*UNIVERSIDAD AUTÓNOMA GABRIEL RENE MORENO*

*FACULTAD DE INGENIERÍA Y CIENCIAS DE LA COMPUTACIÓN*

*Y TELECOMUNICACIONES*

# ESTRUCTURA DE DATOS 2

**CONTENIDO:** Tarea de Arboles Binarios de Busqueda

**PORCENTAJE TERMINADO : 100%.**

**GRUPO:** 15

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Integrantes** | **DT** | **HG** | **HI** | **EVAL** |
| Ibarra Cuellar Gustavo | 1 | 1 | 1 | 100 |

**Fecha de Presentación:** Jueves 24 de octubre 2024

**Fecha Presentada:** Jueves 24 de Octubre 2024

## CLASES

public class Arbol {

public Nodo raiz;

public Arbol(){

raiz=null;

}

**A1.generarElem(n,a,b)**

public void generarElem(int n,int a,int b){

for(int i=0; i<n; i++){

int num= ThreadLocalRandom.current().nextInt(a,b+1);

this.insertar(num);

}

}

**A1.insertar(x)**

public void insertar(int x){

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;

}

**A1.preOrden()**

public void preOrden(){

preOrden(raiz);

}

private void preOrden(Nodo p){

if(p==null)return;

System.out.println(p.elem);

preOrden(p.izq);

preOrden(p.der);

}

**A1.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);

}public void postOrden() {

postOrden(raiz);

}

**A1.posOrden()**

private void postOrden(Nodo p) {

if (p == null) return;

postOrden(p.izq);

postOrden(p.der);

System.out.println(p.elem);

}

**A1.desc()**

public void desc(){

desc(raiz);

}

private void desc(Nodo p){

if(p==null)return;

desc(p.der);

System.out.println(p.elem);

desc(p.izq);

}

**A1.seEncuentra(x)**

public boolean seEncuentra(int x){

return seEncuentra(x,raiz);

}

private boolean seEncuentra(int x,Nodo p) {

if(p==null)return false;

if(p.elem==x)return true; else

return seEncuentra(x,p.izq) || seEncuentra(x,p.der);

}

**A1.cantidad()**

public int cantidad(){

return cantidad(raiz);

}

private int cantidad(Nodo p){

if(p==null) return 0;

else

return cantidad(p.izq)+cantidad(p.der)+1;

}

**A1.suma()**

public int suma(){

return suma(raiz);

}

private int suma(Nodo p){

if(p==null)return 0;

return p.elem+suma(p.izq)+suma(p.der);

}

**A1.mayor()**

public int mayor(){

if(raiz==null) { throw new IllegalStateException("Arbol Vacio"); }

return mayor(raiz);

}

private int mayor(Nodo p){

while(p.der != null){

p=p.der;

}

return p.elem;

}

**A1.menor()**

public int menor(){

if(raiz==null) { throw new IllegalStateException("Arbol Vacio"); }

return menor(raiz);

}

private int menor(Nodo p){

while(p.izq != null){

p=p.izq;

}

return p.elem;

}

**A1.mostrarTerm()**

public void mostrarTerm(){

ArrayList<Integer> L1 = new ArrayList<>();

encontrarTerminales(raiz, L1);

Collections.sort(L1);

System.out.println("Nodos terminales en orden de menor a mayor: " + L1);

}

private void encontrarTerminales(Nodo p,ArrayList<Integer> L1){

if (p == null) return;

if (p.izq == null && p.der == null) {

L1.add(p.elem);

} else {

encontrarTerminales(p.izq,L1);

encontrarTerminales(p.der,L1);

}

}

**A1.cantidadTerm()**

public int cantidadTerm(){

return this.cantidadTerm(raiz);

}

private int cantidadTerm(Nodo p){

if(p==null) return 0;

if(p.izq==null && p.der==null)

return 1;

return cantidadTerm(p.izq) + cantidadTerm(p.der);

}

**A1.lineal()**

public boolean Lineal() {

return esLineal(raiz);

}

private boolean esLineal(Nodo p) {

if (p == null) return true;

if (p.izq != null && p.der != null) {

return false;

}

return esLineal(p.izq) && esLineal(p.der);

}

public Integer inmediatoSup(int x) {

Nodo nodoX = buscarNodo(raiz, x);

if (nodoX == null) {

return x;

}

Nodo sup = encontrarInmediatoSuperior(nodoX);

return (sup != null) ? sup.elem : x; // if (sup != null)? return sup.elem else return x;

}

private Nodo buscarNodo(Nodo p, int x) {

if (p == null) return null;

if (p.elem == x) return p;

return (x < p.elem)? buscarNodo(p.izq, x) : buscarNodo(p.der, x);

}

private Nodo encontrarInmediatoSuperior(Nodo p) {

if (p.der != null) {

return encontrarMinimo(p.der);

}

return null;

}

private Nodo encontrarMinimo(Nodo p) {

while (p.izq != null) {

p = p.izq;

}

return p;

}

}//end.