UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO

FACULTAD DE INGENIERIA Y CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES

ESTRUCTURA DE DATOS II

CONTENIDO:

LAB-1. ÁRBOLES BINARIOS DE BÚSQUEDA.

PORCENTAJE TERMINADO: 100%

Grupo 14

Garcia Taboada Brayan Albaro

Fecha de presentación : Viernes , 03 de mayo de 2024

Fecha Presentada : Viernes , 03 de mayo de 2024

Días de Atraso : 0

ÁRBOLES BINARIOS DE BÚSQUEDA.

TRABAJO INDIVIDUAL.

Sea A1, un Árbol Binario de Búsqueda. Implementar los siguientes métodos:

1. A1.insertar(x) : Método que inserta el elemento x, en el árbol A1 en su lugar correspondiente.

2. A1.preOrden() : Método que muestra los elementos del árbol A1 en preOrden.

3. A1.inOrden() : Método que muestra los elementos del árbol A1 en inOrden.

4. A1.postOrden() : Método que muestra los elementos del árbol A1 en postOrden.

5. A1.seEncuentra(x) : Métodos lógico que devuelve True, si el elemento x, se encuentra en el árbol A1.

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

7. A1.suma() : Método que devuelve la suma de los elementos del árbol A1.

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

9. A1.mayor() : Método que devuelve el elemento mayor del árbol A1.

10. A1.cantidadTerm() : Método que devuelve la cantidad de nodos terminales del árbol A1.

11. A1.sumaPares() : Método que devuelve la suma de los elementos pares del árbol A1.

```
public class Arbol {


  public Nodo raiz;


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


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

```
public 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;
}

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);

}

/*     50
   20     80
10   40  70   90
```

```
 */
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);
}

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

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

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);
   }

}

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

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

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

```
}

private int menor(Nodo p) {
   if (p.izq == null) {
      return p.elem;
   } else {
      return menor(p.izq);
   }
}

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

private int mayor(Nodo p) {
   if (p.der == null) {
      return p.elem;
   } else {
      return menor(p.der);
   }
}

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

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

```
        return p.elem % 2 == 0
                ? sumaPares(p.izq) + suma(p.der) + p.elem
                : sumaPares(p.izq) + suma(p.der);
    }
}

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

private int cantidad(Nodo p) {
    return p == null ? 0 : cantidad(p.izq) + cantidad(p.der) + 1;
}

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

private int cantidadTerm(Nodo p) {
    return (p.izq == null && p.der == null) ? 1 : cantidadTerm(p.der) + cantidadTerm(p.izq);
}

public int cantidadPares() {
    return cantidadPares(raiz);

}

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

```
        return (p.elem % 2 == 0) ? cantidadPares(p.der) + cantidadPares(p.izq) + 1
                : cantidadPares(p.der) + cantidadPares(p.izq);
    }

}

class Nodo {

    public Nodo izq;
    public Nodo der;
    public int elem;

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

}
```