```java
import java.util.Scanner;
public class Main
{
    public static void main(String[] args){
        Scanner keyboard = new Scanner(System.in);
        StuData sd = new StuData();


        displayMenu();
        int option = keyboard.nextInt();

        while(option != 8){
            if (option == 1){
                String[] parms = getParm("stuData").split("\\s+");
                sd.insert(Integer.valueOf(parms[0].trim()),
                          parms[1],
                          Integer.valueOf(parms[2].trim()),
                          Double.valueOf(parms[3].trim()));
            }
            else if (option == 2){sd.delete(Integer.valueOf(getParm("id").t
            else if (option == 3){sd.search(Integer.valueOf(getParm("id").t
            else if (option == 4){
                String[] parms = getParm("stuData").split("\\s+");
                sd.modify(Integer.valueOf(parms[0].trim()),
                          parms[1],
                          Integer.valueOf(parms[2].trim()),
                          Double.valueOf(parms[3].trim()));
            }
            else if (option == 5){sd.display();}
            else if (option == 6){sd.upload(getParm("path"));}
            else if (option == 6){sd.download(getParm("path"));}
            else{System.out.println("Invalid Option");}

            displayMenu();
            option = keyboard.nextInt();
        }


    }
    private static void displayMenu(){
        System.out.println("1 - Add a student\n" +
                           "2 - Delete a student\n" +
                           "3 - Search for a student\n" +
                           "4 - Modify a student\n" +
                           "5 - Display all students\n" +
                           "6 - Upload a data file\n" +
                           "7 - Download the data file\n" +
                           "8 - Exit\n" +
                                "Choose an option - ");
    }


    private static String getParm(String parm){
        Scanner keyboard = new Scanner(System.in);
        String toReturn = "";
        if(parm == "path"){
            System.out.println("Enter the file path: ");
            toReturn = keyboard.nextLine();
        }
        else if (parm == "stuData"){
            System.out.println("Enter student id, name, age, and gpa sepera
            toReturn = keyboard.nextLine();
        }
        else if (parm == "id"){
            System.out.println("Enter student ID #: ");
            toReturn = String.valueOf(keyboard.nextInt());
        }
        else{
            System.out.println("Weird Error, you better contact IT");
        }
        return toReturn;
    }
}
```

```java
import java.io.FileReader;
import java.io.File;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class StuData
{
    private class node{
        int id;
        String lastName;
        int age;
        double gpa;
        node left;
        node right;

        private node(int id, String lastName, int age, double gpa, node lef
            this.id = id;
            this.lastName = lastName;
            this.age = age;
            this.gpa = gpa;
            this.left = left;
            this.right = right;
        }
    }

    node root;

    public StuData()
    {
        root = null;
    }

    public boolean insert(int id, String lastName, int age, double gpa){
        if(age < 10 || age > 100){
            System.out.println("Invalid age");
            return false;
        }
        else if(id < 1 || id > 10000){
            System.out.println("Invalid ID");
            return false;
        }
        else if(gpa < 0.0 || gpa > 4.0){
            System.out.println("Invalid GPA");
            return false;
        }
        else if(root == null){//special case, empty tree
```

```java
            root = new node(id, lastName, age, gpa, null, null);
            return true;
        }
        else{
            node temp = root;
            node prev = null;
            while(temp != null){
                if(id == temp.id){ //student already exists
                    System.out.println("Student already exists");
                    return false;
                }
                else if(id < temp.id){ //go left
                    prev = temp;
                    temp = temp.left;
                }
                else{ //go right
                    prev = temp;
                    temp = temp.right;
                }
            }
            if(id < prev.id){
                prev.left = new node(id, lastName, age, gpa, null, null);
                return true;
            }
            else{
                prev.right = new node(id, lastName, age, gpa, null, null);
                return true;
            }
        }
    }

    public boolean delete(int id){
        node prev = getPrev(id);
        if(prev == null){//delete root
            delete(root, null);
            return true;
        }
        else if(prev.left == null && prev.right == null){// id didn't exist
            return false;
        }
        else if(prev.left != null && prev.left.id == id){// delete left noc
            delete(prev.left, prev);
            return true;
        }
        else if(prev.right != null && prev.right.id == id){ //delete right
            delete(prev.right, prev);
            return true;
        }
        else{
            return false;
        }
    }

    public boolean search(int id){
        node prev = getPrev(id);
        if(prev == null){//print root
            print(root);
            return true;
        }
        else if(prev.left == null && prev.right == null){// id didn't exist
            System.out.println("Student not found");
            return false;
        }
        else if(prev.left != null && prev.left.id == id){// print left node
            print(prev.left);
            return true;
        }
        else if(prev.right != null && prev.right.id == id){ //print right r
            print(prev.right);
            return true;
        }
        else{
            System.out.println("Student not found");
            return false;
        }
    }

    public boolean modify(int id, String lastName, int age, double gpa){
        node prev = getPrev(id);
        if(prev == null){//modify root
            modify(root, lastName, age, gpa);
            return true;
        }
        else if(prev.left == null && prev.right == null){// id didn't exist
            System.out.println("Student not found");
            return false;
        }
        else if(prev.left != null && prev.left.id == id){// print left node
            modify(prev.left, lastName, age, gpa);
            return true;
        }
        else if(prev.right != null && prev.right.id == id){ //print right r
            modify(prev.right, lastName, age, gpa);
            return true;
        }
        else{
            System.out.println("Student not found");
            return false;
        }
    }

    public void display(){
        display(root);
    }

    public void upload(String path){
         System.out.println("Upload");
         try{  //grab inputs from file
                FileReader fileReader = new FileReader(new File(path));
                BufferedReader br = new BufferedReader(fileReader);

                System.out.println("Setup success");

                root = null; // delete current tree
                String line;
                while((line=br.readLine())!=null){
                    //do sstuff
                    System.out.println("Lineread");
                    int id = Integer.valueOf(line.substring(4,9).trim());
                    System.out.println("ID: " + id);
                    int age = Integer.valueOf(line.substring(17,19).trim())
                    System.out.println("AGE: " + age);
                    double gpa = Double.valueOf(line.substring(26,32));
                    System.out.println("GPA: " + gpa);
                    String lastName = line.substring(44);
                }
            }
            catch (Exception e) {
            System.out.println(e.getClass());
            }
    }
    public void download(String path){
        try{
            // Create file
            FileWriter fstream = new FileWriter(path);
            BufferedWriter out = new BufferedWriter(fstream);
            out.write(downloadFormat(root));
            //Close the output stream
            out.close();
        }catch (Exception e){//Catch exception if any
            System.err.println("Error: " + e.getMessage());
        }

    }

    private String downloadFormat(node temp){
        if(temp == null){
            return "";
        }
        else{
            return format(temp) + downloadFormat(temp.left) + downloadForma
        }
    }

    private void display(node temp){
        if(temp == null){
            return;
        }
        else{
            display(temp.left);
            print(temp);
            display(temp.right);
        }
    }

    private void modify(node temp, String lastName, int age, double gpa){
        temp.lastName = lastName;
        temp.age = age;
        temp.gpa = gpa;
    }

    private String format(node temp){
        return String.format("ID: %5d  AGE: %3d  GPA: %1.3f  LASTNAME: %S %




    }

    private void print(node temp){
        System.out.print(format(temp));
    }

    private boolean delete(node temp, node prev){
        if(temp.left == null){ //no left subtree
            System.out.println("No left subtree");
            if (temp == root){ //deleting root
                root = temp.right;
            }
            else if (temp == prev.left){
                prev.left = temp.right;
            }
            else{
                prev.left = temp.right;
            }
        }
        else if(temp.right == null){//no right subtree
            System.out.println("No right subtree");
            if (temp == root){ //deleting root
                root = temp.left;
            }
            else if (temp == prev.right){
                prev.right = temp.left;
            }
            else{
                prev.left = temp.left;
            }
        }
        else{//left and right subtrees present
            System.out.println("both subtree");
            node p = temp.left;
            node q = temp;
            while(p.right != null){ //get max of left subtree
                q = p;
                p = p.right;
            }
            temp.id = p.id;
            temp.age = p.age;
            temp.gpa = p.gpa;
            temp.lastName = p.lastName;
            if(q == p.left){
                q.left = p.left;
            }
            else{
                q.right = p.left;
            }
        }
        return true;
    }

    private node getPrev(int id){
        node temp = root;
        node prev = null;
        while(temp != null){
            if(id == temp.id){ // found it
                break;
            }
            else if(id < temp.id){ //go left
                prev = temp;
                temp = temp.left;
            }
            else{ //go right
                prev = temp;
                temp = temp.right;
            }
        }
        return prev;
    }

}

```

```java
import java.util.Scanner;
import java.util.Random;
public class Main
{
    public static void main(String[] args){
        Scanner keyboard = new Scanner(System.in);

        displayMenu();
        int option = keyboard.nextInt();
        bst bs = null;
        long seed;

        while(option != 4){
            if (option == 1){
                System.out.println("Enter a the desired number of nodes");
                int n = keyboard.nextInt();
                System.out.println("Enter an RNG seed");
                seed = keyboard.nextLong();
                bs = new bst(n,seed);
            }
            else if (option == 2){
                if(bs != null){
                    System.out.println("Height: " + bs.height());
                    System.out.println("Count: " + bs.count());
                }
                else{
                    System.out.println("You havn't initialized your tree ye
                }
            }
            else if (option == 3){
                if(bs != null){
                    System.out.println("Enter an RNG seed");
                    seed = keyboard.nextLong();

                    Random rand = new Random(seed);

                    long start = System.currentTimeMillis();
                    System.out.println("Start: " + start);
                    for(int i=0;i<1000000;i++){
                        bs.search((rand.nextInt(1000000000)+1));
                    }
                    long stop = System.currentTimeMillis();
                    System.out.println("Stop:  " + stop);
                    double time = Double.valueOf(stop-start);
                    System.out.println("Total time required: " + time);
                }
                else{
                    System.out.println("You havn't initialized your tree ye
                }
            }
            else{System.out.println("Invalid Option");}

            displayMenu();
            option = keyboard.nextInt();
        }


    }
    private static void displayMenu(){
        System.out.println("1 - Create BST\n" +
                           "2 - Display height & # of nodes\n" +
                           "3 - Perform 10^6 random searches\n" +
                           "4 - Exit\n" +
                           "Choose an option - ");
    }
}
```

```java
import java.util.Random;
public class bst
{
    private class node{
        int data;
        node left;
        node right;
        node parent;

        private node(int data, node left, node right, node parent){
            this.data = data;
            this.left = left;
            this.right = right;
            this.parent = parent;
        }
    }

    node root;
    //needs: insert, constructor using rng, height, count nodes

    public bst(){
        this.root = null;
    }
    public bst(int n, long seed){
        this.root = null;
        Random rand = new Random(seed);
        for(int i=0;i<n;i++){
            this.insert((rand.nextInt(1000000000)+1));
        }
    }

    public int count(){
        return count(root);
    }

    public int height(){
        return height(root);
    }

    public boolean insert(int data){
        if(root == null){ //special case, empty tree
            root = new node(data, null, null, null);
            return true;
        }
        else{
            node temp = root;
            node prev = null;
            while(temp != null){
                if(temp.data == data){
                    return false;
                }
                else if(data > temp.data){ //go right
                    prev = temp;
                    temp = temp.right;
                }
                else{ //go left
                    prev = temp;
                    temp = temp.left;
                }
            }
            if(data > prev.data){ //create right
                prev.right = new node(data, null, null, prev);
                return true;
            }
            else{ //create left
                prev.left = new node(data, null, null, prev);
                return true;
            }
        }
    }

    public boolean search(int data){
        node temp = root;
        while(temp != null){
            if(temp.data == data){
                    return true;
            }
            else if(data > temp.data){ //go right
                    temp = temp.right;
            }
            else{ //go left
                temp = temp.left;
            }
        }
        return false;
    }

    private int count(node temp){
        if(temp == null){
            return 0;
        }
        else{
            return 1 + count(temp.left) + count(temp.right);
        }
    }

    private int height(node temp){
        if(temp == null){
            return 0;
        }
        int lheight = height(temp.left);
        int rheight = height(temp.right);
        if(lheight > rheight){
            return 1 + lheight;
        }
        else{
            return 1 + rheight;
        }
    }
}

```

```java
import java.util.Scanner;
import java.util.Random;
public class Main
{
    public static void main(String[] args){
        Scanner keyboard = new Scanner(System.in);

        displayMenu();
        int option = keyboard.nextInt();
        avl bs = null;
        long seed;

        while(option != 4){
            if (option == 1){
                System.out.println("Enter a the desired number of nodes");
                int n = keyboard.nextInt();
                System.out.println("Enter an RNG seed");
                seed = keyboard.nextLong();
                bs = new avl(n,seed);
            }
            else if (option == 2){
                if(bs != null){
                    System.out.println("Height: " + bs.height());
                    System.out.println("Count: " + bs.count());
                }
                else{
                    System.out.println("You havn't initialized your tree ye
                }
            }
            else if (option == 3){
                if(bs != null){
                    System.out.println("Enter an RNG seed");
                    seed = keyboard.nextLong();

                    Random rand = new Random(seed);

                    long start = System.currentTimeMillis();
                    System.out.println("Start: " + start);
                    for(int i=0;i<1000000;i++){
                        bs.search((rand.nextInt(1000000000)+1));
                    }
                    long stop = System.currentTimeMillis();
                    System.out.println("Stop:  " + stop);
                    double time = Double.valueOf(stop-start);
                    System.out.println("Total time required: " + time);
                }
                else{
                    System.out.println("You havn't initialized your tree ye
                }
            }
            else{System.out.println("Invalid Option");}

            displayMenu();
            option = keyboard.nextInt();
        }


    }
    private static void displayMenu(){
        System.out.println("1 - Create AVL tree\n" +
                           "2 - Display height & # of nodes\n" +
                           "3 - Perform 10^6 random searches\n" +
                           "4 - Exit\n" +
                           "Choose an option - ");
    }
}


```

```java
import java.util.Random;
public class avl
{
    private class node{
        int data;
        int height;
        node left;
        node right;
        node parent;

        private node(int data, int height, node left, node right, node pare
            this.data = data;
            this.height = height;
            this.left = left;
            this.right = right;
            this.parent = parent;
        }
    }

    node root;
    //needs: insert, constructor using rng, height, count nodes

    public avl(){
        this.root = null;
    }
    public avl(int n, long seed){
        this.root = null;
        Random rand = new Random(seed);
        for(int i=0;i<n;i++){
            //System.out.println("Insert # " + i);
            this.insert((rand.nextInt(1000000000)+1));
        }
    }

    public int count(){
        return count(root);
    }

    public int height(){
        return root.height;
    }

    public boolean insert(int data){
        if(root == null){ //special case, empty tree
            root = new node(data, 1, null, null, null);
            return true;
        }
        else{
            node temp = root;
            node prev = null;
            //System.out.println("About to search for: " + data);
            while(temp != null){
                if(temp.data == data){
                    return false;
                }
                else if(data > temp.data){ //go right
                    prev = temp;
                    temp = temp.right;
                }
                else{ //go left
                    prev = temp;
                    temp = temp.left;
                }
            }
            System.out.println("Got temp");
            if(data > prev.data){ //create right
                prev.right = new node(data, 1, null, null, prev);
                updateHeight(prev.right);
                return true;
            }
            else{ //create left
                prev.left = new node(data, 1, null, null, prev);
                updateHeight(prev.left);
                return true;
            }

        }
    }

    public boolean search(int data){
        node temp = root;
        while(temp != null){
            if(temp.data == data){
                    return true;
            }
            else if(data > temp.data){ //go right
                    temp = temp.right;
            }
            else{ //go left
                temp = temp.left;
            }
        }
        return false;
    }
    public void prePrint(){
        prePrint(root);
    }

    private void prePrint(node temp){
        if(temp == null){
            return;
        }
        else{
            System.out.println(temp.data);
            prePrint(temp.left);
            System.out.println("__________");
            prePrint(temp.right);
        }
    }

    private void updateHeight(node temp){
        if(temp == null){// at root, we're done
            return;
        }
        if(height(temp.left)==temp.height || height(temp.right)==temp.heigh
            temp.height = temp.height+1;
        }
        if(imbalenced(temp)){
            rebalence(temp);
        }
        updateHeight(temp.parent);
    }

    private int height(node temp){
        if(temp == null){
            return 0;
        }
        else{
            return temp.height;
        }
    }

    private boolean imbalenced(node temp){
        int lheight = height(temp.left);
        int rheight = height(temp.right);
        if(rheight - lheight < -1 || rheight - lheight > 1){
            return true;
        }
        else{
            return false;
        }
    }

    private void rebalence(node temp){
        if(height(temp.left) > height(temp.right)){ //left heavy
            if(height(temp.left.left) > height(temp.left.right)){ //left he
                node parent = temp.parent;
                node nodeA = temp;
                node nodeB = temp.left;
                node subY = temp.left.right;

                nodeA.left = subY;
                nodeA.parent = nodeB;
                nodeB.right = nodeA;
                nodeB.parent = parent;
                if(subY != null){subY.parent = nodeA;}
                if(parent == null){ //special case, temp is root
                    root = nodeB;
                }
                else if(parent.left == temp){
                    parent.left = nodeB;
                }
                else{
                    parent.right = nodeB;
                }
                //update heights
                temp.height = Math.max(height(temp.left),height(temp.right)
                temp = nodeB;
                temp.height = Math.max(height(temp.left),height(temp.right)
            }
            else{//left heavy inside
                node parent = temp.parent;
                node nodeA = temp;
                node nodeB = temp.left;
                node nodeC = temp.left.right;
                node subY = temp.left.right.left;
                node subZ = temp.left.right.right;

                if(parent == null){ //special case, temp is root
                    root = nodeB;
                }
                else if(parent.left == temp){
                    parent.left = nodeC;
                }
                else{
                    parent.right = nodeC;
                }
                nodeA.left = subZ;
                nodeA.parent = nodeC;
                nodeB.right = subY;
                nodeB.parent = nodeC;
                nodeC.left = nodeB;
                nodeC.right = nodeA;
                nodeC.parent = parent;
                if(subY != null){subY.parent = nodeB;}
                if(subZ != null){subZ.parent = nodeA;}
                //update heights
                temp.height = Math.max(height(temp.left),height(temp.right)
                temp = nodeB;
                temp.height = Math.max(height(temp.left),height(temp.right)
                temp = nodeC;
                temp.height = Math.max(height(temp.left),height(temp.right)
            }
        }
        else{ //right heavy
            if(height(temp.right.left) > height(temp.right.right)){//right
                node parent = temp.parent;
                node nodeA = temp;
                node nodeB = temp.right;
                node nodeC = temp.right.left;
                node subY = temp.right.left.left;
                node subZ = temp.right.left.right;

                if(parent == null){ //special case, temp is root
                    root = nodeC;
                }
                else if(parent.left == temp){
                    parent.left = nodeC;
                }
                else{
                    parent.right = nodeC;
                }
                nodeA.right = subY;
                nodeA.parent = nodeC;
                nodeB.left = subZ;
                nodeB.parent = nodeC;
                nodeC.left = nodeA;
                nodeC.right = nodeB;
                nodeC.parent = parent;
                if(subY != null){subY.parent = nodeA;}
                if(subZ != null){subZ.parent = nodeB;}
                //update heights
                temp.height = Math.max(height(temp.left),height(temp.right)
                temp = nodeB;
                temp.height = Math.max(height(temp.left),height(temp.right)
                temp = nodeC;
                temp.height = Math.max(height(temp.left),height(temp.right)
            }
            else{//right heavy outside
                node parent = temp.parent;
                node nodeA = temp;
                node nodeB = temp.right;
                node subY = temp.right.left;

                nodeA.right = subY;
                nodeA.parent = nodeB;
                nodeB.left = nodeA;
                nodeB.parent = parent;
                if(subY != null){subY.parent = nodeA;}

                if(parent == null){ //special case, temp is root
                    root = nodeB;
                }
                else if(parent.left == temp){
                    parent.left = nodeB;
                }
                else{
                    parent.right = nodeB;
                }
                //update heights
                temp.height = Math.max(height(temp.left),height(temp.right)
                temp = nodeB;
                temp.height = Math.max(height(temp.left),height(temp.right)
            }
        }
    }

    private int count(node temp){
        if(temp == null){
            return 0;
        }
        else if(temp.right == null && temp.left == null){
            return 1;
        }
        else{
            return 1 + count(temp.left) + count(temp.right);
        }
    }
}
```

```java
import java.util.Scanner;
public class Main
{
    public static void main(String[] args){
        Scanner keyboard = new Scanner(System.in);
        bst bs = null;
        int parm;

        displayMenu();
        int option = keyboard.nextInt();

        while(option != 13){
            if (option == 1){
                int[] in = new int[20];
                for(int i=0;i<20;i++){
                    System.out.println("Enter an integer");
                    in[i] = keyboard.nextInt();
                }

                bs = new bst(in);
            }
            else if (option == 2){System.out.println(bs.nodes());}
            else if (option == 3){bs.leaves();}
            else if (option == 4){System.out.println(bs.height());}
            else if (option == 5){System.out.println(bs.odd());}
            else if (option == 6){System.out.println(bs.zero());}
            else if (option == 7){bs.descend();}
            else if (option == 8){bs.ascend();}
            else if (option == 9){System.out.println(bs.minimum());}
            else if (option == 10){
                System.out.println("Enter an integer");
                parm = keyboard.nextInt();
                bs.greater(parm);
            }
            else if (option == 11){
                System.out.println("Enter an node's value");
                parm = keyboard.nextInt();
                System.out.println(bs.level(parm));
            }
            else if (option == 12){
                System.out.println("Enter an node's value");
                parm = keyboard.nextInt();
                bs.path(parm);
            }
            else{System.out.println("Invalid Option");}

            displayMenu();
            option = keyboard.nextInt();
        }


    }
    private static void displayMenu(){
        System.out.println("1  - Enter a new BST\n" +
                           "2  - Count nodes\n" +
                           "3  - Print leaves\n" +
                           "4  - Print height\n" +
                           "5  - Sum odd values\n" +
                           "6  - Check for zero\n" +
                           "7  - Print in descending order\n" +
                           "8  - Print in ascending order\n" +
                           "9  - Print minimum value\n" +
                           "10 - Greater()\n" +
                           "11 - Find the level of a node\n" +
                           "12 - print the path to a node\n" +
                           "13 - Exit\n" +
                           "Choose an option - ");
    }


    private static String getParm(String parm){
        Scanner keyboard = new Scanner(System.in);
        String toReturn = "";
        if(parm == "path"){
            System.out.println("Enter the file path: ");
            toReturn = keyboard.nextLine();
        }
        else if (parm == "stuData"){
            System.out.println("Enter student id, name, age, and gpa sepera
            toReturn = keyboard.nextLine();
        }
        else if (parm == "id"){
            System.out.println("Enter student ID #: ");
            toReturn = String.valueOf(keyboard.nextInt());
        }
        else{
            System.out.println("Weird Error, you better contact IT");
        }
        return toReturn;
    }
}

```

```
public class bst
{
    private class node{
        int data;
        node left;
        node right;

        private node(int data, node left, node right){
            this.data = data;
            this.left = left;
            this.right = right;
        }
    }

    node root;

    public bst(int[] arr){
        for(int i=0; i<20; i++){
            insert(arr[i]);
        }
    }

    public int nodes(){
        return nodes(root);
    }

    public void leaves(){
        leaves(root);
    }

    public int height(){
        return height(root);
    }

    public int odd(){
        return odd(root);
    }

    public boolean zero(){
        return zero(root);
    }

    public void descend(){
        descend(root);
    }

    public void ascend(){
        ascend(root);
```

```java
    }

    public int minimum(){
        return minimum(root);
    }

    public void greater(int val){
        greater(root, val);
    }

    public int level(int val){
        return level(root, val, 1);
    }

    public void path(int val){
        System.out.println(path(root, val, ""));
    }

    private boolean insert(int data){
        if(root==null){//special case, insert at root
            root = new node(data, null, null);
        }
        else{
            node temp = root;
            node prev = null;

            while(temp != null){
                if(data == temp.data){
                    return false;
                }
                else if(data > temp.data){
                    prev = temp;
                    temp = temp.right;
                }
                else{
                    prev = temp;
                    temp = temp.left;
                }
            }
            if(data > prev.data){
                prev.right = new node(data, null, null);
            }
            else{
                prev.left = new node(data, null, null);
            }
        }
        return true;
    }

    private int nodes(node temp){
        if(temp == null){
            return 0;
        }
        else{
            return 1 + nodes(temp.left) + nodes(temp.right);
        }
    }

    private void leaves(node temp){
        if(temp == null){
            return;
        }
        else if(temp.left == null && temp.right == null){
            System.out.println(temp.data);
        }
        else{
            leaves(temp.left);
            leaves(temp.right);
        }
    }

    private int height(node temp){
        if(temp == null){
            return 0;
        }
        int lheight = height(temp.left);
        int rheight = height(temp.right);
        if(lheight > rheight){
            return 1 + lheight;
        }
        else{
            return 1 + rheight;
        }
    }

    private int odd(node temp){
        if(temp == null){
            return 0;
        }
        else if(temp.data % 2 == 1){
            return temp.data + odd(temp.left) + odd(temp.right);
        }
        else{
            return odd(temp.left) + odd(temp.right);
        }
    }

    private boolean zero(node temp){
        if(temp == null){
            return false;
        }
        else if(temp.data == 0){
            return true;
        }
        else{
            return (zero(temp.left) || zero(temp.right));
        }
    }

    private void descend(node temp){
        if(temp == null){
            return;
        }
        else{
            descend(temp.right);
            System.out.println(temp.data);
            descend(temp.left);
        }
    }

    private void ascend(node temp){
        if(temp == null){
            return;
        }
        else{
            descend(temp.left);
            System.out.println(temp.data);
            descend(temp.right);
        }
    }

    private int minimum(node temp){
        if(temp == null){
            return Integer.MAX_VALUE;
        }
        else{
            int toReturn;
            int lmin = minimum(temp.left);
            int rmin = minimum(temp.right);
            if(lmin < rmin && lmin < temp.data){
                toReturn = lmin;
            }
            else if(rmin < lmin && rmin < temp.data){
                toReturn = rmin;
            }
            else{
                toReturn = temp.data;
            }
            return toReturn;
        }
    }

    private void greater(node temp, int val){
        if(temp == null){
            return;
        }
        else if(temp.data > val){
            System.out.println(temp.data);
        }
        greater(temp.left, val);
        greater(temp.right, val);
    }

    private int level(node temp, int val, int lvl){
        if(temp == null){ //didn't find it
            return -1;
        }
        else if(temp.data == val){ //found it
            return lvl;
        }
        else{ //keep searching
            if(val > temp.data){ //go right
                return level(temp.right, val, lvl+1);
            }
            else{ //go left
                return level(temp.left, val, lvl+1);
            }
        }
    }

    private String path(node temp, int val, String path){
        if(temp == null){ //no path existed
            return "Path did not exist";
        }
        else if(temp.data == val){// found it
            return path + temp.data;
        }
        else{// keep searching
            if(val > temp.data){ //go right
                return path(temp.right, val, path+temp.data+"-->");
            }
            else{ //go left
```

```
                return path(temp.left, val, path+temp.data+"-->");
            }
        }
    }
}

```