5

VALIDATE BST

class Node

{

int data;

Node left = null, right = null;

Node(int data) {

this.data = data;

}

}

class Main

{

public static Node insert(Node root, int key)

{

if (root == null) {

return new Node(key);

}

if (key < root.data) {

root.left = insert(root.left, key);

}

else {

root.right = insert(root.right, key);

}

return root;

}

public static boolean isBST(Node node, int minKey, int maxKey)

{

if (node == null) {

return true;

}

if (node.data < minKey || node.data > maxKey) {

return false;

}

return isBST(node.left, minKey, node.data) &&

isBST(node.right, node.data, maxKey);

}

public static void isBST(Node root)

{

if (isBST(root, Integer.MIN\_VALUE, Integer.MAX\_VALUE)) {

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

}

else {

System.out.println("The tree is not a BST!");

}

}

private static void swap(Node root)

{

Node left = root.left;

root.left = root.right;

root.right = left;

}

public static void main(String[] args)

{

int[] keys = { 15, 10, 20, 8, 12, 16, 25 };

Node root = null;

for (int key: keys) {

root = insert(root, key);

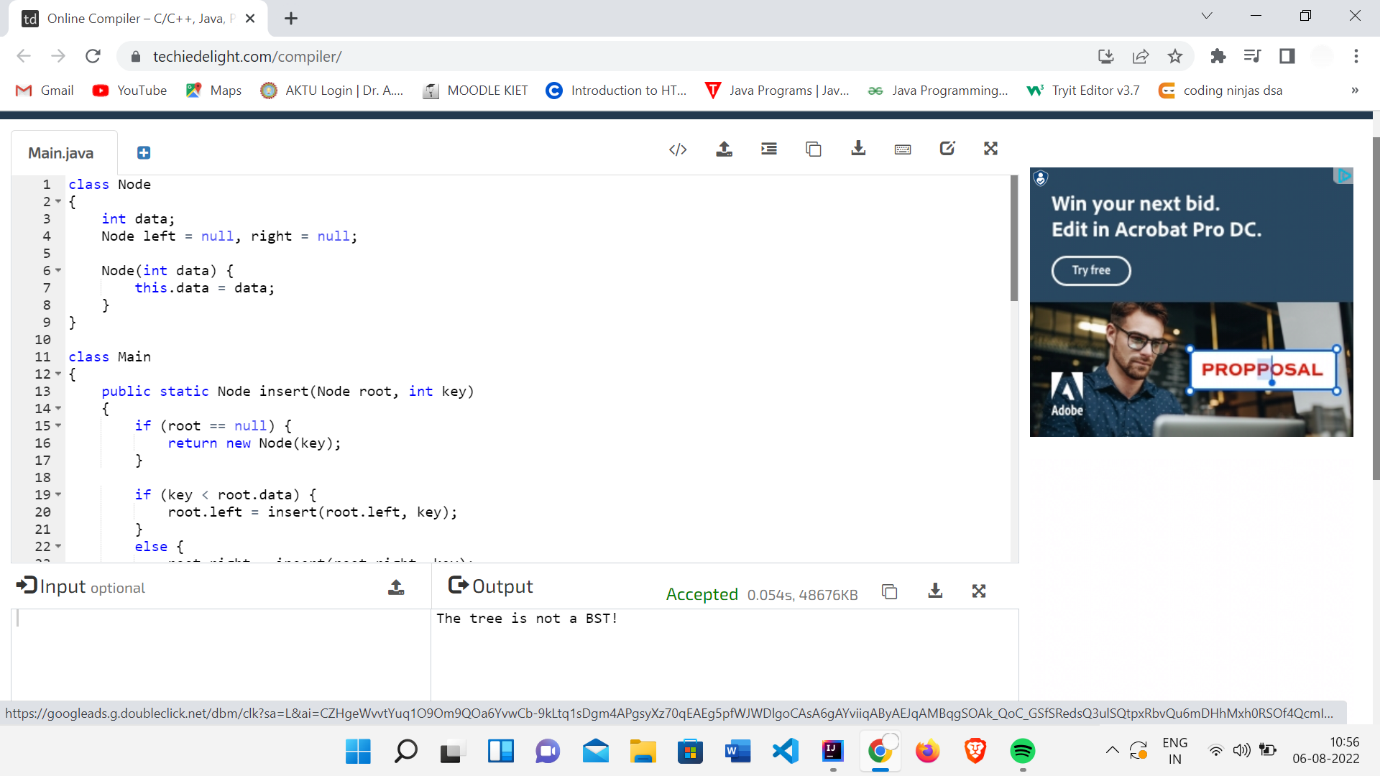
}

swap(root);

isBST(root);

}

}



POST ORDER

class Node {

int item;

Node left, right;

public Node(int key) {

item = key;

left = right = null;

}

}

class Tree {

Node root;

Tree() {

root = null;

}

void postorder(Node node) {

if (node == null)

return;

postorder(node.left);

postorder(node.right);

System.out.print(node.item + "->");

}

public static void main(String[] args) {

Tree tree = new Tree();

tree.root = new Node(1);

tree.root.left = new Node(12);

tree.root.right = new Node(9);

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

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

System.out.println("Postorder traversal");

tree.postorder(tree.root);

}

}

Graphical user interface, text, application

Description automatically generated

PREORDER

class Node {

int item;

Node left, right;

public Node(int key) {

item = key;

left = right = null;

}

}

class Tree {

Node root;

Tree() {

root = null;

}

void preorder(Node node) {

if (node == null)

return;

System.out.print(node.item + "->");

preorder(node.left);

preorder(node.right);

}

public static void main(String[] args) {

Tree tree = new Tree();

tree.root = new Node(1);

tree.root.left = new Node(12);

tree.root.right = new Node(9);

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

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

System.out.println("\nPreorder traversal ");

tree.preorder(tree.root);

}

}

Graphical user interface, text, application

Description automatically generated

INORDER

class Node {

int item;

Node left, right;

public Node(int key) {

item = key;

left = right = null;

}

}

class Tree {

Node root;

Tree() {

root = null;

}

void inOrder(Node node) {

if (node == null)

return;

inOrder(node.left);

System.out.print(node.item + "->");

inOrder(node.right);

}

public static void main(String[] args) {

Tree tree = new Tree();

tree.root = new Node(1);

tree.root.left = new Node(12);

tree.root.right = new Node(9);

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

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

System.out.println("In Order traversal");

tree.inOrder(tree.root);

}

}

Graphical user interface, application

Description automatically generated

LEVEL ORDER

class Node

{

int key;

Node left = null, right = null;

Node(int key) {

this.key = key;

}

}

class Main

{

public static boolean printLevel(Node root, int level)

{

if (root == null) {

return false;

}

if (level == 1)

{

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

return true;

}

boolean left = printLevel(root.left, level - 1);

boolean right = printLevel(root.right, level - 1);

return left || right;

}

public static void levelOrderTraversal(Node root)

{

int level = 1;

while (printLevel(root, level)) {

level++;

}

}

public static void main(String[] args)

{

Node root = new Node(15);

root.left = new Node(10);

root.right = new Node(20);

root.left.left = new Node(8);

root.left.right = new Node(12);

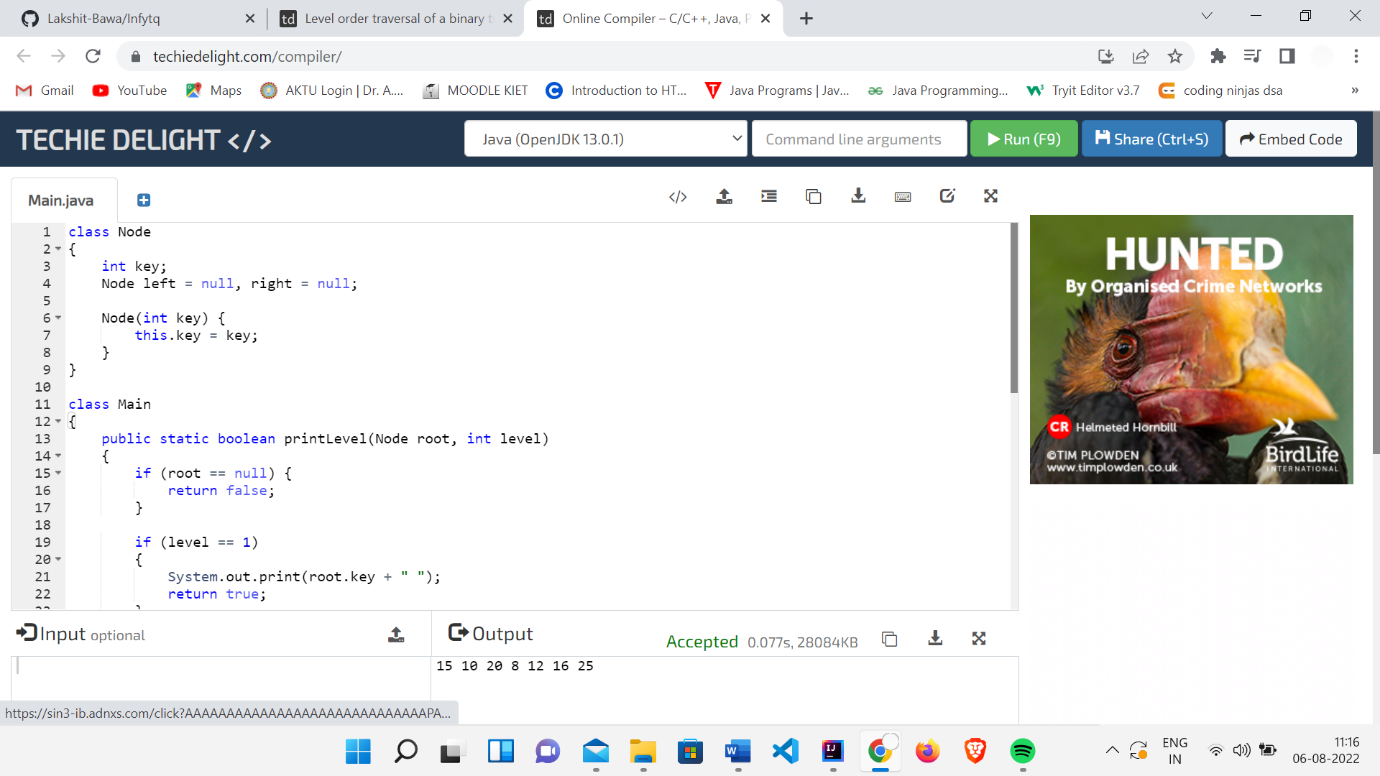
root.right.left = new Node(16);

root.right.right = new Node(25);

levelOrderTraversal(root);

}

}



BFS

import java.util.\*;

public class Graph {

private int V;

private LinkedList<Integer> adj[];

Graph(int v) {

V = v;

adj = new LinkedList[v];

for (int i = 0; i < v; ++i)

adj[i] = new LinkedList();

}

void addEdge(int v, int w) {

adj[v].add(w);

}

void BFS(int s) {

boolean visited[] = new boolean[V];

LinkedList<Integer> queue = new LinkedList();

visited[s] = true;

queue.add(s);

while (queue.size() != 0) {

s = queue.poll();

System.out.print(s + " ");

Iterator<Integer> i = adj[s].listIterator();

while (i.hasNext()) {

int n = i.next();

if (!visited[n]) {

visited[n] = true;

queue.add(n);

}

}

}

}

public static void main(String args[]) {

Graph g = new Graph(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

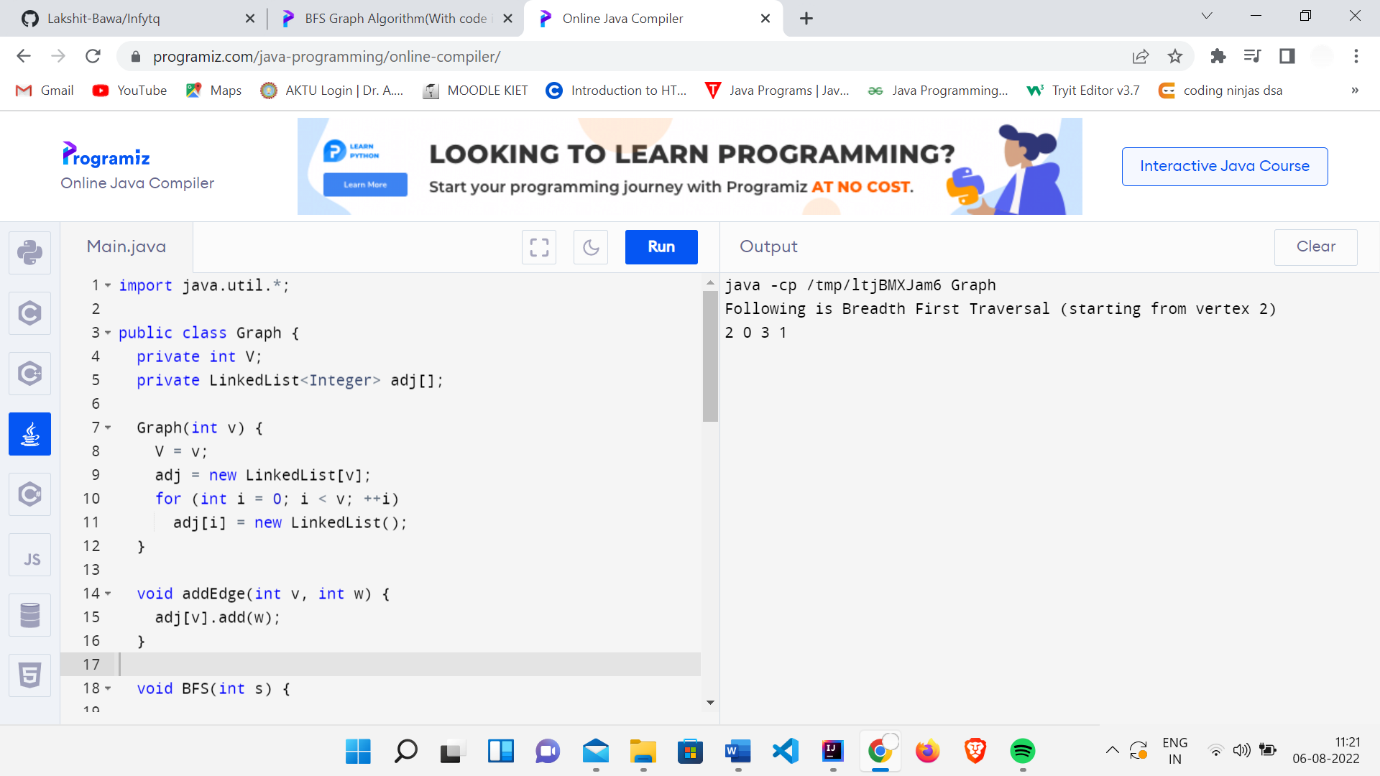
g.addEdge(3, 3);

System.out.println("Following is Breadth First Traversal " + "(starting from vertex 2)");

g.BFS(2);

}

}



DFS

import java.util.\*;

class Graph {

private LinkedList<Integer> adjLists[];

private boolean visited[];

// Graph creation

Graph(int vertices) {

adjLists = new LinkedList[vertices];

visited = new boolean[vertices];

for (int i = 0; i < vertices; i++)

adjLists[i] = new LinkedList<Integer>();

}

// Add edges

void addEdge(int src, int dest) {

adjLists[src].add(dest);

}

// DFS algorithm

void DFS(int vertex) {

visited[vertex] = true;

System.out.print(vertex + " ");

Iterator<Integer> ite = adjLists[vertex].listIterator();

while (ite.hasNext()) {

int adj = ite.next();

if (!visited[adj])

DFS(adj);

}

}

public static void main(String args[]) {

Graph g = new Graph(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 3);

System.out.println("Following is Depth First Traversal");

g.DFS(2);

}

}

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

Description automatically generated