```java
//tem q ser um tipo comparavél, n sei como faz pra comparar
public class Tree<TYPE extends Comparable> {
    private Node<TYPE> root;
    private int height;
    private int size;


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

    //GET
    public int getHeight(){
        return height;
    }
    public int getSize(){
        return size;
    }

    //methods
    public void addNode(TYPE value) {
        Node<TYPE> newNode = new Node<TYPE>(value);
        Node<TYPE> oldNode = this.root;
        int aux=0;
        if (oldNode==null){
            this.root = newNode;
            return;
        }
        boolean added=false;
        while(added!=true){
            //usando metodo doido pra comparar o valor novo com o
antigo, pra saber se vai pra esq ou dir
            // -1 igual menor, +1 igual maior, 0 igual igual
            if (newNode.getValue().compareTo(oldNode.getValue())==-1){
                if(oldNode.getLeft()!=null){
                    oldNode=oldNode.getLeft();
                }
                else{
                    oldNode.setLeft(newNode);
                    added=true;
                }
            }
            else{
```

```
                if(oldNode.getRight()!=null){
                    oldNode=oldNode.getRight();
                }
                else{
                    oldNode.setRight(newNode);
                    added=true;
                }
            }
            aux++;
        }
        //atualizar altura da arvore
        if(aux>this.height){
            this.height=aux;
        }
        this.size++;
    }
    //lembrar de colocar pra alterar a altura e o tamanho após o de
remover
    //METODO REMOVER
    // public boolean remove(TYPE value) {
    //     if (root == null) return false; // se arvore vazia

    //     Node atual = root;
    //     Node pai = root;
    //     boolean filho_esq = true;


    //     while (atual.item != v) {
    //       pai = atual;
    //       if(v < atual.item ) {
    //         atual = atual.esq;
    //         filho_esq = true;
    //       }
    //       else {
    //         atual = atual.dir;
    //         filho_esq = false;
    //       }
    //       if (atual == null) return false;
    //     }



    //     // Se nao possui nenhum filho (é uma folha), elimine-o
```

```
    //     if (atual.esq == null && atual.dir == null) {
    //       if (atual == root ) root = null; // se raiz
    //       else if (filho_esq) pai.esq = null; // se for filho a
esquerda do pai
    //            else pai.dir = null; // se for filho a direita do pai
    //     }

    //     // Se é pai e nao possui um filho a direita, substitui pela
subarvore a direita
    //     else if (atual.dir == null) {
    //        if (atual == root) root = atual.esq; // se raiz
    //        else if (filho_esq) pai.esq = atual.esq; // se for filho
a esquerda do pai
    //             else pai.dir = atual.esq; // se for filho a direita
do pai
    //     }

    //     // Se é pai e nao possui um filho a esquerda, substitui pela
subarvore a esquerda
    //     else if (atual.esq == null) {
    //        if (atual == root) root = atual.dir; // se raiz
    //        else if (filho_esq) pai.esq = atual.dir; // se for filho
a esquerda do pai
    //             else pai.dir = atual.dir; // se for  filho a direita
do pai
    //     }


    //     else {
    //       No sucessor = no_sucessor(atual);

    //       if (atual == root) root = sucessor;
    //       else if(filho_esq) pai.esq = sucessor;
    //            else pai.dir = sucessor; ai
    //       sucessor.esq = atual.esq;
    //     }

    //     return true;
    //   }

    public void inOrder(Node<TYPE> current) {
        if (current != null) {
          inOrder(current.left);
```

```
          System.out.print(current.value+ ", ");
          inOrder(current.right);
        }
      }

    //Descobre altura da arvore recursivamente
    public int height(Node<TYPE> current) {
        if(current == null || (current.left == null && current.right ==
null)){
          return 0;
        }
        else {
          if (height(current.left) > height(current.right)){
            return ( 1 + height(current.left) );
          }
          else{
            return ( 1 + height(current.right) );
        }
        }
    }

    public Node<TYPE> minElement() {
        Node<TYPE> current = root;
        Node<TYPE> previous = null;
        while (current != null) {
            previous = current;
            current = current.left;
        }
        return previous;
      }

    public Node<TYPE> maxElement() {
        Node<TYPE> current = root;
        Node<TYPE> previous = null;
        while (current != null) {
            previous = current;
            current = current.right;
        }
        return previous;
    }
    public boolean searchNode(TYPE value) {
        Node<TYPE> newNode = new Node<TYPE>(value);
        Node<TYPE> oldNode = this.root;
```

```
        if (oldNode==null){
            return false;
        }
        while(oldNode!=null){
            // -1 igual menor, +1 igual maior, 0 igual igual
            if (newNode.getValue().compareTo(oldNode.getValue())==0){
                return true;
            }
            if (newNode.getValue().compareTo(oldNode.getValue())==-1){
                oldNode=oldNode.getLeft();
            }
            else
if(newNode.getValue().compareTo(oldNode.getValue())==1){
                oldNode=oldNode.getRight();
            }
        }
        return false;
    }
    public void removeNode(TYPE value) {
        Node<TYPE> oldNode = this.root;
        if (oldNode==null){
            return;
        }
        Node<TYPE> parent = null;
        Node<TYPE> newNode = new Node<TYPE>(value);

        while(oldNode!=null){
            // -1 igual menor, +1 igual maior, 0 igual igual
            if (newNode.getValue().compareTo(oldNode.getValue())==0){
                //found
                //remove
                if(oldNode.getRight()!=null &&
oldNode.getLeft()!=null){

                }
                else if(oldNode.getRight()!=null){

                }
                else if(oldNode.getLeft()!=null){
                    //parent.setRight
                }
                else{
```

```
                    if
(oldNode.getValue().compareTo(parent.getValue())==1){
                        parent.setRight(null);
                    }
                    else{
                        parent.setLeft(null);
                    }
                }
                return;
            }
            else if
(newNode.getValue().compareTo(oldNode.getValue())==-1){
                parent = oldNode;
                oldNode=oldNode.getLeft();
            }
            else
if(newNode.getValue().compareTo(oldNode.getValue())==1){
                parent = oldNode;
                oldNode=oldNode.getRight();
            }
        }
    }
}
```

```java
public class App {
    public static void main(String[] args) throws Exception {
        Tree<Student> newTree = new Tree<Student>();
        Student teste1 = new Student("Rodrigo",2,0);
        Student teste2 = new Student("jao",3,0);
        Student teste3 = new Student("paulo",1,0);
        Student teste4 = new Student("Rod",4,0);
        newTree.addNode(teste1);
        newTree.addNode(teste2);
        newTree.addNode(teste3);
        newTree.addNode(teste4);
        newTree.searchNode(teste1);
        newTree.searchNode(teste2);
        newTree.searchNode(teste3);
        newTree.searchNode(teste4);
    }
}
```

```java
public class Student implements Comparable<Student> {
    private String name;
    private int id;
    private float grade;


    public Student(String name, int id, float grade) {
        this.name = name;
        this.id = id;
        this.grade = grade;
    }

    @Override
    public int compareTo(Student Student) {
       if(this.id == Student.id){
           return 0;
       }
       else if(this.id > Student.id){
           return 1;
       }
       else{
           return -1;
       }
    }
    @Override
    public String toString() {
        String value = String.valueOf(getId());
        return "Nome: "+ getName() +" Matricula: "+ value;
    }

    //get set
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;

    }

    public float getGrade() {
        return grade;
    }
```

```
    public void setGrade(float grade) {
        this.grade = grade;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

}
```

```java
public class Node<TYPE>{
    //value == Aluno
    TYPE value;
    Node<TYPE> left;
    Node<TYPE> right;

    public Node(TYPE value)
    {
        this.value = value;
    }

    //GET
    public Node<TYPE> getLeft(){
        return left;
    }
    public Node<TYPE> getRight(){
        return right;
    }
    public TYPE getValue(){
        return this.value;
    }

    //SET
    public void  setValor(TYPE value){
        this.value = value;
    }
    public void setRight(Node<TYPE> right){
        this.right = right;
    }
    public void setLeft(Node<TYPE> left){
        this.left = left;
    }


}
```

TY