Juan Sebastián Bernal Jiménez - 1022434946
Guillermo Alejandro Cano Rodriguez - 1121888697
Jhon Edison Prieto Artunduaga - 1007163058
Jonatan Javier Valero Comayan - 1007491800
Estructuras de Datos – Grupo 3
27 de mayo de 2023

Parcial Práctico No. 2 - Estructuras de Datos

1. Viaje:

```
import java.util.ArrayList;

import java.util.LinkedList;

import java.util.Scanner;


public class Main {


    public static class TreeNode<T> {

        private T key;

        private TreeNode<T> left;

        private TreeNode<T> right;


        public TreeNode(T key) {

            this.key = key;

            this.left = null;

            this.right = null;
```

```
}

public T getKey() {
    return key;
}

public void setKey(T key) {
    this.key = key;
}

public TreeNode<T> getLeft() {
    return left;
}

public void setLeft(TreeNode<T> left) {
    this.left = left;
}

public TreeNode<T> getRight() {
    return right;
}

public void setRight(TreeNode<T> right) {
        this.right = right;
    }

}

public static class RecursiveBinarySearchTree<T extends Comparable<T>> {

    protected TreeNode<T> root;

    public RecursiveBinarySearchTree(T data) {
        this.root = new TreeNode<>(data);
    }

    public RecursiveBinarySearchTree() {
        this.root = null;
    }

    public boolean search(T data) {
        return this.search(this.root, data) != null;
    }

    public void insert(T data) {
        this.root = this.insert(this.root, data);
```

```
}

public void delete(T data) {
    this.root = this.delete(this.root, data);
}

public void inOrder() {
    this.inOrder(this.root);
}

public void preOrder() {
    this.preOrder(this.root);
}

public void postOrder() {
    this.postOrder(this.root);
}


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

```
public int size() {
    return this.size(root);
}

public T minValue() {
    return this.minValue(this.root);
}

public T maxValue() {
    return this.maxValue(this.root);
}

public boolean isBalanced() {
    return this.isBalanced(root);
}

public void balance() {
    ArrayList<T> values = new ArrayList<>();
    this.inOrder(root, values);
    this.buildTree(values);
}

private void buildTree(ArrayList<T> values) {
        this.root = buildTreeUtil(0, values.size() - 1, values);
    }

    private TreeNode<T> buildTreeUtil(int start, int end, ArrayList<T>
values) {
        if (start > end) return null;
        int mid = (start + end) / 2;
        TreeNode<T> root = new TreeNode<>(values.get(mid));
        root.setLeft(this.buildTreeUtil(start, mid-1, values));
        root.setRight(this.buildTreeUtil(mid+1,end,values));
        return root;
    }

    private void inOrder(TreeNode<T> root, ArrayList<T> values) {
        if(root == null) return;
        this.inOrder(root.getLeft(), values);
        values.add(root.getKey());
        this.inOrder(root.getRight(), values);
    }

    public String toString() {
        return this.toString(this.root);
    }
```

```
private String toString(TreeNode<T> root) {
    LinkedList<TreeNode<T>> queue = new LinkedList<>();
    LinkedList<Integer> level = new LinkedList<>();
    if(root == null) return "";
    queue.add(root);
    level.add(0);
    StringBuilder sb = new StringBuilder();
    int preLevel = 0;
    while (!queue.isEmpty()) {
        TreeNode<T> temp = queue.poll();
        int l = level.poll();
        if(preLevel != l) {
            sb.append("\n");
            preLevel = l;
        }
        sb.append(temp.getKey() + " ");
        if(temp.getLeft() != null) {
            queue.add(temp.getLeft());
            level.add(l + 1);
        }
        if(temp.getRight() != null) {
            queue.add(temp.getRight());
            level.add(l + 1);
        }
    }
    return sb.toString();
}

private TreeNode<T> delete(TreeNode<T> root, T data) {
    if(root == null) return root;
    if(data.compareTo(root.getKey()) < 0) {
        root.setLeft(this.delete(root.getLeft(), data));
    }
    else if (data.compareTo(root.getKey()) > 0) {
        root.setRight(this.delete(root.getRight(), data));
    }
    else {
        if(root.getLeft() == null) return root.getRight();
        else if(root.getRight() == null) return root.getLeft();
        root.setKey(this.minValue(root.getRight()));
        root.setRight(delete(root.getRight(), root.getKey()));
    }
    return root;
}
```

```
private int size(TreeNode<T> root) {
    if(root == null)  return 0;
    return 1 + size(root.getLeft()) + size(root.getRight());
}

protected int height(TreeNode<T> root) {
    if(root == null) return 0;
    return 1 + Math.max(height(root.getLeft()), height(root.getRight()));
}

protected T minValue(TreeNode<T> root) {
    if(root == null) return null;
    if(root.getLeft() == null) return root.getKey();
    else return minValue(root.getLeft());
}

private T maxValue(TreeNode<T> root) {
    if(root == null) return null;
    if(root.getRight() == null) return root.getKey();
    else return maxValue(root.getRight());
}

private void inOrder(TreeNode<T> root) {
    if(root == null) return;
    this.inOrder(root.getLeft());
    System.out.print(root.getKey() + " ");
    this.inOrder(root.getRight());
}

private void preOrder(TreeNode<T> root) {
    if(root == null) return;
    System.out.print(root.getKey() + " ");
    this.inOrder(root.getLeft());
    this.inOrder(root.getRight());
}

private void postOrder(TreeNode<T> root) {
    if(root == null) return;
    this.inOrder(root.getLeft());
    this.inOrder(root.getRight());
    System.out.print(root.getKey() + " ");
}

private TreeNode<T> insert(TreeNode<T> root, T data) {
    if(root == null) root = new TreeNode<>(data);
    else if(data.compareTo(root.getKey()) < 0) {
        root.setLeft(this.insert(root.getLeft(), data));
    }
    else if(data.compareTo(root.getKey()) > 0) {
        root.setRight(this.insert(root.getRight(),data));
    }
    return root;
}

private T search(TreeNode<T> root, T data) {
    if(root == null) return null;
    if(root.getKey().compareTo(data) == 0) {
        return root.getKey();
    }
    else if(data.compareTo(root.getKey()) < 0) {
        return this.search(root.getLeft(),data);
    }
    else return this.search(root.getRight(), data);
}

protected boolean isBalanced(TreeNode<T> root) {
    if(root == null) return true;
    int lh = this.height(root.getLeft());
    int rh = this.height(root.getRight());
```

```
            if(Math.abs(lh - rh) > 1) return false;

            return this.isBalanced(root.getLeft()) &&
this.isBalanced(root.getRight());

        }


    }

    public static class AvlBinarySearch<T extends Comparable<T>> extends
RecursiveBinarySearchTree<T> {

        public int path(T data1, T data2) {

            return path(this.root,data1,data2);

        }


        public void insert(T data) {

            this.root = this.insert(this.root, data);

        }


        public void delete(T data) {

            this.root = this.delete(this.root, data);

        }


        public void balance() {

            throw new RuntimeException("Not implemented");

        }
```

```
private TreeNode<T> rotateLeft(TreeNode<T> root) {
    TreeNode<T> x = root.getRight();
    TreeNode<T> z = x.getLeft();
    x.setLeft(root);
    root.setRight(z);
    return x;
}

private TreeNode<T> rotateRight(TreeNode<T> root) {
    TreeNode<T> x = root.getLeft();
    TreeNode<T> z = x.getRight();
    x.setRight(root);
    root.setLeft(z);
    return x;
}

private TreeNode<T> insert(TreeNode<T> root, T data) {
    if(root == null) root = new TreeNode<>(data);
    else if(data.compareTo(root.getKey()) < 0) {
        root.setLeft(this.insert(root.getLeft(), data));
    }
    else if(data.compareTo(root.getKey()) > 0) {
        root.setRight(this.insert(root.getRight(),data));
    }
    return reBalance(root);
}

private TreeNode<T> delete(TreeNode<T> root, T data) {
    if(root == null) return root;
    if(data.compareTo(root.getKey()) < 0) {
        root.setLeft(this.delete(root.getLeft(), data));
    }
    else if (data.compareTo(root.getKey()) > 0) {
        root.setRight(this.delete(root.getRight(), data));
    }
    else {
        if(root.getLeft() == null) return root.getRight();
        else if(root.getRight() == null) return root.getLeft();
        root.setKey(this.minValue(root.getRight()));
        root.setRight(delete(root.getRight(), root.getKey()));
    }
    return reBalance(root);
}

private TreeNode<T> reBalance(TreeNode<T> root) {
    if(!this.isBalanced(root)) {
```

```
                int difference = this.height(root.getRight()) -
this.height(root.getLeft());

                if(difference > 1) {//We need to rotate left

                    if(this.height(root.getRight().getRight()) >
this.height(root.getRight().getLeft())) { //Only rotate left

                        root = rotateLeft(root);

                    } else { //Double rotation

                        root.setRight(rotateRight(root.getRight()));

                        root = rotateLeft(root);

                    }

                } else if (difference < -1) {//We need to rotate right

                    if(this.height(root.getLeft().getLeft()) >
this.height(root.getLeft().getRight())) { //Only rotate right

                        root = rotateRight(root);

                    } else { //Double rotation

                        root.setLeft(rotateLeft(root.getLeft()));

                        root = rotateRight(root);

                    }

                }

            }

            return root;

        }
```

```
    public Integer searchCountStep(TreeNode<T> root, T data1) {
        Integer step=0;
        Object[] found=new Object[2];
        while(true) {
            if(root == null) {
            return null;
            }
            if (data1.compareTo(root.getKey()) < 0) {
                step+=1;
                root=root.getLeft();
            } else if (data1.compareTo(root.getKey()) > 0) {
                step+=1;
                root=root.getRight();
            } else {
                return step;
            }
        }
    }

    public Integer path(TreeNode<T> root, T data1, T data2) {


        if(data1.compareTo(root.getKey())<0 &&
data2.compareTo(root.getKey())<0){
```

```
            return path(root.getLeft(),data1,data2);

         } else if(data1.compareTo(root.getKey())>0 &&
data2.compareTo(root.getKey())>0){

            return path(root.getRight(),data1,data2);

         } else {

            Object ans1=searchCountStep(root,data1);

                  Object ans2=searchCountStep(root,data2);

            if(ans1!=null && ans2!= null){

            return searchCountStep(root,data1)+searchCountStep(root,data2)+1;

            } else{ return -1;}

         }


      }




      }


   public static void main(String[] args) {

      AvlBinarySearch<String> cityTree= new AvlBinarySearch<>();

      String[] citiArr= {"Mongui", "Sachica", "Tinjaca", "Combita", "Chiquiza",
"Sutamarchan", "Tibasosa", "Toca", "Guican", "Chivata", "Topaga",

            "Soraca", "Gameza", "Guayata", "Raquira", "Nobsa", "Tenza",
"Aquitania"};

      for(String c: citiArr){
         cityTree.insert(c);
      }

      Scanner scanner= new Scanner(System.in);
      String city1=scanner.next();
      String city2=scanner.next();
      System.out.print(cityTree.path(city1,city2));

  }
}
```

2. **Galápagos:**

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Scanner;

  public class Main {
    private static ArrayList<Integer> zonesArr= new ArrayList<>();
    private static ArrayList<Integer> inAns= new ArrayList<>();

    private static ArrayList<Integer> preAns= new ArrayList<>();
    private static ArrayList<Integer> posAns= new ArrayList<>();

    public static int inOrderArr(int ind, int a){
      if(ind*2<= zonesArr.size()){
        inOrderArr(ind*2,  a);
      }
      if(a==inAns.size()){return 0;}

      if(zonesArr.get(ind-1)!=-1)inAns.add(zonesArr.get(ind-1));

      if(ind*2+1<= zonesArr.size()){
        inOrderArr(ind*2+1,  a);
      }
      return 0;
    };
    public static int preOrderArr(int ind, int a){
```

```
    if(a==preAns.size()){return 0;}

    if(zonesArr.get(ind-1)!=-1)preAns.add(zonesArr.get(ind-1));

    if(ind*2<= zonesArr.size()){
        preOrderArr(ind*2,  a);
    }
    if(ind*2+1<= zonesArr.size()){
        preOrderArr(ind*2+1,  a);
    }
    return 0;
};
public static int posOrderArr(int ind, int a){
    if(ind*2<= zonesArr.size()){
        posOrderArr(ind*2,  a);
    }

    if(ind*2+1<= zonesArr.size()){
        posOrderArr(ind*2+1,  a);
    }

    if(a==posAns.size()){return 0;}

    if(zonesArr.get(ind-1)!=-1)posAns.add(zonesArr.get(ind-1));
    return 0;
};
```

```
    public static void main(String[] args) {
        Scanner scanner= new Scanner(System.in);
        String zones= scanner.nextLine();
        Scanner zonesScan= new Scanner(zones);
        while(zonesScan.hasNextInt()){
            zonesArr.add(zonesScan.nextInt());
        }
        Integer a=scanner.nextInt();

inOrderArr(1,a);
preOrderArr(1,a);
posOrderArr(1,a);




int sIn, sPre,sPos;
        sIn= sPre=sPos=0;
        for(int i=0; i<a;i++){
            if(inAns.get(i)!=-1) sIn+=inAns.get(i);
            if(preAns.get(i)!=-1)sPre+=preAns.get(i);
            if(posAns.get(i)!=-1)sPos+=posAns.get(i);
        }

if(sPre>= sIn && sPre>= sPos){
    System.out.print("Preorder "+sPre);
} else if (sIn> sPre && sIn>= sPos){
   System.out.print("Inorder "+sIn);



} else {


   System.out.print("Postorder "+sPos);


}









      }
   }
```

3. **¿ Digan en qué casos de prueba fallaron, cuál creen que era la causa y cómo la solucionaron?**

   Durante la fase de desarrollo de ambos problemas, nos enfrentamos a un desafío considerable al tratar de identificar de manera precisa el origen del error que afectaba las respuestas obtenidas. A pesar de repetidos intentos y pruebas exhaustivas, nos encontramos con numerosos errores en tiempo de ejecución y errores de compilación que dificultaban nuestro avance. Además, realizamos cambios en el lenguaje de programación utilizado con el objetivo de descartar cualquier posibilidad de que el problema residiera en esa área específica.

Además de explorar diferentes enfoques para proporcionar las respuestas, modificamos la forma en que se presentaban, incluso ajustando los tipos de datos involucrados. A medida que perseveramos en la resolución de estos desafíos, nos dimos cuenta de la existencia de casos particulares que requerían un enfoque especial y cuidadoso para obtener resultados precisos.

Después de un arduo esfuerzo y una dedicación incansable, finalmente logramos superar los obstáculos y obtener un éxito rotundo en ambos problemas, alcanzando un puntaje perfecto del 100%.