

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p align="center"><b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p><b>Aprobación:</b> 2025/05/03</p>	<p><b>Código:</b> GUIA-PRLE-001</p>	<p align="right"><b>Página:</b> 1</p>

## INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	Programacion Web 2				
TÍTULO DE LA PRÁCTICA:	<i>Laboratorio 06</i>				
NÚMERO DE PRÁCTICA:	6	AÑO LECTIVO:	2025	NRO. SEMESTRE:	1
FECHA DE PRESENTACIÓN	21/05/2024	Repositorio	<a href="https://github.com/code50/82924112.git">https://github.com/code50/82924112.git</a>		
INTEGRANTE (s): Silva Pino Jesus Francisco				NOTA:	
DOCENTE(s): Edson Luque Mamani					

SOLUCIÓN Y RESULTADOS
<p><b>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</b></p> <p><b>EJERCICIOS PROPUESTOS</b></p> <p>Elabore un informe implementando Arboles AVL con toda la lista de operaciones :</p> <ul style="list-style-type: none"> <li>• search(),</li> <li>• getMin(),</li> <li>• getMax(),</li> <li>• parent(),</li> <li>• son(),</li> <li>• insert()</li> </ul> <p>INPUT: Una sólo palabra en mayúsculas.</p> <p>OUTPUT: Se debe construir el árbol AVL considerando el valor decimal de su código ascii.</p>

Clase Nodo.java

```
class Node {  
  
    int value;  
    Node left;  
    Node right;  
    int height;  
  
    Node(int value) {  
        this.value = value;  
        this.height = 1; // La altura de un nuevo nodo es siempre 1  
    }  
}
```

Clase AVLtree.java

```
public class AVLTree {  
  
    private Node root;  
  
    // Obtener la altura de un nodo  
    private int height(Node node) {  
        if (node == null) {  
            return 0;  
        }  
        return node.height;  
    }  
  
    // Actualizar la altura de un nodo  
    private void updateHeight(Node node) {  
        if (node != null) {  
            node.height = 1 + Math.max(height(node.left), height(node.right));  
        }  
    }  
  
    // Obtener el factor de equilibrio de un nodo  
    private int getBalanceFactor(Node node) {  
        if (node == null) {  
            return 0;  
        }  
        return height(node.left) - height(node.right);  
    }  
}
```

```
// --- ROTACIONES ---
private Node rightRotate(Node y) {
Node x = y.left;
Node T2 = x.right;

// Realizar rotación
x.right = y;
y.left = T2;

// Actualizar alturas
updateHeight(y);
updateHeight(x);

return x; // Nueva raíz
}

private Node leftRotate(Node x) {
Node y = x.right;
Node T2 = y.left;

// Realizar rotación
y.left = x;
x.right = T2;

// Actualizar alturas
updateHeight(x);
updateHeight(y);

return y; // Nueva raíz
}

// --- OPERACIÓN DE INSERCIÓN ---
public void insert(int value) {
root = insert(root, value);
}

private Node insert(Node node, int value) {
// 1. Inserción normal en un Árbol Binario de Búsqueda
if (node == null) {
return new Node(value);
}

if (value < node.value) {
```

```
node.left = insert(node.left, value);
} else if (value > node.value) {
node.right = insert(node.right, value);
} else {
// Valores duplicados no son permitidos
return node;
}

// 2. Actualizar la altura del nodo actual
updateHeight(node);

// 3. Obtener el factor de equilibrio para verificar si el árbol se desbalanceó
int balance = getBalanceFactor(node);

// 4. Si el nodo está desbalanceado, realizar rotaciones

// Caso Izquierda-Izquierda
if (balance > 1 && value < node.left.value) {
return rightRotate(node);
}

// Caso Derecha-Derecha
if (balance < -1 && value > node.right.value) {
return leftRotate(node);
}

// Caso Izquierda-Derecha
if (balance > 1 && value > node.left.value) {
node.left = leftRotate(node.left);
return rightRotate(node);
}

// Caso Derecha-Izquierda
if (balance < -1 && value < node.right.value) {
node.right = rightRotate(node.right);
return leftRotate(node);
}

// Si no hay desbalance, retornar el nodo sin cambios
return node;
}

// --- OPERACIONES DE BÚSQUEDA Y CONSULTA ---
```

```
// search()
public boolean search(int value) {
    return search(root, value) != null;
}

private Node search(Node node, int value) {
    if (node == null || node.value == value) {
        return node;
    }
    if (value < node.value) {
        return search(node.left, value);
    }
    return search(node.right, value);
}

// getMin()
public int getMin() {
    if (root == null) {
        throw new IllegalStateException("El árbol está vacío.");
    }
    Node minNode = getMin(root);
    return minNode.value;
}

private Node getMin(Node node) {
    Node current = node;
    while (current.left != null) {
        current = current.left;
    }
    return current;
}

// getMax()
public int getMax() {
    if (root == null) {
        throw new IllegalStateException("El árbol está vacío.");
    }
    Node maxNode = getMax(root);
    return maxNode.value;
}

private Node getMax(Node node) {
    Node current = node;
    while (current.right != null) {
        current = current.right;
    }
}
```

```
}  
return current;  
}  
  
// parent()  
public Integer parent(int value) {  
    Node parentNode = parent(root, value);  
    return (parentNode != null) ? parentNode.value : null;  
}  
  
private Node parent(Node node, int value) {  
    if (node == null || root.value == value) {  
        return null; // El nodo no existe o es la raíz  
    }  
  
    if ((node.left != null && node.left.value == value) ||  
        (node.right != null && node.right.value == value)) {  
        return node;  
    }  
  
    if (value < node.value) {  
        return parent(node.left, value);  
    } else {  
        return parent(node.right, value);  
    }  
}  
  
// son()  
public String son(int value) {  
    if (root == null || root.value == value) {  
        return "El nodo es la raíz, no tiene padre.";  
    }  
    Node parentNode = parent(root, value);  
    if (parentNode == null) {  
        return "El nodo no existe en el árbol.";  
    }  
    if (parentNode.left != null && parentNode.left.value == value) {  
        return "Es hijo izquierdo.";  
    } else {  
        return "Es hijo derecho.";  
    }  
}  
  
// Método para imprimir el árbol (In-Order) para verificación  
public void printInOrder() {
```

```
printInOrder(root);  
System.out.println();  
}
```

```
private void printInOrder(Node node) {  
if (node != null) {  
printInOrder(node.left);  
System.out.print((char)node.value + " ");  
printInOrder(node.right);  
}  
}  
}
```

Clase LaboratorioAVL.java

```
public class LaboratorioAVL {
```

```
public static void main(String[] args) {  
ArbolAVL arbolAVL = new ArbolAVL();  
String palabra = "ESTRUCTURA";
```

```
System.out.println("INPUT: " + palabra);  
System.out.println("Construyendo Árbol AVL con los valores ASCII...");
```

```
for (char caracter : palabra.toCharArray()) {  
System.out.println("Insertando: " + caracter);  
arbolAVL.insertar(caracter);  
}
```

```
System.out.println("\n--- ÁRBOL CONSTRUIDO ---");  
System.out.print("Recorrido In-Order (valores ordenados: ");  
arbolAVL.imprimirEnOrden();
```

```
System.out.println("\n--- PROBANDO OPERACIONES ---");
```

```
// Prueba de buscar()  
System.out.println("\n1. Operación buscar():");  
System.out.println("¿Existe la letra 'R'? " + arbolAVL.buscar('R'));  
System.out.println("¿Existe la letra 'Z'? " + arbolAVL.buscar('Z'));
```

```
// Prueba de obtenerMinimo()  
System.out.println("\n2. Operación obtenerMinimo():");  
System.out.println("La letra con el menor valor en el árbol es: " + (char)valorMinimo + "");
```

```
// Prueba de obtenerMaximo()  
System.out.println("\n3. Operación obtenerMaximo():");  
System.out.println("La letra con el mayor valor en el árbol es: " + (char)valorMaximo + "");
```

```
// Prueba de padre()  
System.out.println("\n4. Operación padre():");  
System.out.println("El padre del nodo 67 (C) es: " + arbolAVL.padre(67));  
System.out.println("El padre del nodo 84 (T) es: " + arbolAVL.padre(84));  
System.out.println("El padre del nodo 83 (S, la raíz) es: " + arbolAVL.padre(83));
```

```
// Prueba de hijo()  
System.out.println("\n5. Operación hijo():");  
System.out.println("El nodo 67 (C) es: " + arbolAVL.hijo(67));  
System.out.println("El nodo 85 (U) es: " + arbolAVL.hijo(85));  
System.out.println("El nodo 83 (S) es: " + arbolAVL.hijo(83));  
}  
}
```

Luego, pruebe todas sus operaciones implementadas.



PROBLEMAS 5 SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS 1

```
$ /usr/bin/env /opt/jdk/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /home/ubuntu/.vscode-remote/data/User/workspaceStorage/-2f00f915/redhat.java/jdt_ws/829241  
12_3dd81b53/bin LaboratorioAVL
```

INPUT: ESTRUCTURA

Construyendo Árbol AVL con los valores ASCII...

Insertando: E (69)

Insertando: S (83)

Insertando: T (84)

Insertando: R (82)

Insertando: U (85)

Insertando: C (67)

Insertando: T (84)

Insertando: U (85)

Insertando: R (82)

Insertando: A (65)

--- ÁRBOL CONSTRUIDO ---

Recorrido In-Order (valores ordenados): 65 67 69 82 83 84 85

--- PROBANDO OPERACIONES ---

1. Operación search():

¿Existe el valor 82 (R)? true

¿Existe el valor 90 (Z)? false

2. Operación getMin():

El valor mínimo en el árbol es: 65 (Corresponde a 'A')

3. Operación getMax():

El valor máximo en el árbol es: 85 (Corresponde a 'U')

4. Operación parent():

El padre del nodo 67 (C) es: 69

El padre del nodo 84 (T) es: 83

El padre del nodo 83 (S, la raíz) es: null



5. Operación son():

El nodo 67 (C) es: Es hijo izquierdo.

El nodo 85 (U) es: Es hijo derecho.

El nodo 83 (S) es: El nodo es la raíz, no tiene padre.

\$ ^C

	<p align="center"> <b>UNIVERSIDAD NACIONAL DE SAN AGUSTIN</b>  <b>FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y</b>  <b>SERVICIOS</b>  <b>ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</b> </p>	
<b>Formato:</b> Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
<b>Aprobación:</b> 2025/05/03	<b>Código:</b> GUIA-PRLE-001	<b>Página:</b> 10

<b>II. SOLUCIÓN DEL CUESTIONARIO</b>
<b>III. CONCLUSIONES</b>

<b>RETROALIMENTACIÓN GENERAL</b>

<b>REFERENCIAS Y BIBLIOGRAFÍA</b>