
	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Forma :o: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLD-001	Página: 1

GUÍA DE LABORATORIO

(formato docente)

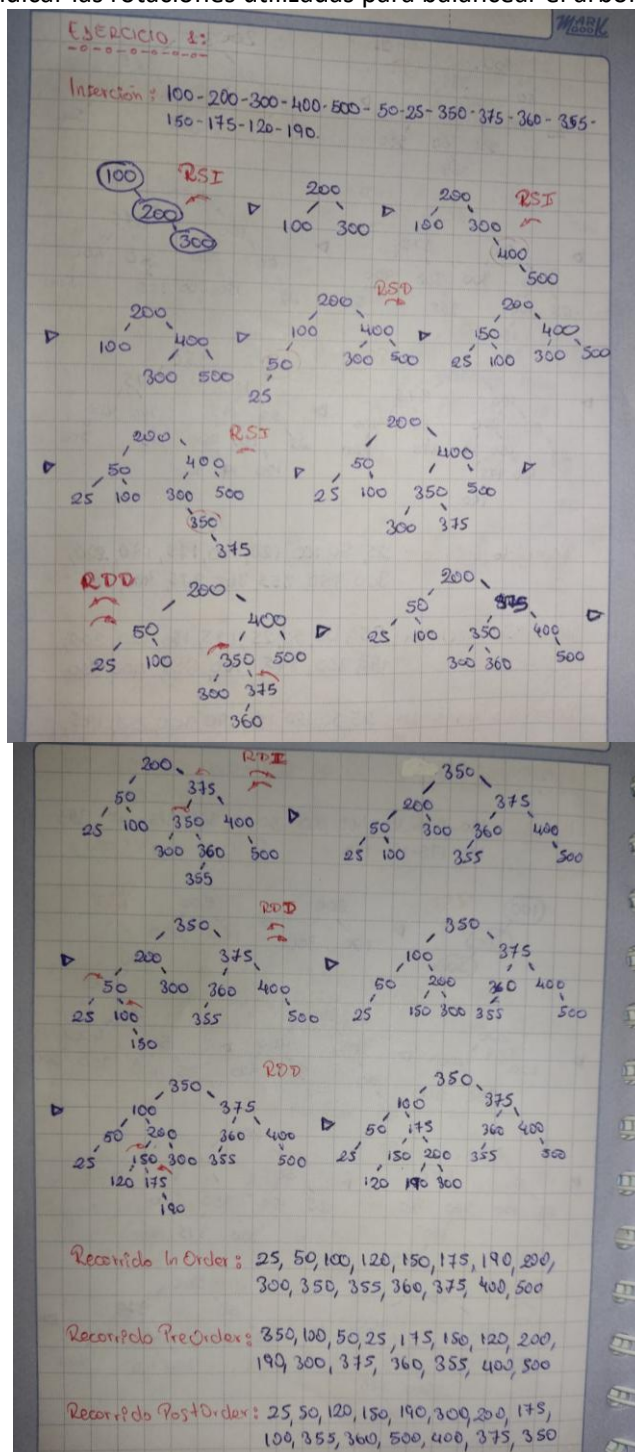
INFORMACIÓN BÁSICA					
ASIGNATURA:	ESTRUCTURA DE DATOS Y ALGORITMOS				
TÍTULO DE LA PRÁCTICA:	ÁRBOL BINARIO AVL BALANCEADO				
NÚMERO DE PRÁCTICA:	07	AÑO LECTIVO:	2025 – A	NRO. SEMESTRE:	TERCERO III
TIPO DE PRÁCTICA:	INDIVIDUAL	X			
	GRUPAL	—	MÁXIMO DE ESTUDIANTES	01	
FECHA INICIO:	23/06/2025	FECHA FIN:	27/06/2025	DURACIÓN:	90 minutos.
RECURSOS A UTILIZAR: <ul style="list-style-type: none"> Repositorio GITHUB: https://github.com/JosueClaudioQP/EDA-Lab Lenguaje de Programación Java. Ide Java Eclipse/Visual Studio Code. 					
DOCENTE(s): <ul style="list-style-type: none"> Mg. Ing. Rene Alonso Nieto Valencia. 					
ALUMNO: <ul style="list-style-type: none"> Quispe Paucar, Josué Claudio 					

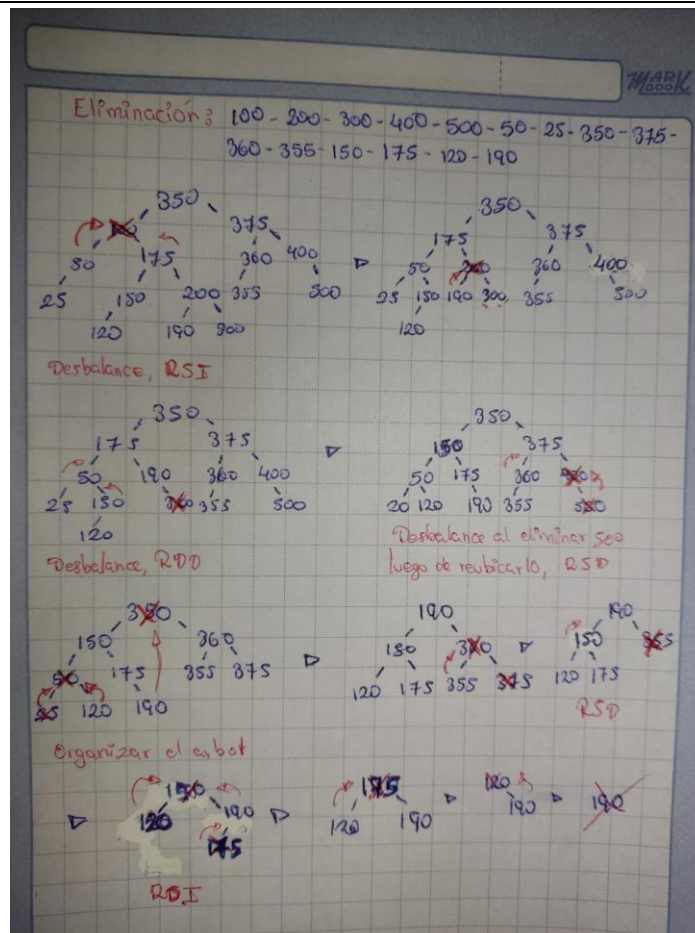
OBJETIVOS/TEMAS Y COMPETENCIAS
OBJETIVOS: <ul style="list-style-type: none"> Aprenda Árboles AVL. Aplicar conceptos elementales de programación a resolver utilizando POO en problemas de algoritmos. Desarrollar pruebas.
TEMAS: <ul style="list-style-type: none"> Árbol AVL Operaciones AVL Balanceo y rotaciones

I. PROBLEMAS PROPUESTOS:

Ejercicio 1: Mediante la dinámica de un árbol AVL realizar lo siguiente: o Inserción de los siguientes nodos: 100 – 200 – 300 – 400 – 500 - 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190. o

Mostrar los recorridos en inOrder, preOrder y postOrder, del árbol resultante de la inserción. o Mostrar el árbol resultante de eliminar los nodos: 100 – 200 – 300 – 400 – 500 – 50 – 25 – 350 – 375 – 360 – 355 – 150 – 175 – 120 – 190. o Mostrar el paso a paso de la inserción y eliminación de cada nodo. o Indicar las rotaciones utilizadas para balancear el árbol resultante.





Ejercicio 2: Implementar un árbol AVL: o Implementar todas las operaciones: `destroy()`, `isEmpty()`, `insert(x)`, `remove(x)`, `search(x)`, `Min()`, `Max()`, `Predecesor()`, `Sucesor()`, `Recorridos: InOrder()`, `PostOrder()`, `PreOrder()`, `balancearIzquierda()`, `balancearDerecha()`, `rotacionSimpleIzquierda()`, `rotacionSimpleDerecha()`. o Implementar una clase Test para probar los métodos y mostrar los resultados. utilizando clases y métodos genéricos, utilizando un menú de opciones para todas las operaciones del árbol AVL.

Para empezar la implementación de nuestro árbol AVL, primero implementamos nuestro NodeAVL

```

1 package ProblemasPropuestos.Ejercicio2;
2
3 public class NodeAVL<T> {
4     public T data;
5     public NodeAVL<T> left;
6     public NodeAVL<T> right;
7     public int height;
8
9     public NodeAVL(T data){
10         this.data = data;
11         this.height = 1;
12     }
13 }

```

El nodo es idéntico al Nodo usado en el árbol BST, sin embargo, este tiene un atributo `height`, el cual es un entero que almacenará la altura del árbol (servirá para el balanceo)

```
1 package ProblemasPropuestos.Ejercicio2;
2
3 public class AVLTree<T extends Comparable<T>> {
4     private NodeAVL<T> root;
5
6     public NodeAVL<T> getRoot() {
7         return root;
8     }
9
10    public void destroy() {
11        root = null;
12    }
13
14    public boolean isEmpty() {
15        return root == null;
16    }
17
18    public void insert(T data) {
19        root = insert(root, data);
20    }
21
22    public void remove(T data) {
23        root = remove(root, data);
24    }
25
26    public boolean search(T data) {
27        return search(root, data) != null;
28    }
29
30    public T min() {
31        NodeAVL<T> node = minValueNode(root);
32        return (node != null) ? node.data : null;
33    }
34
35    public T max() {
36        NodeAVL<T> node = maxValueNode(root);
37        return (node != null) ? node.data : null;
38    }
39 }
```

Se crean el getter para poder acceder a la raíz. Luego creamos todos los métodos a usar los cuales invocan otros métodos que veremos implementados más adelante.

```
125 private NodeAVL<T> minValueNode(NodeAVL<T> node) {
126     while (node != null && node.left != null)
127         node = node.left;
128     return node;
129 }
130
131 private NodeAVL<T> maxValueNode(NodeAVL<T> node) {
132     while (node != null && node.right != null)
133         node = node.right;
134     return node;
135 }
```

En el caso de Min y Max, estos instancian un nodo y lo analizan tanto de izquierda como derecha, dandonos el mínimo y máximo del árbol.

```

40 public T predecesor(T data) {
41     NodeAVL<T> curr = root, pred = null;
42     while (curr != null) {
43         if (data.compareTo(curr.data) > 0) {
44             pred = curr;
45             curr = curr.right;
46         } else {
47             curr = curr.left;
48         }
49     }
50     return (pred != null) ? pred.data : null;
51 }
52
53 public T sucesor(T data) {
54     NodeAVL<T> curr = root, succ = null;
55     while (curr != null) {
56         if (data.compareTo(curr.data) < 0) {
57             succ = curr;
58             curr = curr.left;
59         } else {
60             curr = curr.right;
61         }
62     }
63     return (succ != null) ? succ.data : null;
64 }

```

Predecesor encuentra el mayor valor menor que el dato dado como parámetro.

Sucesor encuentra el menor valor mayor que el dato dado como parámetro.

```

66 public void inOrder() {
67     inOrder(root);
68     System.out.println();
69 }
70
71 public void preOrder() {
72     preOrder(root);
73     System.out.println();
74 }
75
76 public void postOrder() {
77     postOrder(root);
78     System.out.println();
79 }
80
81 private void inOrder(NodeAVL<T> node) {
82     if (node != null) {
83         inOrder(node.left);
84         System.out.print(node.data + " ");
85         inOrder(node.right);
86     }
87 }
88
89 private void preOrder(NodeAVL<T> node) {
90     if (node != null) {
91         System.out.print(node.data + " ");
92         preOrder(node.left);
93         preOrder(node.right);
94     }
95 }
96
97 private void postOrder(NodeAVL<T> node) {
98     if (node != null) {
99         postOrder(node.left);
100        postOrder(node.right);
101        System.out.print(node.data + " ");
102    }
103 }

```

Los métodos inorden, preorden y postorden imprimen los datos según el tipo de orden gracias a la recursividad.

```
81     private NodeAVL<T> insert(NodeAVL<T> node, T data) {
82         if (node == null) return new NodeAVL<>(data);
83
84         if (data.compareTo(node.data) < 0)
85             node.left = insert(node.left, data);
86         else if (data.compareTo(node.data) > 0)
87             node.right = insert(node.right, data);
88         else
89             return node;
90
91         updateHeight(node);
92         return balance(node);
93     }
```

Este método ingresa un nodo, ya sea a la izquierda o derecha (excepto duplicados) y realiza el balanceo de ser necesario.

```
95     private NodeAVL<T> remove(NodeAVL<T> node, T data) {
96         if (node == null) return null;
97
98         if (data.compareTo(node.data) < 0)
99             node.left = remove(node.left, data);
100        else if (data.compareTo(node.data) > 0)
101            node.right = remove(node.right, data);
102        else {
103            if (node.left == null || node.right == null) {
104                node = (node.left != null) ? node.left : node.right;
105            } else {
106                NodeAVL<T> temp = minValueNode(node.right);
107                node.data = temp.data;
108                node.right = remove(node.right, temp.data);
109            }
110        }
111
112        if (node == null) return null;
113
114        updateHeight(node);
115        return balance(node);
116    }
```

Elimina un nodo recursivamente, luego actualiza la altura y rebalancea.

```
118     private NodeAVL<T> search(NodeAVL<T> node, T data) {
119         if (node == null || node.data.equals(data)) return node;
120         if (data.compareTo(node.data) < 0)
121             return search(node.left, data);
122         return search(node.right, data);
123     }
```

Realiza la búsqueda binaria comparando el nodo con el dato ingresado.

```

161     private void updateHeight(NodeAVL<T> node) {
162         node.height = 1 + Math.max(height(node.left), height(node.right));
163     }
164
165     private int height(NodeAVL<T> node) {
166         return node != null ? node.height : 0;
167     }
168
169     private int getBalance(NodeAVL<T> node) {
170         return (node != null) ? height(node.left) - height(node.right) : 0;
171     }
172
173     private NodeAVL<T> balance(NodeAVL<T> node) {
174         int balance = getBalance(node);
175
176         if (balance > 1) {
177             if (getBalance(node.left) < 0)
178                 node.left = rotateLeft(node.left);
179             return rotateRight(node);
180         }
181
182         if (balance < -1) {
183             if (getBalance(node.right) > 0)
184                 node.right = rotateRight(node.right);
185             return rotateLeft(node);
186         }
187
188         return node;
189     }

```

UpdateHeight actualiza la altura del nodo actual, height devuelve la altura de un nodo (0 si es null), getBalance calcula el factor del balance del nodo y luego balance aplica las rotaciones necesarias si el nodo resulta estar en desbalance (ya sea rotación simple o doble).

```

191     public NodeAVL<T> rotateLeft(NodeAVL<T> x) {
192         NodeAVL<T> y = x.right;
193         NodeAVL<T> T2 = y.left;
194
195         y.left = x;
196         x.right = T2;
197
198         updateHeight(x);
199         updateHeight(y);
200
201         return y;
202     }
203
204     public NodeAVL<T> rotateRight(NodeAVL<T> y) {
205         NodeAVL<T> x = y.left;
206         NodeAVL<T> T2 = x.right;
207
208         x.right = y;
209         y.left = T2;
210
211         updateHeight(y);
212         updateHeight(x);
213
214         return x;
215     }
216 }
217

```

Estos métodos reemplazan los valores de los nodos para que se puedan realizar las rotaciones.

Finalmente, creamos nuestro TEST para hacer las pruebas necesarias

```
ProblemasPropuestos > Ejercicio2 > Test.java > Test > main(String[])
1 package ProblemasPropuestos.Ejercicio2;
2 import java.util.Scanner;
3
4 import ProblemasPropuestos.Ejercicio3.VisualizarAVL;
5
6 public class Test {
7     Run | Debug
8     public static void main(String[] args) {
9         AVLTree<Integer> tree = new AVLTree<>();
10        Scanner sc = new Scanner(System.in);
11        int opc, valor;
12
13        for(int i = 0; i < 15; i++){
14            tree.insert(i+1);
15        }
16
17        do {
18            System.out.println(x:"\n--- MENÚ ÁRBOL AVL ---");
19            System.out.println(x:"1. Insertar");
20            System.out.println(x:"2. Eliminar");
21            System.out.println(x:"3. Buscar");
22            System.out.println(x:"4. Mostrar Min y Max");
23            System.out.println(x:"5. Predecesor / Sucesor");
24            System.out.println(x:"6. Recorrido InOrder");
25            System.out.println(x:"7. Recorrido PreOrder");
26            System.out.println(x:"8. Recorrido PostOrder");
27            System.out.println(x:"9. Vaciar árbol");
28            System.out.println(x:"10. ¿Está vacío?");
29            System.out.println(x:"11. Visualizar árbol (Swing)");
30            System.out.println(x:"0. Salir");
31            System.out.print(s:"Opción: ");
32            opc = sc.nextInt();
```

Antes del menú interactivo, se añaden directamente datos para analizar rápidamente nuestro árbol.

Se crea el menú de consola.

```
33 switch (opc) {
34     case 1:
35         System.out.print(s:"Valor a insertar: ");
36         tree.insert(sc.nextInt());
37         break;
38     case 2:
39         System.out.print(s:"Valor a eliminar: ");
40         tree.remove(sc.nextInt());
41         break;
42     case 3:
43         System.out.print(s:"Valor a buscar: ");
44         System.out.println(tree.search(sc.nextInt()) ? "Encontrado" : "No encontrado");
45         break;
46     case 4:
47         System.out.println("Mínimo: " + tree.min());
48         System.out.println("Máximo: " + tree.max());
49         break;
50     case 5:
51         System.out.print(s:"Valor de referencia: ");
52         valor = sc.nextInt();
53         System.out.println("Predecesor: " + tree.predecesor(valor));
54         System.out.println("Sucesor: " + tree.sucesor(valor));
55         break;
56     case 6:
57         tree.inOrder();
58         break;
59     case 7:
60         tree.preOrder();
61         break;
```



```

62         case 8:
63             tree.postOrder();
64             break;
65         case 9:
66             tree.destroy();
67             System.out.println(x:"Árbol eliminado.");
68             break;
69         case 10:
70             System.out.println(tree.isEmpty() ? "Árbol vacío" : "Árbol no vacío");
71             break;
72         case 11:
73             VisualizarAVL.mostrar(tree);
74             break;
75     }
76 } while (opc != 0);
77 sc.close();
78 }
79 }
80

```

Se crean los casos según la opción ingresada por consola junto con mensajes de confirmación.

Analizando las pruebas:

```

PS C:\Users\JOSUE\Documents\UNSA\5to SEMESTRE\EDA\EDA Lab\Laboratorio7>
.\JOSUE\AppData\Roaming\Code\User\workspaceStorage\adef1a6dcac6dffa46756af

--- MENÚ ÁRBOL AVL ---
1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir
Opción:

```

<pre> Opción: 6 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 --- MENÚ ÁRBOL AVL --- 1. Insertar 2. Eliminar 3. Buscar 4. Mostrar Min y Max 5. Predecesor / Sucesor 6. Recorrido InOrder 7. Recorrido PreOrder 8. Recorrido PostOrder 9. Vaciar árbol 10. ¿Está vacío? 11. Visualizar árbol (Swing) 0. Salir Opción: █ </pre>	<pre> Opción: 7 8 4 2 1 3 6 5 7 12 10 9 11 14 13 15 --- MENÚ ÁRBOL AVL --- 1. Insertar 2. Eliminar 3. Buscar 4. Mostrar Min y Max 5. Predecesor / Sucesor 6. Recorrido InOrder 7. Recorrido PreOrder 8. Recorrido PostOrder 9. Vaciar árbol 10. ¿Está vacío? 11. Visualizar árbol (Swing) 0. Salir Opción: █ </pre>	<pre> Opción: 8 1 3 2 5 7 6 4 9 11 10 13 15 14 12 8 --- MENÚ ÁRBOL AVL --- 1. Insertar 2. Eliminar 3. Buscar 4. Mostrar Min y Max 5. Predecesor / Sucesor 6. Recorrido InOrder 7. Recorrido PreOrder 8. Recorrido PostOrder 9. Vaciar árbol 10. ¿Está vacío? 11. Visualizar árbol (Swing) 0. Salir Opción: █ </pre>
--	--	--

<pre> Opción: 4 Mínimo: 1 Máximo: 15 --- MENÚ ÁRBOL AVL --- 1. Insertar 2. Eliminar 3. Buscar 4. Mostrar Min y Max 5. Predecesor / Sucesor 6. Recorrido InOrder 7. Recorrido PreOrder 8. Recorrido PostOrder 9. Vaciar árbol 10. ¿Está vacío? 11. Visualizar árbol (Swing) 0. Salir Opción: █ </pre>	<pre> Opción: 10 Árbol no vacío --- MENÚ ÁRBOL AVL --- 1. Insertar 2. Eliminar 3. Buscar 4. Mostrar Min y Max 5. Predecesor / Sucesor 6. Recorrido InOrder 7. Recorrido PreOrder 8. Recorrido PostOrder 9. Vaciar árbol 10. ¿Está vacío? 11. Visualizar árbol (Swing) 0. Salir Opción: █ </pre>
---	--

Opción: 1
 Valor a insertar: 16

--- MENÚ ÁRBOL AVL ---

1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir

Opción: █

Opción: 1
 Valor a insertar: 17

--- MENÚ ÁRBOL AVL ---

1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir

Opción: █

Opción: 6
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

--- MENÚ ÁRBOL AVL ---

1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir

Opción: █

Opción: 2
 Valor a eliminar: 10

--- MENÚ ÁRBOL AVL ---

1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir

Opción: █

Opción: 2
 Valor a eliminar: 15

--- MENÚ ÁRBOL AVL ---

1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir

Opción: █

Opción: 6
 1 2 3 4 5 6 7 8 9 11 12 13 14 16 17

--- MENÚ ÁRBOL AVL ---

1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir

Opción: █

Valor a buscar: 9
 Encontrado

--- MENÚ ÁRBOL AVL ---

1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir

Opción: █

Opción: 3
 Valor a buscar: 18
 No encontrado

--- MENÚ ÁRBOL AVL ---

1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir

Opción: █

Opción: 5
 Valor de referencia: 11
 Predecesor: 9
 Sucesor: 12

--- MENÚ ÁRBOL AVL ---

1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir

Opción: █

Opción: 10
 Árbol vacío

--- MENÚ ÁRBOL AVL ---

1. Insertar
2. Eliminar
3. Buscar
4. Mostrar Min y Max
5. Predecesor / Sucesor
6. Recorrido InOrder
7. Recorrido PreOrder
8. Recorrido PostOrder
9. Vaciar árbol
10. ¿Está vacío?
11. Visualizar árbol (Swing)
0. Salir

Opción: █

Opción: 9
 Árbol eliminado.

Ejercicio 3: Implementar un método para graficar el árbol AVL resultante, mostrando todos sus nodos, sus aristas izquierda y derecha utilizando clases y métodos genéricos, utilizar la librería Graph Stream o similar.

Para graficar nuestro arbol con interfaz, se usó la librería Swing, propio de java.

```
1 package ProblemasPropuestos.Ejercicio3;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 import ProblemasPropuestos.Ejercicio2.NodeAVL;
7
8 public class AVLVisualizerPanel<T extends Comparable<T>> extends JPanel {
9     private NodeAVL<T> root;
10
11     public AVLVisualizerPanel(NodeAVL<T> root) {
12         this.root = root;
13         setPreferredSize(new Dimension(width:800, height:600));
14         setBackground(Color.WHITE);
15     }
16 }
```

Se importan los componentes swing y awt para los graficos, colores, etc.

La clase extiende JPanel para dibujar el arbol, se guarda la raíz y el constructor sirve para crear el panel incluyendo el nodo raíz.

```
17 @Override
18 protected void paintComponent(Graphics g) {
19     super.paintComponent(g);
20     drawTree(g, root, getWidth() / 2, y:40, getWidth() / 4);
21 }
22 }
```

Este método se llama automáticamente por Swing cuando el panel necesita ser dibujado o actualizado. Se llama al método drawTree para dibujar el árbol.

```
23 private void drawTree(Graphics g, NodeAVL<T> node, int x, int y, int xOffset) {
24     if (node == null) return;
25
26     g.setColor(Color.BLACK);
27     g.fillOval(x - 15, y - 15, width:30, height:30);
28     g.setColor(Color.WHITE);
29     g.drawString(node.data.toString(), x - 7, y + 5);
30
31     g.setColor(Color.BLACK);
32     if (node.left != null) {
33         int childX = x - xOffset;
34         int childY = y + 60;
35         g.drawLine(x, y, childX, childY);
36         drawTree(g, node.left, childX, childY, xOffset / 2);
37     }
38
39     if (node.right != null) {
40         int childX = x + xOffset;
41         int childY = y + 60;
42         g.drawLine(x, y, childX, childY);
43         drawTree(g, node.right, childX, childY, xOffset / 2);
44     }
45 }
```

Se crea el metodo recursivo para dibujar el arbol. "X", "y" y xOffset sirven para dar distancia y coordenadas. Se dibuja, colorea el nodo y se le da su valor, así sucesivamente por los hijos tanto derecho como izquierdo.

```

47     public void updateTree(NodeAVL<T> newRoot) {
48         this.root = newRoot;
49         repaint();
50     }

```

Finalmente este método actualiza el árbol. Permitiendo redibujar el árbol cuando se inserta o elimina un nodo.

Finalmente se crea la clase VisualizarAVL.

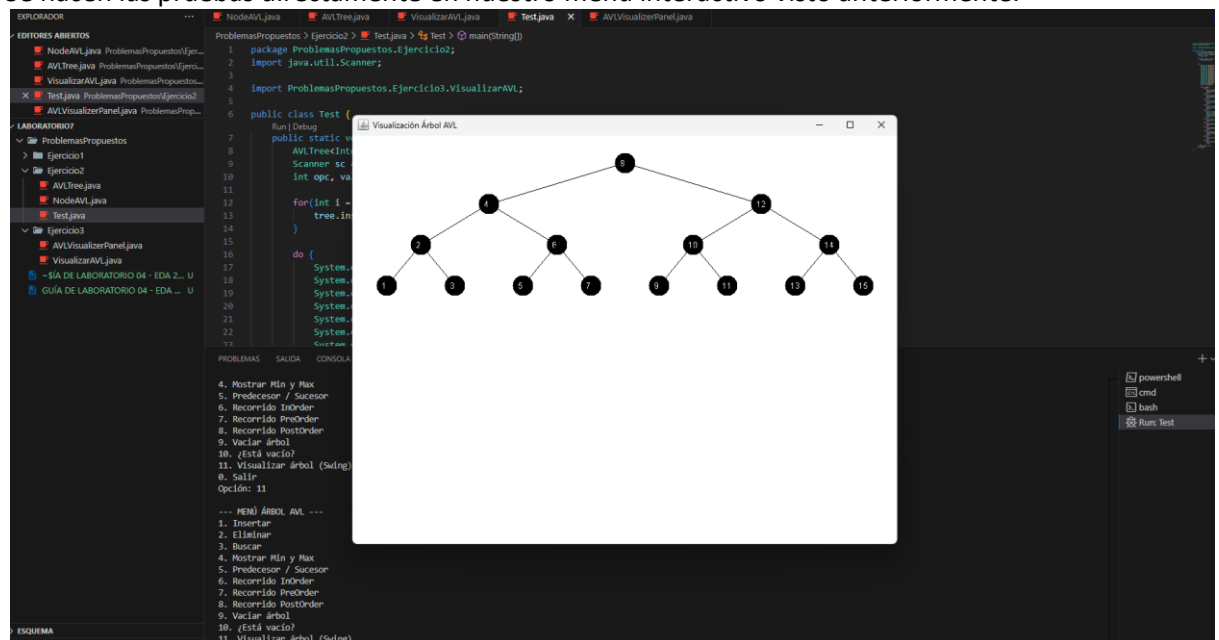
```

1  package ProblemasPropuestos.Ejercicio3;
2
3  import javax.swing.*;
4
5  import ProblemasPropuestos.Ejercicio2.AVLTree;
6
7  public class VisualizarAVL {
8      public static <T extends Comparable<T>> void mostrar(AVLTree<T> tree) {
9          JFrame frame = new JFrame(title:"Visualización Árbol AVL");
10         AVLVisualizerPanel<T> panel = new AVLVisualizerPanel<>(tree.getRoot());
11
12         frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
13         frame.getContentPane().add(panel);
14         frame.pack();
15         frame.setLocationRelativeTo(c:null);
16         frame.setVisible(b:true);
17     }
18 }
19

```

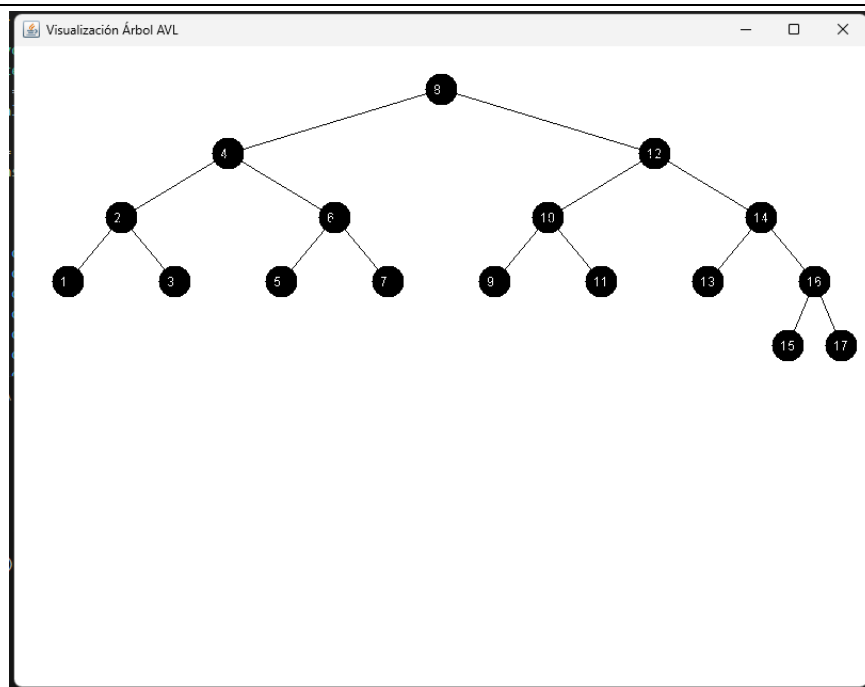
Esta clase contiene un método estático que permite visualizar un árbol AVL usando una ventana Swing (JFrame). Es una utilidad para abrir una ventana emergente con el dibujo del árbol.

Se hacen las pruebas directamente en nuestro menú interactivo visto anteriormente.

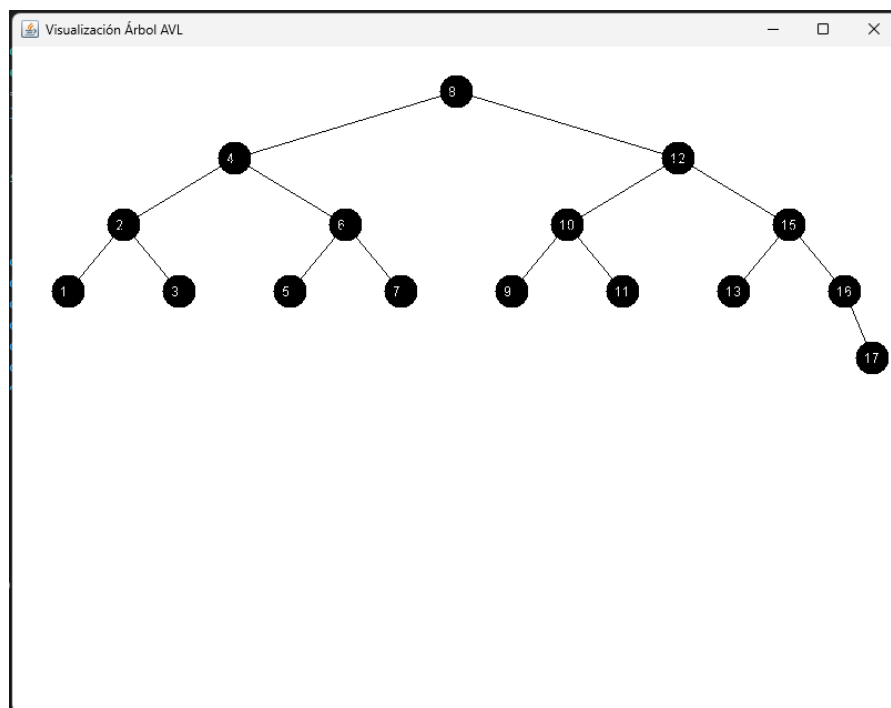


The screenshot shows an IDE with several files open: NodeAVL.java, AVLTree.java, VisualizarAVL.java, Test.java, and AVLVisualizerPanel.java. The main window displays a binary tree structure with 15 nodes. The tree is rooted at node 8. The left child of 8 is node 4, and the right child is node 12. Node 4 has children 2 and 6. Node 12 has children 10 and 14. Node 2 has children 1 and 3. Node 6 has children 5 and 7. Node 10 has children 9 and 11. Node 14 has children 13 and 15. The tree is balanced, indicating it is an AVL tree.



Se muestra el grafico con los nodos existentes



Se añaden 16 y 17 (hubo balanceo).



Se elimina 14, al ser reemplazado por 13 se realiza la rotacion y balanceo.

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Forma :o: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLD-001</p>	<p>Página: 14</p>

IV. CUESTIONARIO

1. ¿Cuáles fueron las dificultades que encontraste al desarrollar los ejercicios propuestos? por ejemplo, poca documentación, complejidad del lenguaje, etc.

Escasa documentación específica, ya que, si bien existen recursos sobre árboles AVL y Swing por separado, hay muy poca documentación clara que combine estructuras de datos con visualización gráfica en Java.

El balanceo del árbol AVL ya que al implementar correctamente las rotaciones simples y dobles (izquierda/derecha), y verificar el factor de balance en cada inserción y eliminación fue un reto. Cualquier error podía hacer que el árbol dejara de cumplir sus propiedades AVL.

2. ¿Explique cómo es el algoritmo que implemento para obtener el AVL con la librería Graph Stream? Recuerda que puede agregar operaciones sobre la clase BST.

Ya que usamos Java Swing para realizar este árbol, el algoritmo implementado combina dos partes clave: la estructura lógica del árbol AVL y su representación visual usando Swing.

Se implementó la clase AVLTree<T> que incluye:

- Inserción de nodos con recursividad.
- Cálculo del factor de balance (diferencia de alturas de los subárboles).
- Aplicación de rotaciones para mantener el balance (izquierda, derecha, doble izquierda-derecha, doble derecha-izquierda).
- Métodos auxiliares como búsqueda, mínimo, máximo, predecesor y sucesor.
- Recorridos (inorden, preorden, postorden).

Esto garantiza que el árbol se mantenga balanceado automáticamente tras cada operación.

Para mostrar el árbol gráficamente:



- Se creó una subclase de JPanel llamada AVLVisualizerPanel<T>, que dibuja el árbol recursivamente en el método paintComponent(Graphics g).
- Cada nodo se representa como un círculo con su valor dentro, y se conectan con líneas a sus hijos izquierdo y derecho.
- El dibujo se realiza con coordenadas calculadas dinámicamente para centrar el árbol y evitar solapamientos, usando una separación horizontal (xOffset) que se reduce por nivel.

Se encapsuló todo en una clase VisualizarAVL que abre un JFrame con el panel visualizador.

Este método es fácil de usar: solo se llama VisualizarAVL.mostrar(miArbolAVL); y se abre una ventana con el árbol renderizado.

V. REFERENCIAS Y BIBLIOGRAFÍA RECOMENDADAS:

- Weiss M., Data Structures & Problem Solving Using Java, 2010, Addison-Wesley.
- Weiss M., Data Structures and Algorithms Analysis in Java, 2012, Addison-Wesley.
- Cormen T., Leiserson C., Rivest R., Stein C., Introduction to Algorithms, 2022, The MIT Press
- The Java™ Tutorials - <https://docs.oracle.com/javase/tutorial/>
- Sedgewick, R., Algorithms in Java, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching, Part 5: Graph Algorithms, Addison-Wesley.
- Malik D., Data Structures Usign C++, 2003, Thomson Learning.
- Knuth D., The Art of Computer Programming, Vol. 1 y 3, Addison - Wesley.

	<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">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p align="center">Código: GUIA-PRLD-001</p>	<p align="right">Página: 15</p>

TÉCNICAS E INSTRUMENTOS DE EVALUACIÓN	
<p>TÉCNICAS: <i>Actividades Resueltas</i> <i>Ejercicios Propuestos</i></p>	<p>INSTRUMENTOS: <i>Rubricas</i></p>
<p>CRITERIOS DE EVALUACIÓN Los criterios de evaluación se encuentran en el silabo DUFA ANEXO en la sección EVOLUCIÓN CONTINUA</p>	