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-7. ARBOLES BINARIOS DE BÚSQUEDA. ELIMINAR

PORCENTAJE TERMINADO : 100%.

Nombre	Grupo	Registro
Jorge Choque Calle	12	219074240
Cristian Gabriel Gedalge Cayhuara	14	219022062

Fecha de Presentación: Jueves ,04 de junio de 2024

```
public class Arbol {

    public Nodo raiz;

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

    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.ele) {
                p.izq = insertar(p.izq, ele);
            } else {
                p.der = insertar(p.der, ele);
            }
        }
        return p;
    }

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

    private void inOrdenOrdAsc(Nodo p) {
        if (p == null) {
            return;
        }
        inOrdenOrdAsc(p.izq);
        System.out.println(p.ele);
        inOrdenOrdAsc(p.der);

    }

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

    private void inOrdenOrdDesc(Nodo p) {
        if (p == null) {
            return;
        }
```

```
    inOrdenOrdDesc(p.der);
    System.out.println(p.ele);
    inOrdenOrdDesc(p.izq);

}

public void eliminar(int x) {
    raiz = eliminar(x, raiz);
}

public Nodo eliminar(int x, Nodo p) {
    if (p == null) {
        return null;
    }

    if (x == p.ele) {
        return eliminarNodo(p);
    }
    if (x < p.ele) {
        p.izq = eliminar(x, p.izq);
    } else {
        p.der = eliminar(x, p.der);
    }
    return p;

}

public Nodo eliminarNodo(Nodo p) {
    if (p.izq == null && p.der == null) {
        return null;
    }
    if (p.izq != null && p.der == null) {
        return p.izq;
    }
    if (p.izq == null && p.der != null) {
        return p.der;
    }
    Nodo q = p.izq;
    while (q.der != null) {
        q = q.der;
    }

    int y = q.ele;
    eliminar(y);
    p.ele = y;
    return p;
}
```

```
    public void eliminar2(int x) {
        raiz = eliminar2(x, raiz);
    }

    public Nodo eliminar2(int x, Nodo p) {
        if (p == null) {
            return null;
        }

        if (x == p.ele) {
            return eliminarNodo2(p);
        }
        if (x < p.ele) {
            p.izq = eliminar2(x, p.izq);
        } else {
            p.der = eliminar2(x, p.der);
        }
        return p;

    }

    public Nodo eliminarNodo2(Nodo p) {
        if (p.izq == null && p.der == null) {
            return null;
        }
        if (p.izq != null && p.der == null) {
            return p.izq;
        }
        if (p.izq == null && p.der != null) {
            return p.der;
        }
        Nodo q = p.der;
        while (q.izq != null) {
            q = q.izq;
        }

        int y = q.ele;
        eliminar(y);
        p.ele = y;
        return p;
    }
```

```
//A1.eliminarSup(x) : Método que elimina el elemento x, del árbol A1. Si el elemento a eliminar es un nodo raíz, buscar el elemento inmediato Superior, para eliminar.
   public void eliminarSup(int x){
       eliminar2(x);
   }
```

```
//A1.eliminarInf(x): Método que elimina el elemento x, del
árbol A1. Si el elemento a eliminar es un nodo raíz, buscar
el elemento inmediato Inferior, para eliminar

    public void eliminarInf(int x) {
        eliminar(x);
    }

//A1.eliminarHojas() : Método que elimina los nodos hoja de
árbol A1.

    public void eliminarHojas()
    {
        eliminarHojas(raiz);
    }
    public Nodo eliminarHojas(Nodo p)
    {
        if(p==null)
            return null;
        if(p.izq==null && p.der==null){
            return null;
        }
        p.izq=eliminarHojas(p.izq);
        p.der=eliminarHojas(p.der);
        return p;
    }
//A1.eliminarPares() : Método que elimina los elementos pares
del árbol A1.
    public void eliminarPares(){
        eliminarPares(raiz);
    }

     public Nodo eliminarPares(Nodo p)
    {
       if(p==null)
            return null;
         if(p.ele%2==0)
         {
            return eliminar(p.ele,p);
         }
         p.izq=eliminarHojas(p.izq);
         p.der=eliminarHojas(p.der);
         return p;
    }
//A1.eliminar(L1) : Método que elimina los elementos de la
lista L1 que se encuentran en el árbol A1.
     public void eliminar(List<Integer> L1) {
        for (Integer elem : L1)
```

```
            raiz = eliminar(elem, raiz);
    }

//A1.eliminarMenor(): Método que elimina el elemento menor
del árbol A1.
       public void eliminarMenor() {
        if (raiz != null)
            raiz = eliminarMenor(raiz);
    }

    private Nodo eliminarMenor(Nodo p) {
        if (p.izq == null)
            return p.der;
        p.izq = eliminarMenor(p.izq);
        return p;
    }
//A1.eliminarMayor(): Método que elimina el elemento mayor
del árbol A1.
     public void eliminarMayor() {
        if (raiz != null)
            raiz = eliminarMayor(raiz);
    }

    private Nodo eliminarMayor(Nodo p) {
        if (p.der == null)
            return p.izq;
        p.der = eliminarMayor(p.der);
        return p;
    }
    public static void main(String[] arg) {
        Arbol A1 = new Arbol();
        A1.insertar(14);
        A1.insertar(15);
        A1.insertar(13);
        A1.insertar(4);
        A1.insertar(5);

        A1.eliminar(5);
    }
```

CLASE NODO

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