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

ESTRUCTURAS DE DATOS 2

CONTENIDO: *// Breve Descripción del Contenido*

TAREA-3. ABB CON LIBRERÍA DE LISTAS

PORCENTAJE TERMINADO : 100%.

GRUPO: 14

Nombre	Registro
Cristian Gabriel Gedalge Cayhuara	219022062

Fecha de Presentación: Lunes ,13 de mayo de 2024

COMENTARIO:
Todos los ejercicios tuvieron su dificultas pero pude realizarlos todos

CLASE NODO

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

```
public class Arbol {

    public Nodo raiz;

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

// 1.   A1.generarElem(n, a, b) : Método que genera n
elementos aleatorios enteros diferentes entre a y b
inclusive.
    public void generarElem(int n, int a, int b) {
        for (int i = n; i > 0; i--) {
            int x = (int) (Math.random() * (b - a)) + a;
            insertar(x);
        }
    }
//2.       A1.insertar(x) : Método que inserta el elemento x,
en el árbol A1 en su lugar correspondiente.
```

```
    public void insertar(int ele) {
        this.raiz = insertar(this.raiz, ele);
    }

    private Nodo insertar(Nodo p, int ele) {
        if (p == null) {
            p = new Nodo(ele);
            return p;
        } else {
            if (ele < p.elem) {
                p.izq = insertar(p.izq, ele);
            } else {
                p.der = insertar(p.der, ele);
            }
        }
        return p;
    }
//3.       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);
    }
//4.       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);
    }
```

```
//5.       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 + ",");
    }
//6.       A1.niveles(): Método que muestra los elementos del
árbol A1, por niveles.

    public int nivel() {
        if (this.raiz != null) {
            int nivelizq = nivel(raiz.izq) + 1;
            int nivelder = nivel(raiz.der) + 1;
            if (nivelizq >= nivelder) {
                return nivelizq;
            }
            return nivelder;
        } else {
            return 0;
        }

    }

    private int nivel(Nodo p) {
        int x;
        if (p == null) {
            return 0;
        } else {
            if (nivel(p.izq) > nivel(p.der)) {
                x = 1 + nivel(p.izq);
            } else {
                x = 1 + nivel(p.der);
            }
        }
        return x;

    }
```

```
    public void niveles() {
        for (int i = 0; i < nivel(); i++) {
            System.out.println("Nivel->" + i);
            niveles(this.raiz, i, 0);
        }
    }

    public void niveles(Nodo p, int nivelfijo, int nivel) {
        if (p == null || nivel <= nivelfijo) {
            return;
        }

        niveles(p.izq, nivelfijo, nivel + 1);
        niveles(p.der, nivelfijo, nivel + 1);
        if (nivelfijo == nivel) {
            System.out.println(p.elem);
        }
    }
//7.       A1.desc(): Método que muestra los elementos del
árbol A1 de mayor a menor.

    public void desc() {
        desc(this.raiz);
    }

    private void desc(Nodo p) {
        if (p == null) {
            return;
        }
        desc(p.der);
        System.out.print(p.elem + " ,");
        desc(p.izq);

    }
//8.       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);
        }

    }

//9.       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;
    }
//10.   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;
    }
//11.   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);
        }
    }

//12.   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);
        }
    }
//13.   A1.preoOrden(L1) : Método que encuentra en la lista
L1, el recorrido de preOrden de los elementos del árbol A1.

    public void preOrden(ArrayList<Integer> L1) {
        preOrden(this.raiz, L1);
    }

    private void preOrden(Nodo p, ArrayList<Integer> L1) {
        if (p == null) {
            return;
        }
        L1.add(p.elem);
        preOrden(p.izq, L1);
        preOrden(p.der, L1);

    }
//14.   A1.inOrden(L1) : Método que encuentra en la lista L1,
el recorrido de inOrden de los elementos del árbol A1.

    public void inOrden(ArrayList<Integer> L1) {
        inOrden(this.raiz, L1);
    }

    private void inOrden(Nodo p, ArrayList<Integer> L1) {
        if (p == null) {
            return;
        }
```

```
        inOrden(p.izq, L1);
        L1.add(p.elem);
        inOrden(p.der, L1);

    }
//15.   A1.postOrden(L1) : Método que encuentra en la lista
L1, el recorrido de postOrden de los elementos del árbol A1.

    public void postOrden(ArrayList<Integer> L1) {
        postOrden(raiz, L1);
    }

    public void postOrden(Nodo p, ArrayList<Integer> L1) {
        if (p == null) {
            return;
        }
        postOrden(p.izq, L1);
        postOrden(p.der, L1);
        L1.add(p.elem);
    }
//16.   A1.niveles(L1) : Método que encuentra en la lista L1,
el recorrido por niveles de los elementos del árbol A1.

    public void niveles(ArrayList<Integer> L1, int nivel) {
        niveles(this.raiz, L1, nivel);
    }

    private void niveles(Nodo p, ArrayList<Integer> L1, int
nivel) {
        if (p == null) {
            return;
        }
        elementoNivel(p.izq, nivel + 1);
        System.out.println(p.elem + "t" + nivel);
        L1.add(nivel);
        elementoNivel(p.izq, nivel + 1);
    }
//17.   A1.mostrarNivel(): Método que muestra los elementos
del árbol y el nivel en el que se encuentran. (Recorrer el
árbol en cualquier orden)

    public void mostrarnivel() {
        LinkedList<Nodo> L1 = new LinkedList();
        if (raiz == null) {
            return;
        }
        L1.add(raiz);
        while (!L1.isEmpty()) {
```

```
            Nodo p = L1.getFirst();
            System.out.print(p.elem);
            if (p.izq != null) {
                L1.add(p.izq);
            }
            if (p.der != null) {
                L1.add(p.der);
            }
            L1.removeFirst();
        }
    }
//18. A1.sumarNivel(L1) : Método que encuentra en la Lista de
acumuladores por nivel L1, la suma de los elementos de cada
nivel.

    public void sumaNivel() {
        int max = cantidad();
        ArrayList<Integer> L1 = new ArrayList(max);
        for (int i = 0; i < max; i++) {
            L1.add(0);
        }
        sumaNivel(this.raiz, 0, L1);
        int i = 0;
        while (L1.get(i) != 0) {
            System.out.println(i + 1 + "\t" + L1.get(i));
            i++;
        }

    }

    public void sumaNivel(Nodo p, int nivel,
ArrayList<Integer> L1) {
        if (p == null) {
            return;
        }
        L1.set(nivel, L1.get(nivel) + p.elem);
        sumaNivel(p.izq, nivel + 1, L1);
        sumaNivel(p.der, nivel + 1, L1);

    }
//19.   A1.alturaMayor(): Método que devuelve la altura del
árbol A1. (Altura es la máxima longitud de la raíz a un nodo
hoja en el árbol)

    public int alturaMayor() {
        if (this.raiz != null) {
            int nivelizq = nivel(raiz.izq)+1;
            int nivelder = nivel(raiz.der)+1;
```

```
            if (nivelizq >= nivelder) {
                return nivelizq;
            }
            return nivelder;
        } else {
            return 0;
        }
    }
//20.   A1.alturaMenor(): Método que devuelve la menor altura
del árbol A1.

    public int alturaMenor() {
        if (this.raiz != null) {
            int nivelizq = nivel2(raiz.izq)+1;
            int nivelder = nivel2(raiz.der)+1;
            if (nivelizq <= nivelder) {
                return nivelizq;
            }
            return nivelder;
        } else {
            return 0;
        }
    }
    private int  nivel2(Nodo p)
    {
        int x;
        if(p==null)
            return 0;
        else
        {
            if(nivel2(p.izq)<nivel2(p.der)){
                if(nivel2(p.izq)!=0)
                   x=1+nivel2(p.izq);
                else
                   x=1+nivel(p.der);
            }else{
                if(nivel2(p.der)!=0)
                   x=1+nivel2(p.der);
                else
                    x=1+nivel(p.izq);
            }
        }
        return x;
    }
//21.   A1.mostrarTerm(): Método que muestra los elementos de
los nodos terminales del árbol A1. Mostrar los elementos de
menor a mayor.
```

```
    public void mostrarTerm() {
        mostrarTerm(this.raiz);
    }

    private void mostrarTerm(Nodo p) {
        if (p == null) {
            return;
        } else if (p.izq == null && p.der == null) {
            System.out.println(p.elem);
        } else {
            mostrarTerm(p.izq);
            mostrarTerm(p.der);
        }
    }
//22.   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;
    }
//23.   A1.lineal() : Método lógico que devuelve True, si el
árbol A1 es un árbol degenerado o lIneal. (Se puede dar
cuando se genera el árbol con una secuencia ordenada de
elementos)

    public boolean lineal() {
        return lineal(this.raiz);
    }

    private boolean lineal(Nodo p) {
        if (p == null) {
            return false;
        } else {
            return (lineal(p.der) && lineal(p.izq));
        }
```

```
    }
//24. A1.inmediatoSup(x) : Método que devuelve el elemento
inmediato superior a x, si x se encuentra en A1, caso
contrario devuelve el mismo elemento.

    public int inmediatoSup(int x) {
        if (seEncuentra(x)) {
            return inmediatoSup(x, this.raiz);
        } else {
            return x;
        }
    }

    private int inmediatoSup(int x, Nodo p) {
        int res;
        if (p == null) {
            return x;
        } else if (x == p.elem) {
            return p.elem;
        } else {
            if (x < p.elem) {
                res = inmediatoSup(x, p.izq);
            } else {
                res = inmediatoSup(x, p.der);
            }

            if (res == x) {
                res = p.elem;
            }

        }
        return res;
    }
//25. A1.inmediatoInf(x) : Método que devuelve el elemento
inmediato inferior a x, si x se encuentra en A1, caso
contrario devuelve el mismo elemento.

    public int inmediatoInf(int x) {
        if (seEncuentra(x)) {
            return inmediatoInf(x, this.raiz);
        } else {
            return x;
        }
    }

    private int inmediatoInf(int x, Nodo p) {
        int res;
        if (x == p.elem) {
```

```
            if (p.izq == null && p.der == null) {
                return x;
            }
            if (p.izq != null) {
                return p.izq.elem;
            } else {
                return p.der.elem;
            }
        } else {
            if (x < p.elem) {
                res = inmediatoInf(x, p.izq);
            } else {
                res = inmediatoInf(x, p.der);
            }
        }
        return res;
    }
//26.   Implementar al menos 5 Ejercicios adicionales
cualesquiera, de consultas sobre uno o más árboles binarios
de búsqueda. Citar fuentes.
    //1. Implementar una funcion para determinar la altura de
un
    //arbol

    public int altura() {  // la altura la determinamos
apartir del nodo hijo para adelante en est ejercicio
        return altura(this.raiz);
    }

    private int altura(Nodo p) {
        int x;
        if (p == null) {
            return -1;
        } else {
            if (altura(p.izq) > altura(p.der)) {
                x = 1 + altura(p.izq);
            } else {
                x = 1 + altura(p.der);
            }
        }
        return x;
    }

//2. implementar una funcion que sume los elementos pares de
//    un arbol
    public int sumaPares() {
        return sumaPares(this.raiz);
    }
```

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

// 3. Devolver true si exiten mas elemenos pares que impares
//    en el arbol
    public boolean masPares() {
        return (cantPares() > cantImpares());
    }

    private int cantPares() {
        return cantPares(this.raiz);
    }

    public int cantPares(Nodo p) {
        int cantPares;
        if (p == null) {
            return 0;
        } else {
            cantPares = cantPares(p.izq) + cantPares(p.der);
            if (p.elem % 2 == 0) {
                cantPares = cantPares + 1;
            }
        }
        return cantPares;

    }
// 4. mostrar la cantidad de elementos impares del arbol

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

    private int cantImpares(Nodo p) {
        int cantImpares;
        if (p == null) {
            return 0;
        } else {
```

```
            cantImpares = cantImpares(p.izq) +
cantImpares(p.der);
            if (p.elem % 2 != 0) {
                cantImpares = cantImpares + 1;
            }
        }
        return cantImpares;

    }

// 5. Insertar el nodo Izq(RAMA IZQ) de A1 en  A2
    public void insertarenA2(Arbol A1) {
        insertarenA2(A1.raiz.izq);
    }

    private void insertarenA2(Nodo p) {
        if (p != null) {
            insertar(p.elem);
            insertarenA2(p.izq);
            insertarenA2(p.der);
        }
    }

    public void elementoNivel(int nivel) {
        elementoNivel(this.raiz, nivel);
    }

    private void elementoNivel(Nodo p, int nivel) {
        if (p == null) {
            return;
        }
        elementoNivel(p.izq, nivel + 1);
        System.out.println(p.elem + "t" + nivel);
        elementoNivel(p.izq, nivel + 1);
    }

    public static void main(String[] arg) {
        Arbol A1 = new Arbol();
        A1.insertar(14);
        A1.insertar(15);
        A1.insertar(13);
        //A1.insertar(13);
        A1.insertar(4);

        A1.insertar(5);
        A1.insertar(14);
        A1.insertar(14);
```

```
        A1.insertar(7);
        A1.insertar(16);
        A1.insertar(17);
//        A1.insertar(18);

        System.out.println(A1.alturaMenor());

    }

}
```

BIBLIOGRAFIA Ejercicios sobre Árboles Generales (ugr.es) Ejercicios Arboles 1920 Soluciones(1).pdf (cartagena99.com)