For this project assignment, you will solve a problem based on what you have learnt in this course.

**Instructions**

* Write your name and SNU\_ID in the header of this document.
* Assignment submitted after the due date or having plagiarism will not be evaluated and a score of zero will be awarded.
* Plagiarized assignments will also be awarded zero.
* Upload a word version of this document.
* Properly document/comment your code, followed by snapshots of output as desired.

**Submitting this Assignment**

* You will submit (upload) this assignment in Blackboard.
* Email/paper submissions will not be accepted.
* Name this document as Project\_CSD203-2020\_John\_Bill.doc in case the first names of group members are John and Bill respectively.

**Grading Criteria**

**This assignment has 13 points (with weightage of 13% in your overall 100 points). Points will be awarded as follows:**

1. Functionality – **10 points**
2. Look and Feel of node creation, deletion and searching implementations – **3 points**

**Project Problem**

Write a java program to create an **AVL Search Tree (Balanced Tree)** that will make use of several JavaFX components, event handling, graphics, and Java Collections Framework to implement. GUI must contain buttons to perform following operations:

1. Insert - to insert a node (element) into the tree (3 Marks)
2. Delete - to delete a node from the tree (3 Marks)
3. Find- to search an element in the tree (1 Marks)
4. Print – to print the sorted list of elements (1 Marks)

Program should keep updating the following details at the bottom of the Frame:

1. height of the tree (1 Marks)
2. number of vertices (1 Marks)

**SUBMISSION:**

package avl;

import javafx.application.Application;

import javafx.event.ActionEvent;

import javafx.scene.control.Button;

import javafx.scene.layout.\*;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.control.\*;

import javafx.stage.Stage;

import javafx.scene.text.Font;

import javafx.scene.text.Text;

import javafx.scene.layout.Pane;

import javafx.scene.paint.Color;

import javafx.scene.shape.Circle;

import javafx.scene.shape.Line;

import javafx.scene.text.FontPosture;

import javafx.scene.text.FontWeight;

import java.util.ArrayList;

import javafx.scene.Scene;

import javafx.scene.input.KeyCode;

/\*\*

\*

\* @author Kaavya Jain

\*/

/\* Our application of AVL Tree can perform insertion, deletion of a node.

Also, find button reports if a particular node is present in the tree or not.

The traversal of the AVL Tree are also printed.

Buttons for inorder, preorder, postorder traversal is given.\*/

public class AVL extends Application{

@Override

public void start(Stage s)

{

AVLTree tree = new AVLTree();//create a tree

BorderPane bp = new BorderPane();//create a borderpane

AVLView view = new AVLView(tree); // Create a AVLView

bp.setCenter(view);//set the view in centre of borderpane

//set properties of stage

s.setTitle("AVL Tree Visualization");

s.setMinWidth(1020);

//create a vbox

VBox vbox = new VBox();

vbox.setPadding(new Insets(15, 15, 15, 15));

vbox.setSpacing(20);

vbox.setStyle("-fx-background-color: #081856;");

//vbox nodes

TextField value = new TextField("Type int");

value.setMaxWidth(130);

value.setStyle("-fx-text-fill: black");

value.setFont(new Font("ARIAL", 14));

value.setAlignment(Pos.CENTER);

Button insert = new Button("Insert");

insert.setPrefSize(100, 40);

insert.setStyle("-fx-font-size:14");

Button delete= new Button("Delete");

delete.setPrefSize(100, 40);

delete.setStyle("-fx-font-size:14");

Button find= new Button("Find\n");

find.setPrefSize(100, 40);

find.setStyle("-fx-font-size:14");

Label print = new Label("Print : \n");

print.setStyle("-fx-font-weight: bold");

print.setTextFill(Color.rgb(255,214,70));

print.setFont(new Font("ARIAL",15));

Button preorder= new Button("PreOrder");

preorder.setPrefSize(100, 40);

preorder.setStyle("-fx-font-size:14");

Button inorder= new Button("InOrder");

inorder.setPrefSize(100, 40);

inorder.setStyle("-fx-font-size:14");

Button postorder= new Button("PostOrder");

postorder.setPrefSize(100, 40);

postorder.setStyle("-fx-font-size:14");

Label lb = new Label("HAPPY TRAVERSING!");

lb.setStyle("-fx-font-weight: bold;");

lb.setTextFill(Color.rgb(255,214,70));

lb.setFont(new Font("ARIAL",15));

//add nodes in vbox

vbox.getChildren().addAll(value, insert, delete, find, print, preorder, inorder, postorder,lb);

vbox.setAlignment(Pos.CENTER);// set alignment of nodes in vbox in center

bp.setRight(vbox);// place vbox in right section of borderpane

bp.setBackground(new Background(new BackgroundFill(Color.BLACK, CornerRadii.EMPTY, Insets.EMPTY)));//black background of borderpane

//add bp in scene

Scene sc = new Scene(bp, 1000, 600);

// set the scene

s.setScene(sc);

s.show(); //show stage

//add delay for 1050 ms, after that, show welcome alert

try {Thread.sleep(1050);}

catch(InterruptedException ex){

Thread.currentThread().interrupt();

}

//welcome message alert

Alert a = new Alert(Alert.AlertType.INFORMATION);

//set title

a.setTitle("Welcome");

//set header text

a.setHeaderText("!!\* WELCOME \*!!");

// set content text

a.setContentText("This AVL Tree simulation is created by Kaavya Jain and Prakriti Agrawal for CSD203.\nIt can perform insertion, deletion, search and print the tree according to the chosen traversal.");

// show the dialog

a.show();

//lambda handlers for buttons

//insert function is performed when the insert button is pressed.

insert.setOnAction((ActionEvent e) -> {

if(value.getText().trim().equals("")){

invalidInput(value,"No key entered!");

}

else{

try {int key = Integer.parseInt(value.getText());

if (tree.search(key)) { // key is in the tree already

view.displayAVLTree();//display tree

view.setStatus(key + " is already in the tree");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

value.clear();//clear the textfield

} else {

tree.insert(key); // Insert a new key

view.displayAVLTree();

view.setStatus(key + " is inserted in the tree");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

value.clear();//clear the textfield

}

}

catch (NumberFormatException ex){

invalidInput(value,"Key must be an integer");

}

}

});

//now you can insert using enter key

//if ENTER key is pressed on the value textfield, then the input is inserted in the tree, if there is any.

value.setOnKeyPressed(keyEvent -> {

try {

if (keyEvent.getCode()==(KeyCode.ENTER)) {

int key = Integer.parseInt(value.getText());

if (tree.search(key)) { // key is in the tree already

view.displayAVLTree();

view.setStatus(key + " is already in the tree");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

value.clear();

} else {

tree.insert(key); // Insert a new key

view.displayAVLTree();

view.setStatus(key + " is inserted in the tree");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

value.clear();

}

}

}

catch (NumberFormatException ex){

invalidInput(value,"Key must be an integer");

}

});

//delete function is performed when the delete button is pressed.

delete.setOnAction((ActionEvent e) -> {

if(value.getText().trim().equals("")){

invalidInput(value,"No key entered!");

}

else{

try {

int key = Integer.parseInt(value.getText());

if (!tree.search(key)) { // key is not in the tree

view.displayAVLTree();

view.setStatus(key + " is not in the tree");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

value.clear();

} else {

tree.delete(key); // Delete a key

view.displayAVLTree();

view.setStatus(key + " is deleted from the tree");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

value.clear();

}

}

catch (NumberFormatException ex){

invalidInput(value,"Key must be an integer");

}

}

});

//find function is performed when the find button is pressed.

//an alert is popped to show the output

find.setOnAction((ActionEvent e) -> {

if(value.getText().trim().equals("")){

invalidInput(value,"No key entered!");

}

else{

try {

int key = Integer.parseInt(value.getText());

Alert fi = new Alert(Alert.AlertType.INFORMATION);

fi.setHeaderText("SEARCH : "+key);

fi.setTitle("Search Output");

if(!tree.search(key)){

view.displayAVLTree();

view.setStatus(key+" not found");

fi.setContentText(key+" is not present in the tree.");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

value.clear();

fi.show();

} else{

view.displayAVLTree();

view.setStatus(key+" found");

fi.setContentText(key+" is present in the tree.");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

value.clear();

fi.show();

}

}

catch(NumberFormatException ex){

invalidInput(value,"Key must be an integer");

}

}

});

//inorder traversal

//an alert is popped to show the output

inorder.setOnAction((ActionEvent e) ->{

view.displayAVLTree();

view.setStatus("Inorder Traversal printed.");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

Alert in = new Alert(Alert.AlertType.INFORMATION);

in.setHeaderText("INORDER:");

in.setTitle("Inorder Output");

ArrayList list = new ArrayList();//list to store the elements

in.setContentText((tree.inorder(tree.getRoot(), list)).toString());//convert the list to string and display as content

in.show();

});

//preorder traversal

//an alert is popped to show the output

preorder.setOnAction((ActionEvent e) ->{

view.displayAVLTree();

view.setStatus("Preorder Traversal printed.");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

Alert pre = new Alert(Alert.AlertType.INFORMATION);

pre.setHeaderText("PREORDER:");

pre.setTitle("Preorder Output");

ArrayList list = new ArrayList();//list to store the elements

pre.setContentText((tree.preorder(tree.getRoot(), list)).toString());//convert the list to string and display as content

pre.show();

});

//postorder traversal

//an alert is popped to show the output

postorder.setOnAction((ActionEvent e) -> {

view.displayAVLTree();

view.setStatus("Postorder Traversal printed.");

view.setHeight("Height: "+tree.height(tree.getRoot()));

view.setVertices("No. of vertices: "+tree.countNodes());

Alert post = new Alert(Alert.AlertType.INFORMATION);

post.setHeaderText("POSTORDER:");

post.setTitle("Postorder Output");

ArrayList list = new ArrayList();//list to store the elements

post.setContentText((tree.postorder(tree.getRoot(), list)).toString());//convert the list to string and display as content

post.show();

});

}

//invalid input method

private void invalidInput(TextField value, String a){

Alert alert = new Alert(Alert.AlertType.WARNING);

alert.setTitle("Warning");

alert.setHeaderText(a);

alert.setContentText("Please enter an integer in the input box and try again.");

value.requestFocus();

alert.showAndWait();

value.clear();

}

public static void main(String args[])

{

// launch the application

Application.launch(args);

}

}

//AVLView to make the tree and set the status and height and vertices

class AVLView extends Pane {

private AVLTree tree = new AVLTree();

public double radius = 25; // Tree node radius

private final double vGap = 90; // Gap between two levels in a tree

AVLView(AVLTree tree) {

this.tree = tree;

setStatus("Tree is empty");

setHeight("Height: -1");

setVertices("No. of vertices: 0");

}

public final void setStatus(String msg) {

Text t = new Text(20, 20, msg);

t.setFont(Font.font("ARIAL", FontWeight.BOLD,FontPosture.REGULAR, 15));

t.setFill(Color.rgb(255,246,241));

getChildren().add(t);

}

public final void setHeight(String h){

Text t = new Text(370, 20, h);

t.setFont(Font.font("ARIAL", FontWeight.BOLD,FontPosture.REGULAR, 15));

t.setFill(Color.rgb(255,246,241));

getChildren().add(t);

}

public final void setVertices(String v){

Text t = new Text(680, 20, v);

t.setFont(Font.font("ARIAL", FontWeight.BOLD,FontPosture.REGULAR, 15));

t.setFill(Color.rgb(255,246,241));

getChildren().add(t);

}

public void displayAVLTree(){

this.getChildren().clear();

if(tree.getRoot() != null){

displayAVLTree(tree.getRoot(), getWidth() / 2, vGap, getWidth() / 4);

}

}

//to display the lines and circles in tree accordingly

protected void displayAVLTree(AVLTree.Node root, double x, double y, double hGap){

if(root.left != null){

Line l = new Line(x - hGap, y + vGap, x, y);

l.setStroke(Color.rgb(8,91,185));

l.setStrokeWidth(5.0f);

getChildren().add(l);

displayAVLTree(root.left, x - hGap, y + vGap, hGap / 2);

}

if (root.right != null){

Line l = new Line(x + hGap, y + vGap, x, y);

l.setStroke(Color.rgb(8,91,185));

l.setStrokeWidth(5.0f);

getChildren().add(l);

displayAVLTree(root.right, x + hGap, y + vGap, hGap / 2);

}

Circle circle = new Circle(x, y, radius);

circle.setFill(Color.rgb(255,214,70));

circle.setStroke(Color.rgb(8,91,185));

circle.setStrokeWidth(4.0f);

Text node = new Text(x - 4, y + 4, root.key + "");

node.setStyle("-fx-font-size: 15");

getChildren().addAll(circle, node);

}

}

//class containing all functions performed on tree

class AVLTree {

public Node root;

public static class Node {

public int key;

public int balance;

public int height;

public Node left;

public Node right;

public Node parent;

Node(int key, Node parent) {

this.key = key;

this.parent = parent;

}

}

public boolean insert(int key) {

if (root == null) {

root = new Node(key, null);

return true;

}

Node n = root;

while (true) {

if (n.key == key)

return false;

Node parent = n;

boolean goLeft = n.key > key;

n = goLeft ? n.left : n.right;

if (n == null) {

if (goLeft) {

parent.left = new Node(key, parent);

} else {

parent.right = new Node(key, parent);

}

rebalance(parent);

break;

}

}

return true;

}

private void delete(Node node) {

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

if (node.parent == null) {

root = null;

} else {

Node parent = node.parent;

if (parent.left == node) {

parent.left = null;

} else {

parent.right = null;

}

rebalance(parent);

}

return;

}

if (node.left != null) {

Node child = node.left;

while (child.right != null) child = child.right;

node.key = child.key;

delete(child);

} else {

Node child = node.right;

while (child.left != null) child = child.left;

node.key = child.key;

delete(child);

}

}

public void delete(int delKey) {

if (root == null)

return;

Node child = root;

while (child != null) {

Node node = child;

child = delKey >= node.key ? node.right : node.left;

if (delKey == node.key) {

delete(node);

return;

}

}

}

private void rebalance(Node n) {

setBalance(n);

if (n.balance == -2) {

if (height(n.left.left) >= height(n.left.right))

n = rotateRight(n);

else

n = rotateLeftThenRight(n);

} else if (n.balance == 2) {

if (height(n.right.right) >= height(n.right.left))

n = rotateLeft(n);

else

n = rotateRightThenLeft(n);

}

if (n.parent != null) {

rebalance(n.parent);

} else {

root = n;

}

}

private Node rotateLeft(Node a) {

Node b = a.right;

b.parent = a.parent;

a.right = b.left;

if (a.right != null)

a.right.parent = a;

b.left = a;

a.parent = b;

if (b.parent != null) {

if (b.parent.right == a) {

b.parent.right = b;

} else {

b.parent.left = b;

}

}

setBalance(a, b);

return b;

}

private Node rotateRight(Node a) {

Node b = a.left;

b.parent = a.parent;

a.left = b.right;

if (a.left != null)

a.left.parent = a;

b.right = a;

a.parent = b;

if (b.parent != null) {

if (b.parent.right == a) {

b.parent.right = b;

} else {

b.parent.left = b;

}

}

setBalance(a, b);

return b;

}

private Node rotateLeftThenRight(Node n) {

n.left = rotateLeft(n.left);

return rotateRight(n);

}

private Node rotateRightThenLeft(Node n) {

n.right = rotateRight(n.right);

return rotateLeft(n);

}

public int height(Node n) {

if (n == null)

return -1;

return n.height;

}

private void setBalance(Node... nodes) {

for (Node n : nodes) {

reheight(n);

n.balance = height(n.right) - height(n.left);

}

}

public Node getRoot() {

return root;

}

public boolean search(int val)

{

return search(root, val);

}

private boolean search(Node r, int val)

{

boolean found = false;

while ((r != null) && !found)

{

int rval = r.key;

if (val < rval)

r = r.left;

else if (val > rval)

r = r.right;

else

{

found = true;

break;

}

found = search(r, val);

}

return found;

}

public int countNodes()

{

return countNodes(root);

}

private int countNodes(Node r)

{

if (r == null)

return 0;

else

{

int l = 1;

l += countNodes(r.left);

l += countNodes(r.right);

return l;

}

}

private void reheight(Node node) {

if (node != null) {

node.height = 1 + Math.max(height(node.left), height(node.right));

}

}

public ArrayList inorder(Node r,ArrayList l){

if (r != null) {

inorder(r.left,l);

l.add(r.key);

inorder(r.right,l);

}

return l;

}

public ArrayList preorder(Node r,ArrayList l){

if (r != null) {

l.add(r.key);

inorder(r.left,l);

inorder(r.right,l);

}

return l;

}

public ArrayList postorder(Node r,ArrayList l){

if (r != null) {

inorder(r.left,l);

inorder(r.right,l);

l.add(r.key);

}

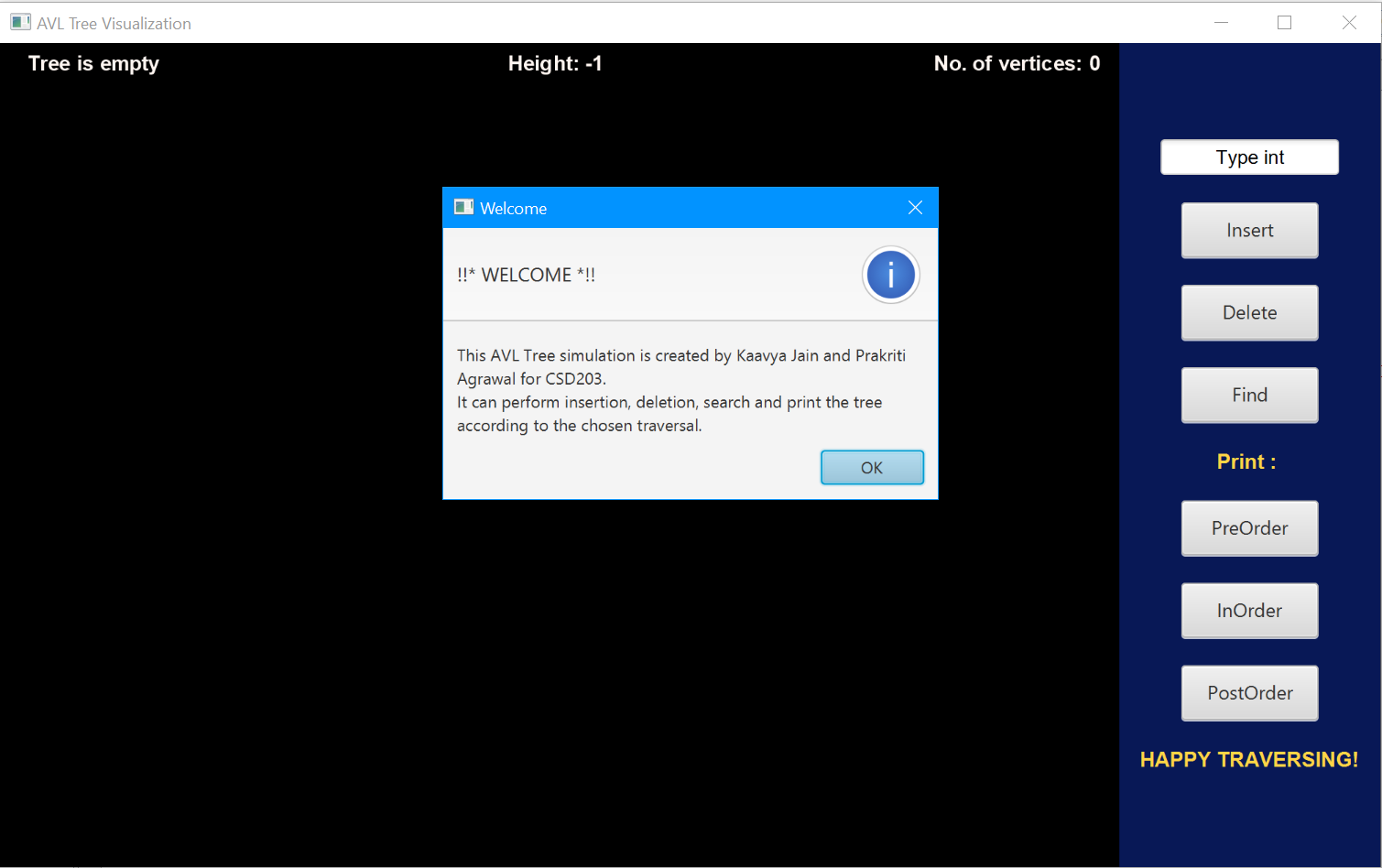
return l;

}

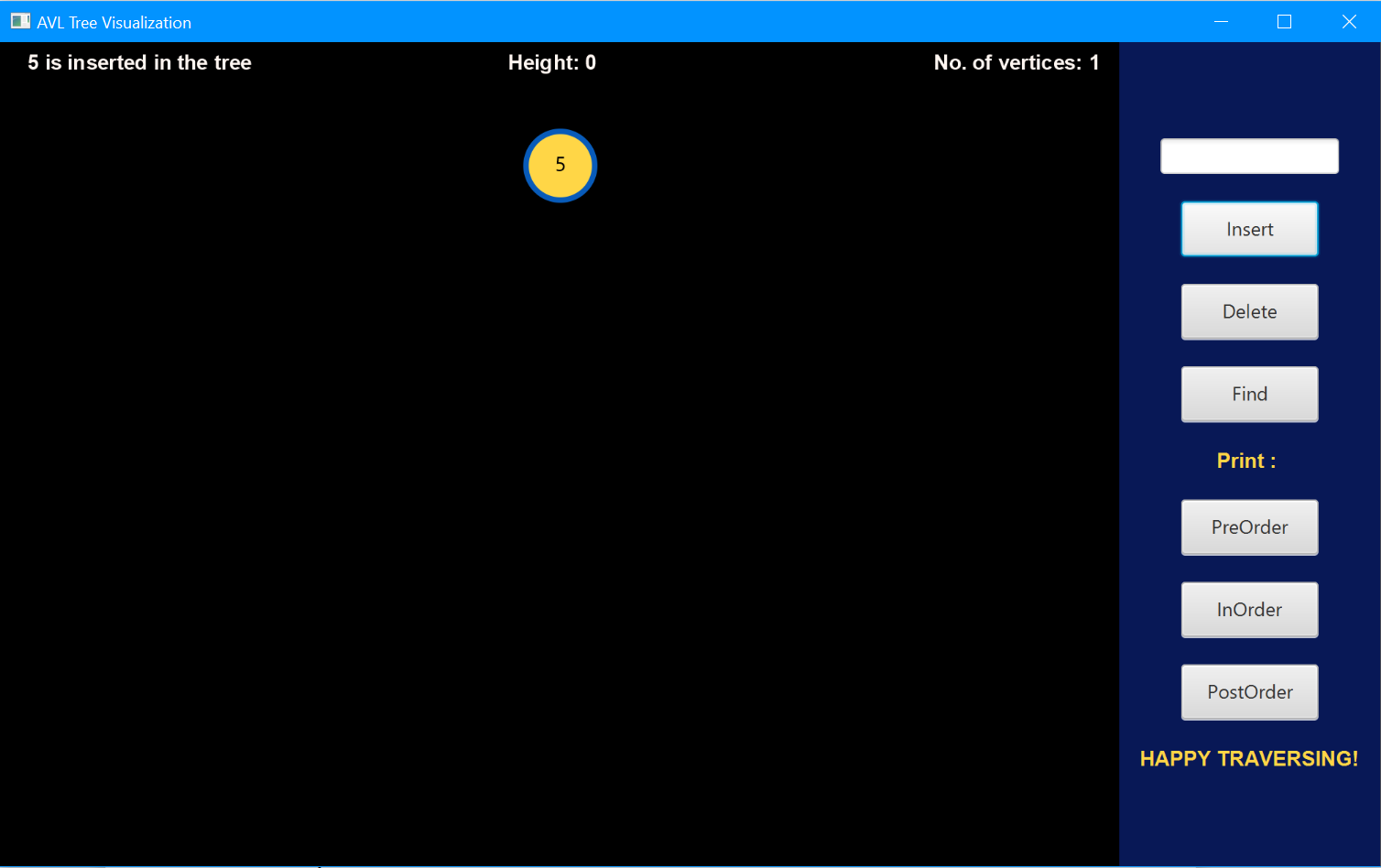
}

**OUTPUT PICTURES:**

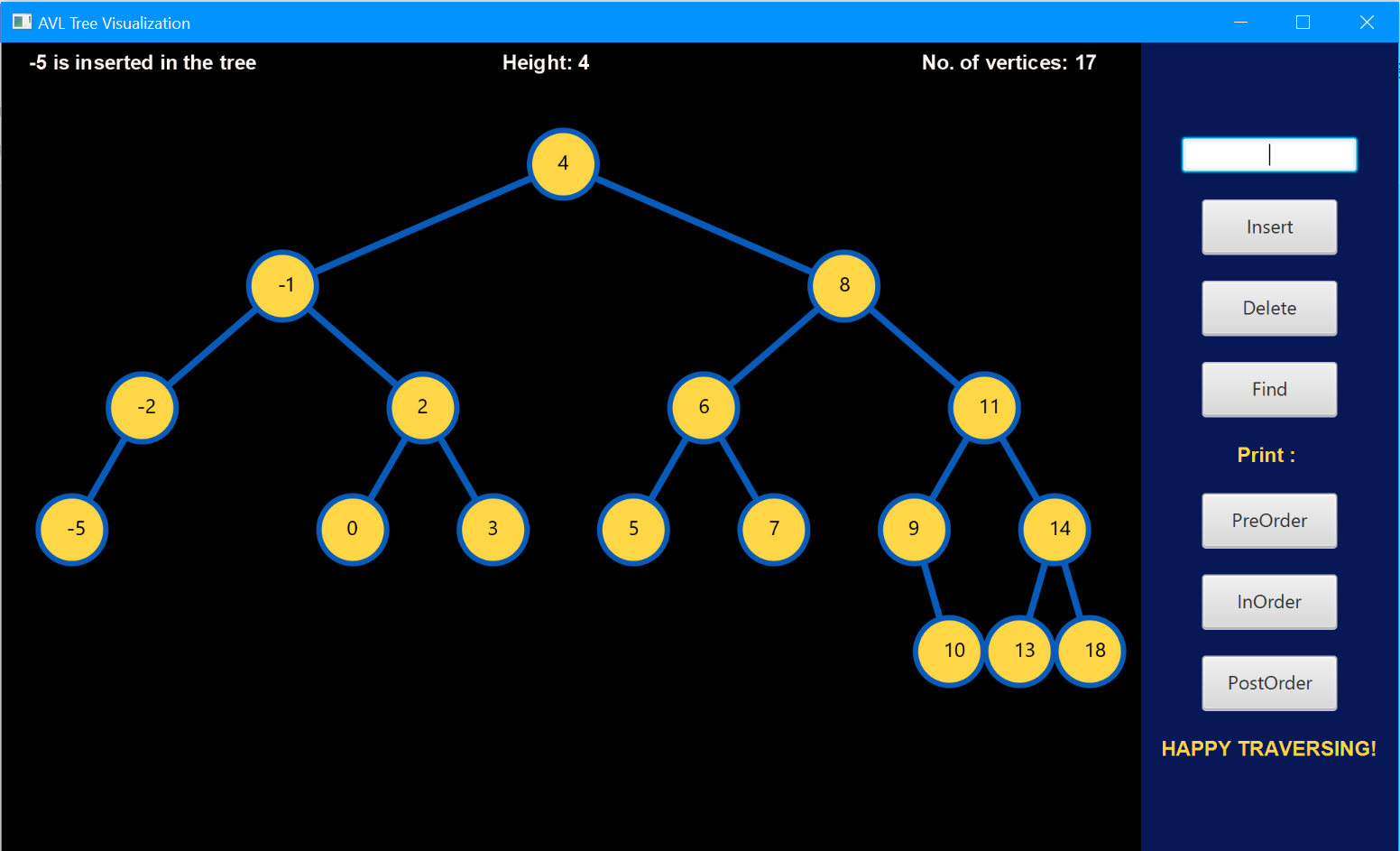
1. **Welcome message**



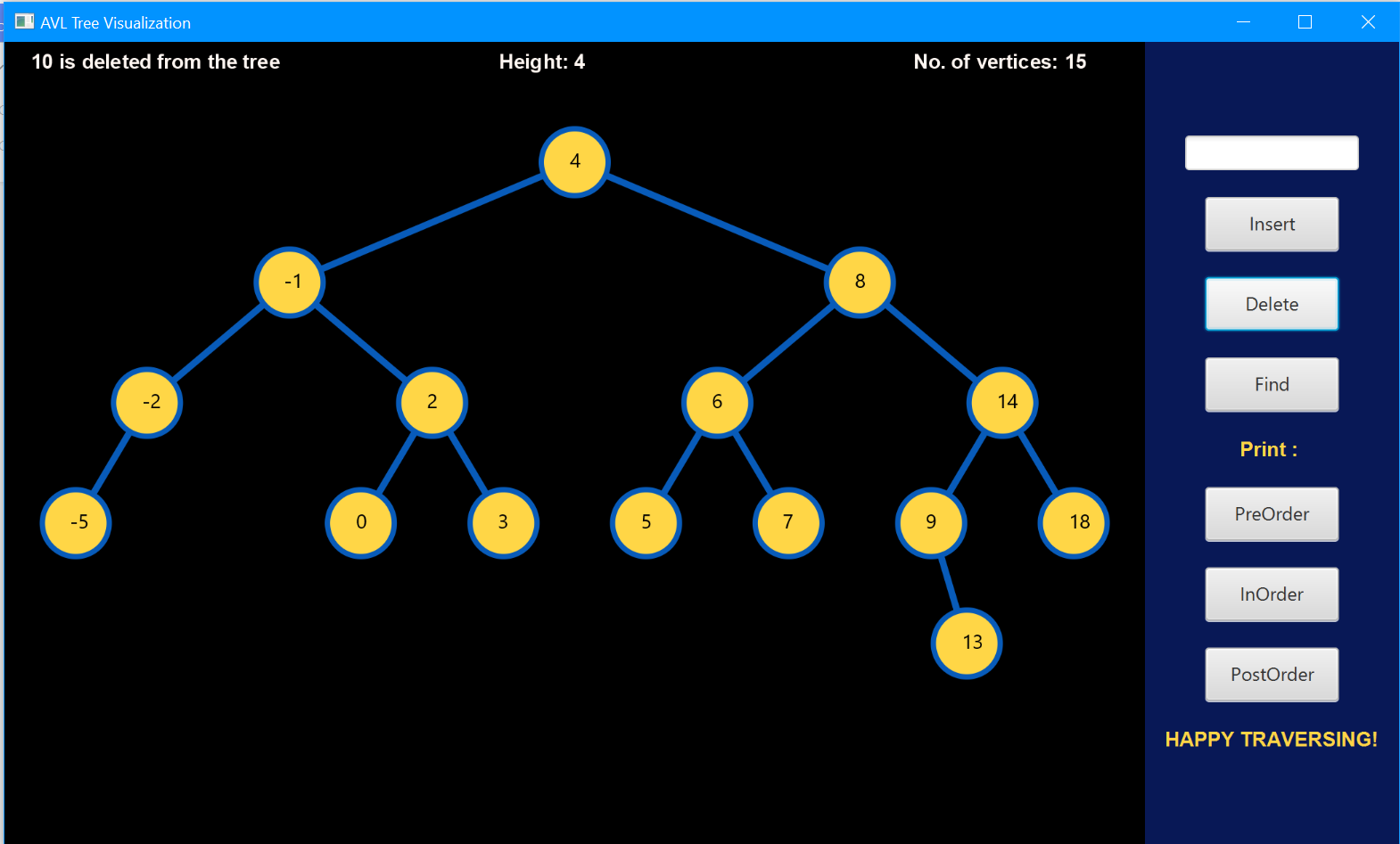
1. **Insert a node in the tree**



1. **A tree**

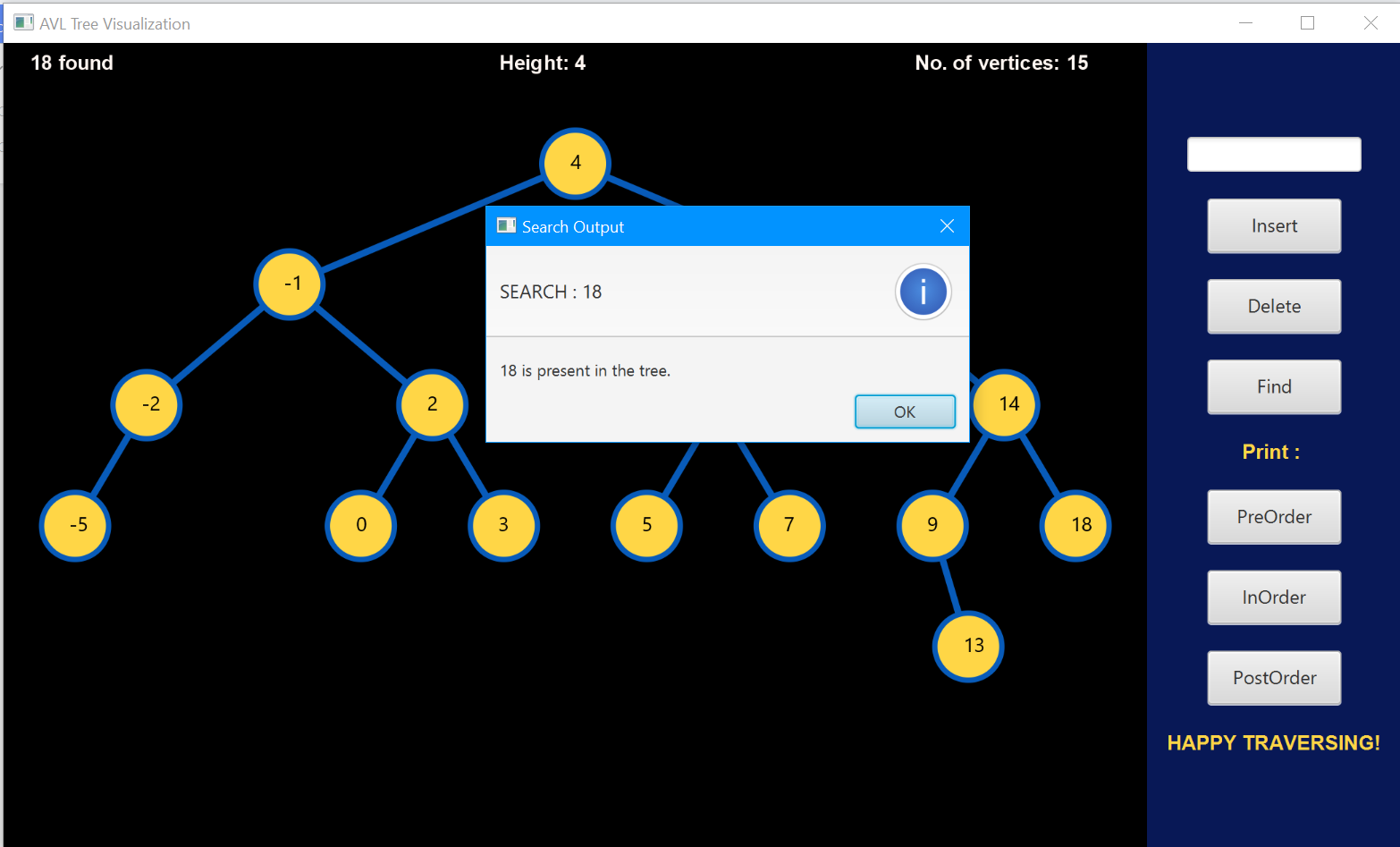


1. **Delete a node from the tree**

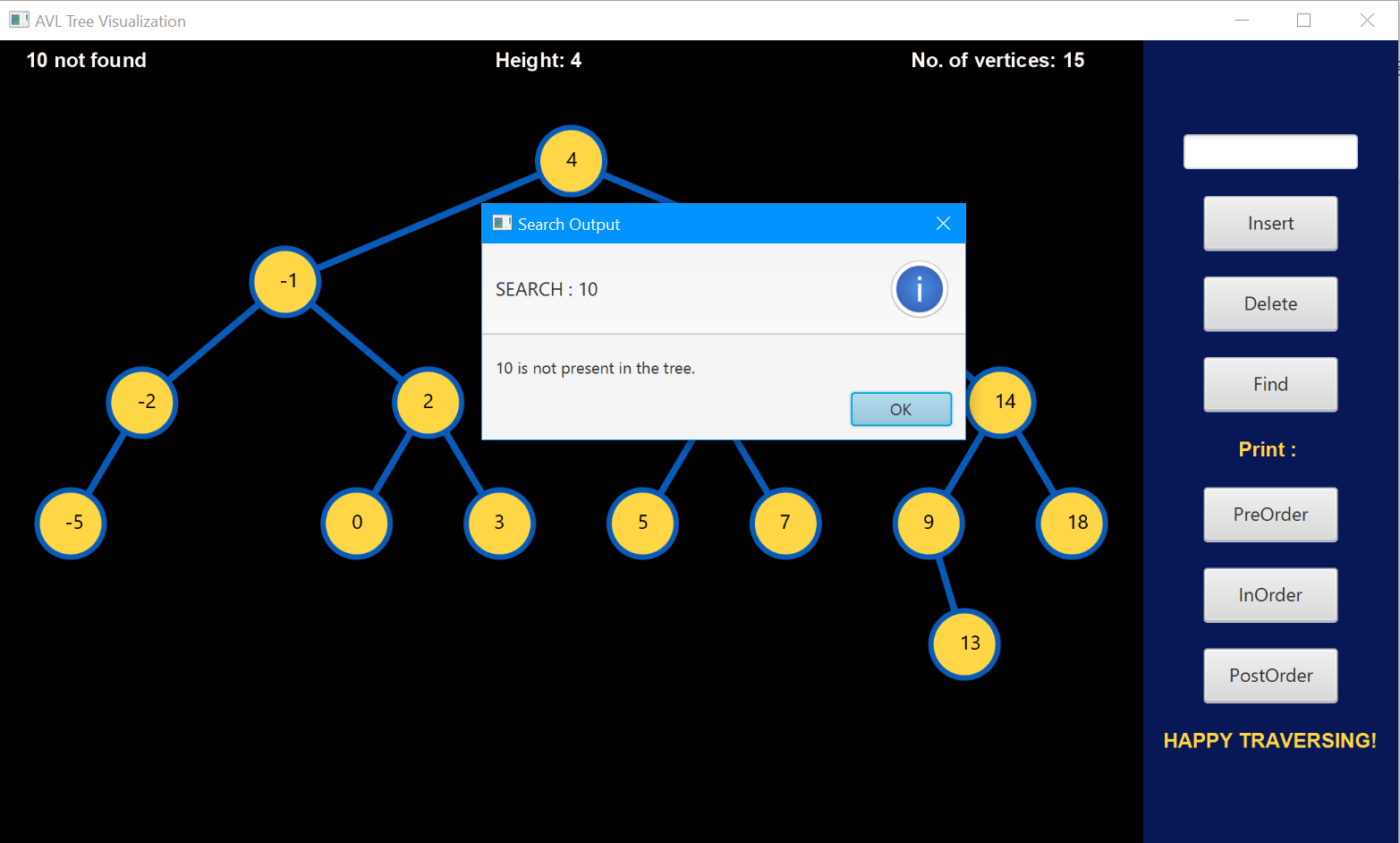


1. **Find a particular node in the tree**

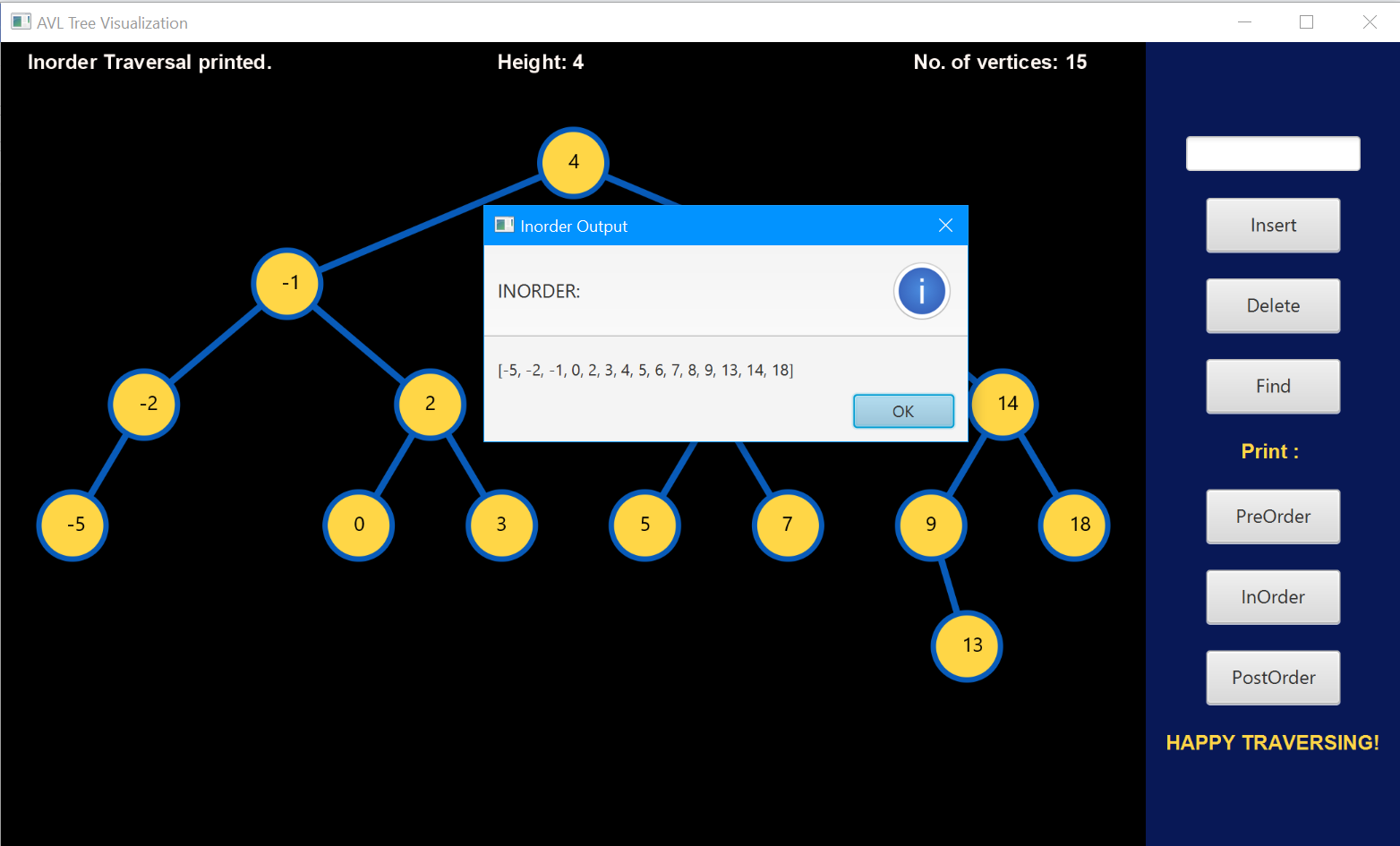
**Found:**



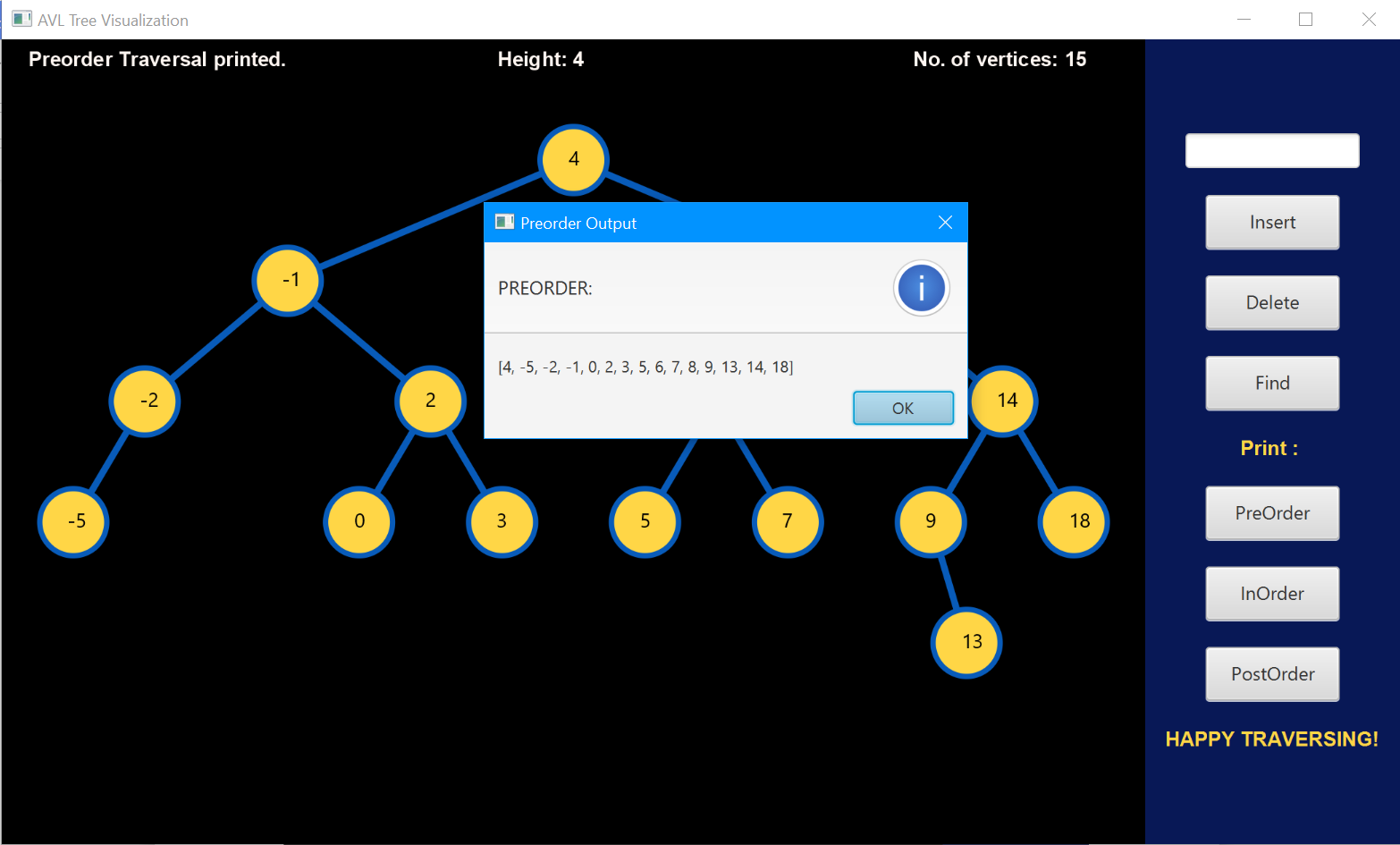
**Not found:**



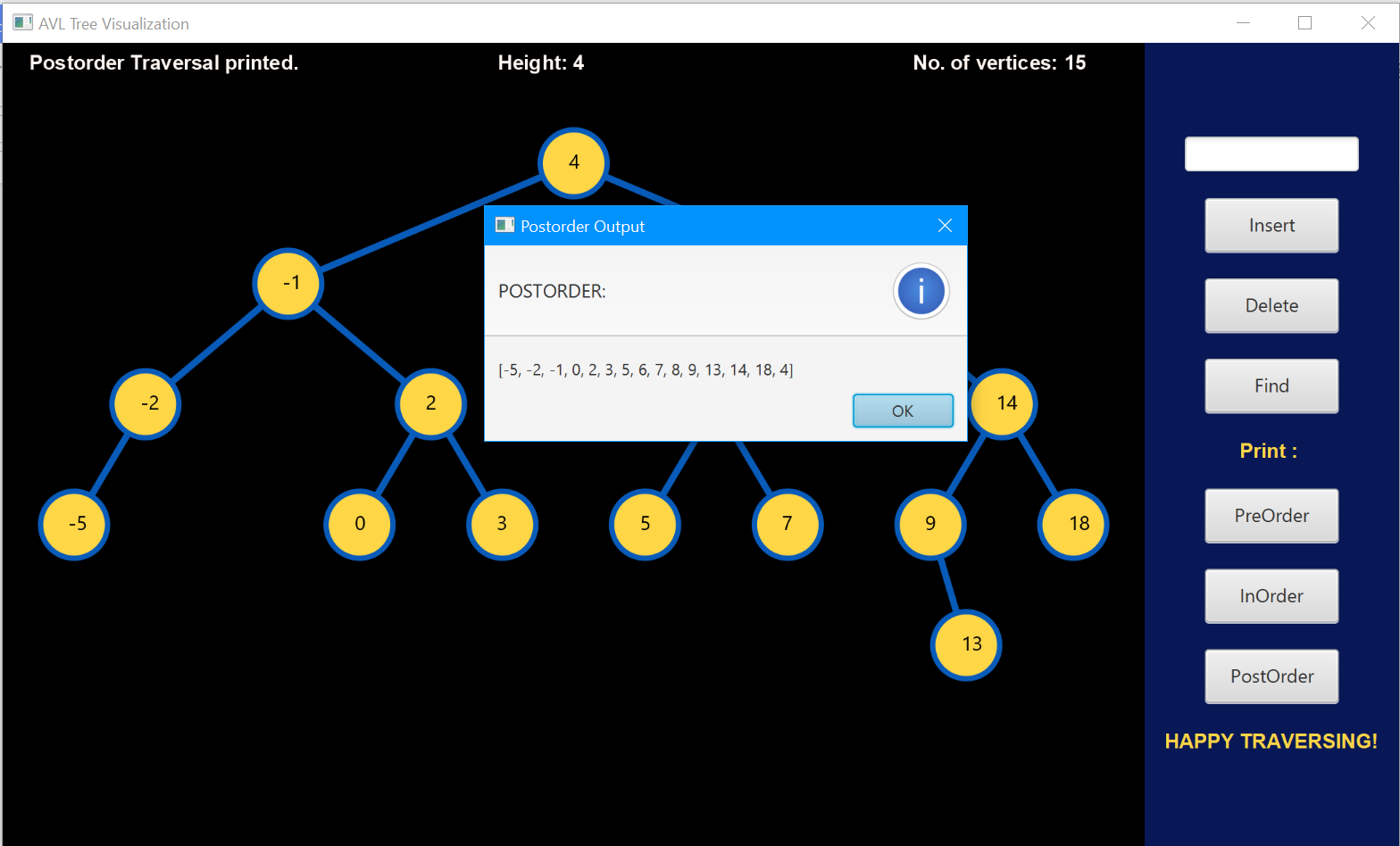
1. **Show the inorder traversal of the tree**



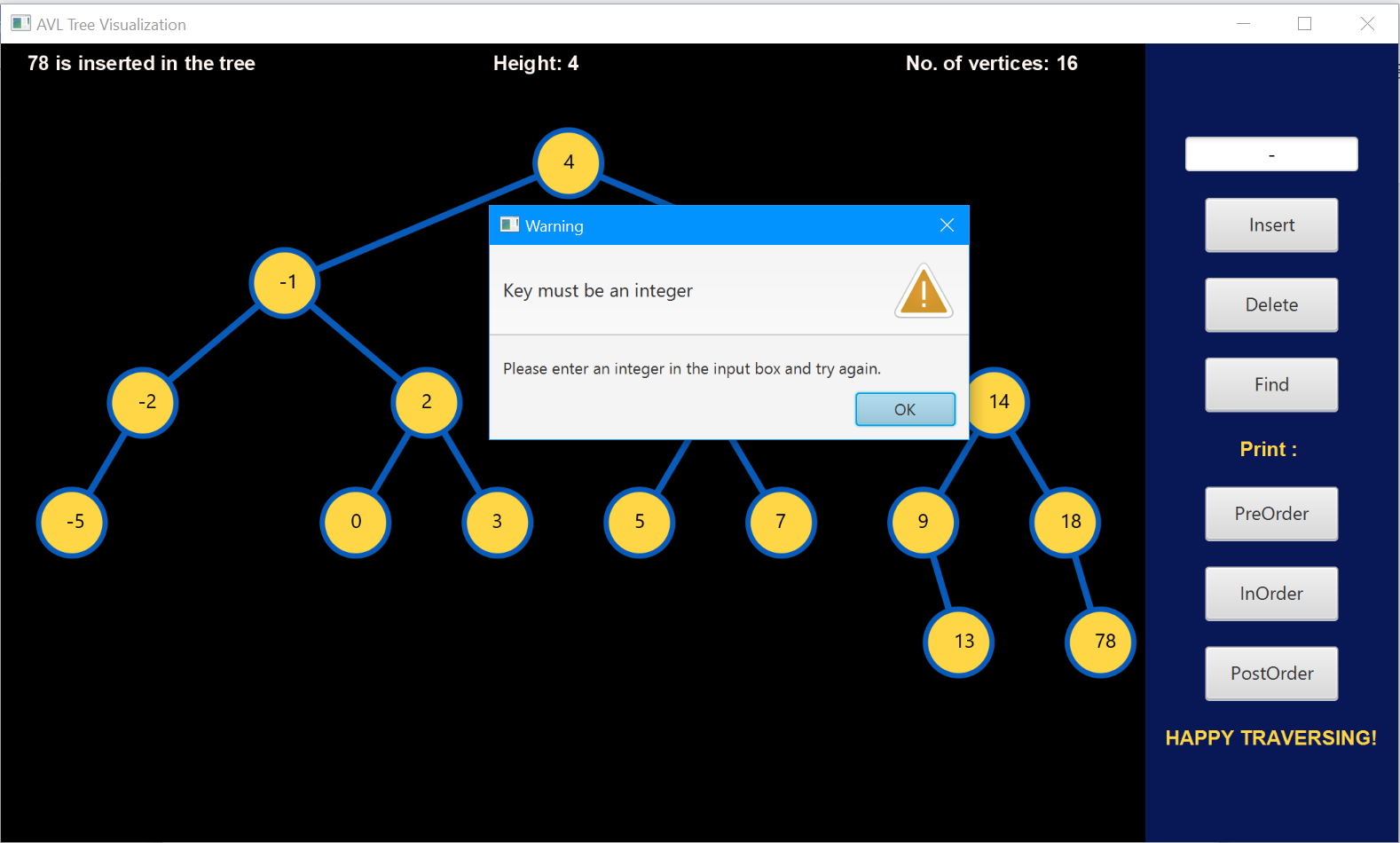
1. **Show the preorder traversal of the tree**



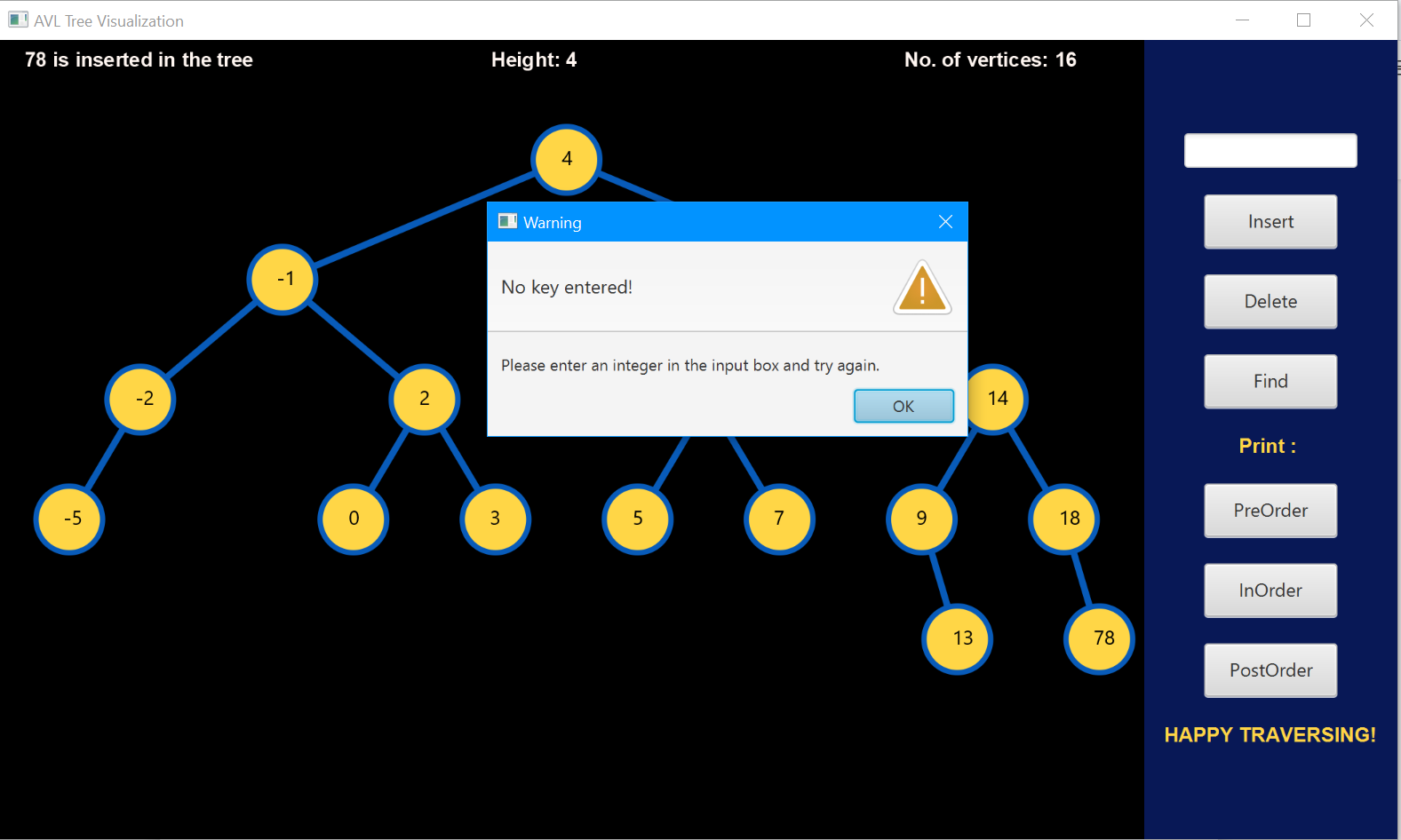
1. **Show the postorder traversal of the tree**



1. **Warning alert when wrong input is given**



1. **Warning alert when no input is given**



**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**