UNIVERSIDAD AUTÓNOMA GABRIEL RENE MORENO
FACULTAD DE INGENIERÍA Y CIENCIAS DE LA COMPUTACIÓN Y TELECOMUNICACIONES

ESTRUCTURAS DE DATOS 2

CONTENIDO: *// Breve Descripción del Contenido*

LAB-1. ÁRBOLES BINARIOS DE BÚSQUEDA.

PORCENTAJE TERMINADO : 100%.

GRUPO: 14

Nombre	Registro
Cristian Gabriel Gedalge Cayhuara	219022062

Fecha de Presentación: Viernes ,03 de mayo de 2024

COMENTARIO:
Me sorprendió que en el tema de arboles todos los algoritmos utilizan recursividad, fuera de eso los ejercicios me parecieron fáciles de hacer y no tuve ningún problema

CLASE NODO

```
public class Nodo {
    public Nodo izq;
    public Nodo der;
    public int elem;
    public Nodo(int ele)
    {
        this.elem=ele;
        this.izq=this.der=null;
    }
}
```

CLASE ARBOL

```
public class Arbol {

    public Nodo raiz;

    public Arbol() {
        this.raiz = null;
    }

//1. A1.insertar(x) : Método que inserta el elemento x, en el
árbol A1 en su lugar correspondiente.
    public void insertar(int x) {
        this.raiz = insertar(x, raiz);
    }

    private Nodo insertar(int x, Nodo p) {
        if (p == null) {
            return new Nodo(x);
        }
        if (x < p.elem) {
            p.izq = insertar(x, p.izq);
        } else {
            p.der = insertar(x, p.der);
        }
        return p;
    }
//2. A1.preOrden() : Método que muestra los elementos del
árbol A1 en preOrden.

    public void preOrden()
    {
        preOrden(raiz);
    }
```

```
    public void preOrden(Nodo p)
    {
        if(p==null)
            return;
        System.out.println(p.elem);
        preOrden(p.izq);
        preOrden(p.der);
    }
//3. A1.inOrden() : Método que muestra los elementos del
árbol A1 en inOrden.

    public void inOrden() {
        inOrden(raiz);
    }

    private void inOrden(Nodo p) {
        if (p == null) {
            return;
        }
        inOrden(p.izq);
        System.out.println(p.elem);
        inOrden(p.der);
    }
//4. A1.postOrden() : Método que muestra los elementos del
árbol A1 en postOrden.
    public void postOrden()
    {
        System.out.print("[");
        postOrden(raiz);
        System.out.print("]");
    }

    public void postOrden(Nodo p)
    {
        if(p==null)
            return;
        postOrden(p.izq);
        postOrden(p.der);
        System.out.print(p.elem+",");
    }
//5. A1.seEncuentra(x) : Métodos lógico que devuelve True, si
el elemento x, se encuentra en el árbol A1.

    public boolean seEncuentra(int x) {
        return seEncuentra(x, raiz);
    }
```

```
    private boolean seEncuentra(int x, Nodo p) {
        if (p == null) {
            return false;
        }
        if (x == p.elem) {
            return true;
        }
        if (x < p.elem) {
            return seEncuentra(x, p.izq);
        } else {
            return seEncuentra(x, p.der);
        }

    }
//6. A1.cantidad() : Método que devuelve la cantidad de nodos
del árbol A1

    public int cantidad() {
        return cantidad(raiz);
    }

    private int cantidad(Nodo p) {
        int cant;
        if (p == null) {
            return 0;
        } else {
            cant = 1 + cantidad(p.izq) + cantidad(p.der);
        }
        return cant;
    }
//7. A1.suma() : Método que devuelve la suma de los elementos
del árbol A1.

    public int suma() {
        return suma(raiz);
    }

    private int suma(Nodo p) {
        int sum;
        if (p == null) {
            return 0;
        } else {
            sum = p.elem + (suma(p.izq) + suma(p.der));
        }
        return sum;
    }
```

```
//8. A1.menor() : Método que devuelve el elemento menor del
árbol A1.

    public int menor() {
        return menor(raiz);
    }

    private int menor(Nodo p) {
        if (p.izq == null) {
            return p.elem;
        } else {
            return menor(p.izq);
        }
    }
//9. A1.mayor() : Método que devuelve el elemento mayor del
árbol A1.

    public int mayor() {
        return mayor(raiz);
    }

    private int mayor(Nodo p) {
        if (p.der == null) {
            return p.elem;
        } else {
            return menor(p.der);
        }
    }
//10. A1.cantidadTerm() : Método que devuelve la cantidad de
nodos terminales del árbol A1.

    public int cantidadTerm() {
        return cantidadTerm(this.raiz);
    }

    private int cantidadTerm(Nodo p) {
        int cantTerm;
        if (p == null) {
            return 0;
        } else if (p.izq == null && p.der == null) {
            return 1;
        } else {
            cantTerm = cantidadTerm(p.izq) +
cantidadTerm(p.der);
        }
        return cantTerm;
    }
```

```
//11. A1.sumaPares() : Método que devuelve la suma de los
elementos pares  del árbol A1.
    public int sumaPares() {
        return sumaPares(raiz);
    }

    private int sumaPares(Nodo p) {
        int sum;
        if (p == null) {
            return 0;

        } else if ((p.elem % 2) == 0) {
            sum = p.elem + (sumaPares(p.izq) +
sumaPares(p.der));
        } else {
            sum = (sumaPares(p.izq) + sumaPares(p.der));
        }
        return sum;
    }

    public static void main(String[] args) {
        Arbol A1 = new Arbol();
        A1.insertar(3);
        A1.insertar(4);
        A1.insertar(8);
        A1.insertar(5);
        A1.insertar(3);
        A1.insertar(2);

        A1.postOrden();
        System.out.println(A1.cantidadTerm());

    }

}
```