Taller semana 13: Arboles

**Implementación de un Árbol Trinario**

**Integrantes**: Alan Freire, Sebastián Condo, Mateo Peñaherrera y Camila Mora

**Resumen de modificaciones realizadas**

Se añadió soporte para un hijo central en el árbol, incorporando el atributo centro en la clase Nodo y adaptando la clase Arbol para gestionar esta posición al agregar nuevos nodos. Se actualizó la interfaz gráfica con un JComboBox para seleccionar la posición del nodo (izquierda, centro, derecha) y se ajustó el método de dibujo para representar visualmente el hijo central. Además, se mejoraron los mensajes de error para facilitar la identificación de problemas al añadir nodos. Estos cambios permiten gestionar y visualizar correctamente árboles con un hijo central adicional.

**Explicación de técnicas y algoritmos para el árbol trinario**

Un árbol trinario es una estructura de datos jerárquica en la que cada nodo puede tener hasta tres hijos: izquierdo, central y derecho. A continuación, se explican las técnicas y algoritmos más comunes relacionados con los árboles trinarios, desde su construcción hasta sus operaciones más relevantes:

**1. Representación del Árbol Trinario**

**Nodos:** Cada nodo contiene:

* Información (por ejemplo, un valor o etiqueta).
* Tres referencias (punteros): izquierda, centro y derecha.
* Información adicional como coordenadas (x, y) en aplicaciones gráficas.

### ****2. Construcción del Árbol Trinario****

**Creación de Nodos:** Se crean instancias de Nodo para cada valor que se desee agregar.

**Inserción:**

* Si el árbol está vacío, el nodo se convierte en la raíz.
* Si no, se especifica un nodo padre y la posición donde se añadirá el nuevo nodo (izquierda, centro o derecha).

### ****3. Recorridos en un Árbol Trinario****

Los algoritmos de recorrido en un árbol trinario son extensiones de los de un árbol binario, adaptados para tener en cuenta el hijo central.

### ****4. Operaciones Comunes****

#### **4.1 Búsqueda**

Busca un nodo específico basado en su etiqueta.

#### **4.2 Altura del Árbol**

Calcula la altura del árbol, considerando la rama más larga.

#### **4.3 Número de Nodos**

Cuenta la cantidad total de nodos en el árbol.

**Desafíos encontrados y sus soluciones**

### ****1. Complejidad Incrementada por el Hijo Central****

**Desafío:**  
A diferencia de un árbol binario, donde cada nodo tiene como máximo dos hijos, en un árbol trinario es necesario manejar un hijo central adicional, lo que complica la lógica de inserción, eliminación y recorrido.

**Solución:**

* Extender los métodos existentes de árboles binarios para incluir el manejo del hijo central. Por ejemplo, agregar una verificación explícita para el nodo central en los métodos de recorrido, dibujo e inserción.
* Usar estructuras de datos auxiliares, como listas o colas, para mantener un diseño modular y evitar la repetición de código.

### ****2. Dibujo Correcto del Árbol en Pantalla****

**Desafío:**  
Representar gráficamente el hijo central sin interferir con las posiciones de los hijos izquierdo y derecho es complicado, especialmente cuando el árbol es profundo.

**Solución:**

* Implementar un cálculo dinámico de las coordenadas basado en el nivel del nodo y la separación horizontal.
* Ajustar las posiciones de los hijos central y derecho para que se distribuyan uniformemente:
  + El hijo izquierdo se dibuja hacia la izquierda del nodo padre.
  + El hijo central, directamente abajo.
  + El hijo derecho, hacia la derecha.

### ****3. Integración con Interfaces Gráficas****

**Desafío:**  
La actualización de la representación gráfica del árbol en una interfaz (como un panel de Java) puede volverse inconsistente si los nodos no están bien sincronizados.

**Solución:**

* Mantener actualizadas las coordenadas (x, y) de cada nodo durante las operaciones de inserción y eliminación.
* Redibujar todo el árbol en lugar de intentar actualizar nodos específicos para evitar inconsistencias visuales.

**Incluya ejemplos de:**

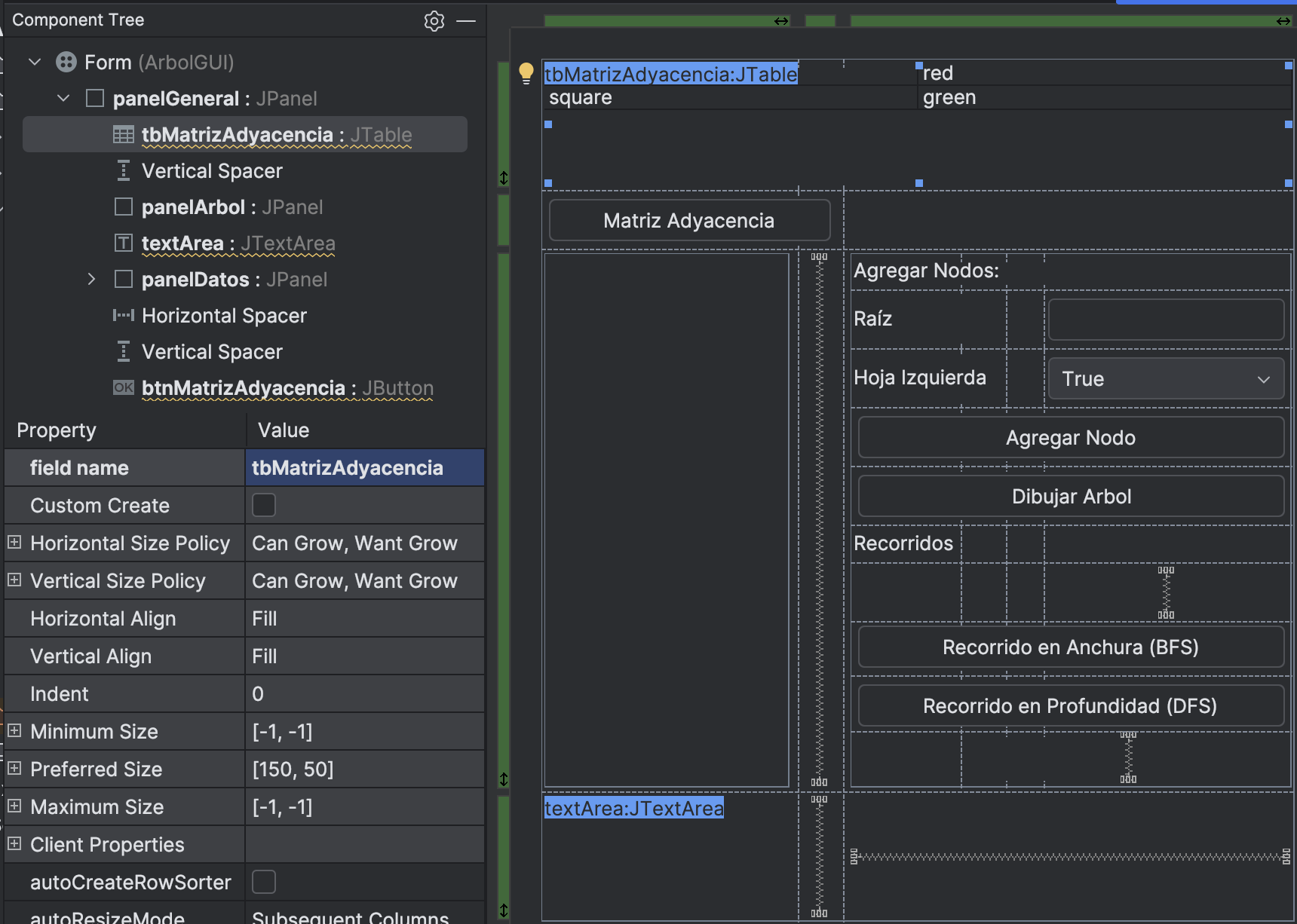
* + Inserción de nodos
  + Eliminación de nodos
  + Recorridos del árbol trinario
* import java.util.ArrayList;  
  import java.util.LinkedList;  
  import java.util.Queue;  
  import java.util.\*;  
    
  public class Arbol {  
   private Nodo raiz;  
   private ArrayList<Nodo> nodos;  
   private int numNodos;  
    
   public Arbol() {  
   raiz = null;  
   nodos = new ArrayList<>();  
   numNodos = 0;  
   }  
    
   public void anadirNodo(Nodo nodo, Nodo padre, boolean esIzq) {  
   if (padre == null) {  
   if (raiz == null) {  
   raiz= nodo;  
   } else {  
   throw new IllegalArgumentException("La raíz ya existe");  
   }  
   } else {  
   if (esIzq) {  
   if (padre.izquierda == null) {  
   padre.izquierda = nodo;  
   } else {  
   throw new IllegalArgumentException("Hoja Izq ya existe");  
   }  
   } else {  
   if (padre.derecha == null) {  
   padre.derecha = nodo;  
   } else {  
   throw new IllegalArgumentException("Hoja Der ya existe");  
   }  
   }  
   }  
   nodos.add(nodo);  
   }  
    
   public ArrayList<Nodo> getNodos() {  
   return nodos;  
   }  
    
   public Nodo getRaiz() {  
   return raiz;  
   }  
    
   public String getEtiquetaNodoSiguiente() {  
   return String.*valueOf*((char) ('A' + numNodos++));  
   }  
    
   public String bfs() {  
   if (raiz== null) return "";  
    
   StringBuilder resultado = new StringBuilder();  
   Queue<Nodo> queue = new LinkedList<>();  
   queue.add(raiz);  
    
   while (!queue.isEmpty()) {  
   Nodo nodo = queue.poll();  
   resultado.append(nodo.etiqueta).append(" ");  
   if (nodo.izquierda != null) queue.add(nodo.izquierda );  
   if (nodo.derecha != null) queue.add(nodo.derecha);  
   }  
    
   return resultado.toString().trim();  
   }  
    
   public String dfs() {  
   if (raiz== null) return "";  
    
   StringBuilder resultado = new StringBuilder();  
   Stack<Nodo> stack = new Stack<>();  
   stack.push(raiz);  
    
   while (!stack.isEmpty()) {  
   Nodo nodo = stack.pop();  
   resultado.append(nodo.etiqueta).append(" ");  
   if (nodo.derecha != null) stack.push(nodo.derecha );  
   if (nodo.izquierda != null) stack.push(nodo.izquierda);  
   }  
    
   return resultado.toString().trim();  
   }  
    
   public String preorden() {  
   return preordenImpresion(raiz).trim();  
   }  
    
   private String preordenImpresion(Nodo nodo) {  
   if (nodo == null) return "";  
   return nodo.etiqueta + " " + preordenImpresion(nodo.izquierda) + preordenImpresion(nodo.derecha);  
   }  
    
   public String inorden() {  
   return inordenImpresion(raiz).trim();  
   }  
    
   private String inordenImpresion(Nodo nodo) {  
   if (nodo == null) return "";  
   return inordenImpresion(nodo.izquierda) + nodo.etiqueta + " " + inordenImpresion(nodo.derecha);  
   }  
    
   public String postorden() {  
   return postordenImpresion(raiz).trim();  
   }  
    
   private String postordenImpresion(Nodo nodo) {  
   if (nodo == null) return "";  
   return postordenImpresion(nodo.izquierda) + postordenImpresion(nodo.derecha) + nodo.etiqueta + " ";  
   }  
    
   public Object[][] getMatrizAdyacencia() {  
   int tam = nodos.size();  
   Object[][] matriz = new Object[tam][tam];  
   Map<String, Integer> etiquetaAIndice = new HashMap<>();  
    
   for (int i = 0; i < tam; i++) {  
   etiquetaAIndice.put(nodos.get(i).etiqueta, i);  
   for (int j = 0; j < tam; j++) {  
   matriz[i][j] = 0;  
   }  
   }  
    
   for (Nodo nodo : nodos) {  
   int desdeIndice = etiquetaAIndice.get(nodo.etiqueta);  
   if (nodo.izquierda != null) {  
   int hastaIndice = etiquetaAIndice.get(nodo.izquierda.etiqueta);  
   matriz[desdeIndice][hastaIndice] = 1;  
   }  
   if (nodo.derecha!= null) {  
   int hastaIndice = etiquetaAIndice.get(nodo.derecha.etiqueta);  
   matriz[desdeIndice][hastaIndice] = 1;  
   }  
   }  
    
   return matriz;  
   }  
  }

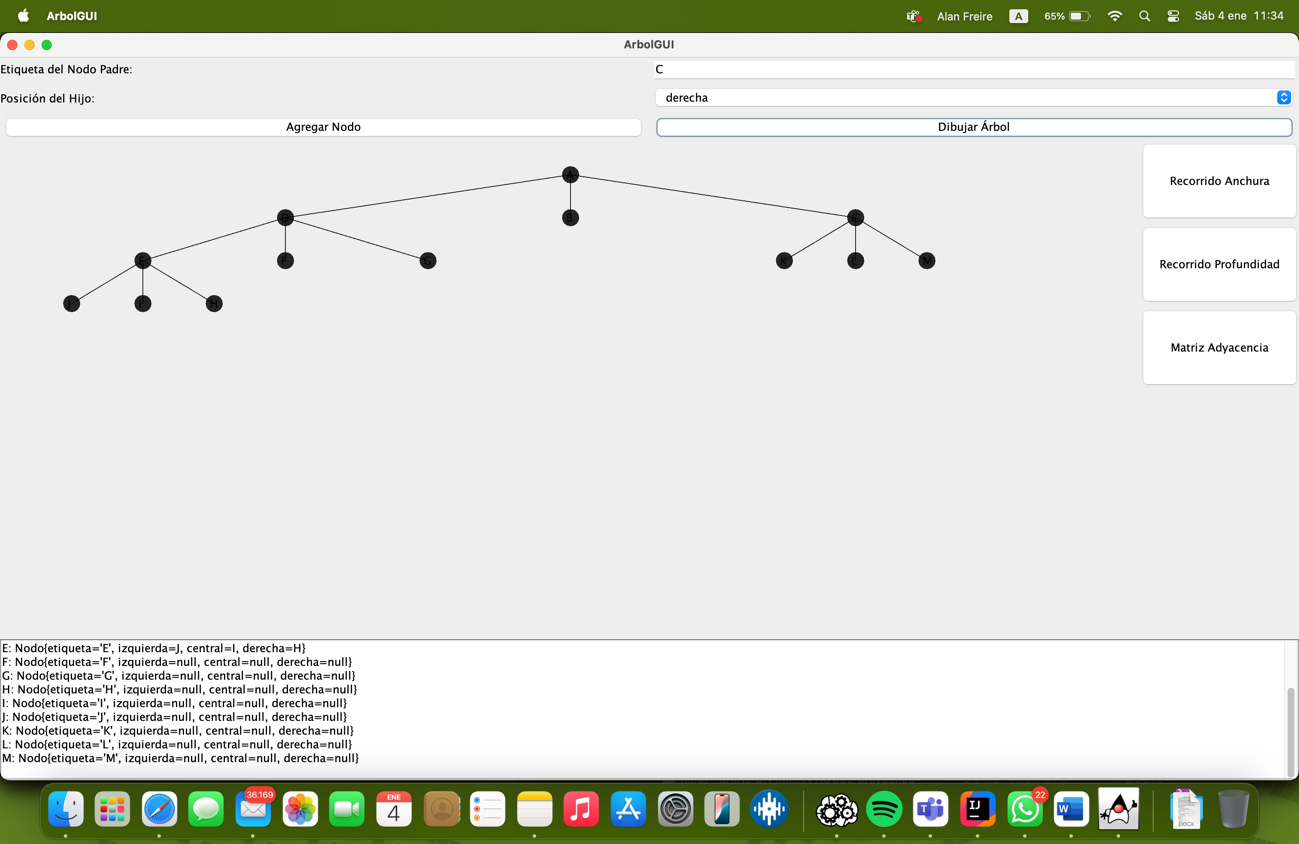
import java.awt.\*;  
  
class ArbolGrafico {  
 private ArbolTrinario arbol;  
  
 public ArbolGrafico(ArbolTrinario arbol) {  
 this.arbol = arbol;  
 }  
  
 public void dibujarArbol(Graphics2D g2d, Nodo nodo, int x, int y, int dimensionX, int dimensionY) {  
 if (nodo != null) {  
 // Tamaño reducido de los nodos (10 píxeles de radio)  
 int radio = 10;  
 g2d.fillOval(x - radio, y - radio, radio \* 2, radio \* 2);  
 g2d.drawString(nodo.etiqueta, x - radio / 2, y + radio / 2);  
  
 if (nodo.izquierda != null) {  
 g2d.drawLine(x, y, x - dimensionX, y + dimensionY);  
 dibujarArbol(g2d, nodo.izquierda, x - dimensionX, y + dimensionY, dimensionX / 2, dimensionY);  
 }  
  
 if (nodo.central != null) {  
 g2d.drawLine(x, y, x, y + dimensionY);  
 dibujarArbol(g2d, nodo.central, x, y + dimensionY, dimensionX / 2, dimensionY);  
 }  
  
 if (nodo.derecha != null) {  
 g2d.drawLine(x, y, x + dimensionX, y + dimensionY);  
 dibujarArbol(g2d, nodo.derecha, x + dimensionX, y + dimensionY, dimensionX / 4, dimensionY);  
 }  
 }  
 }  
}

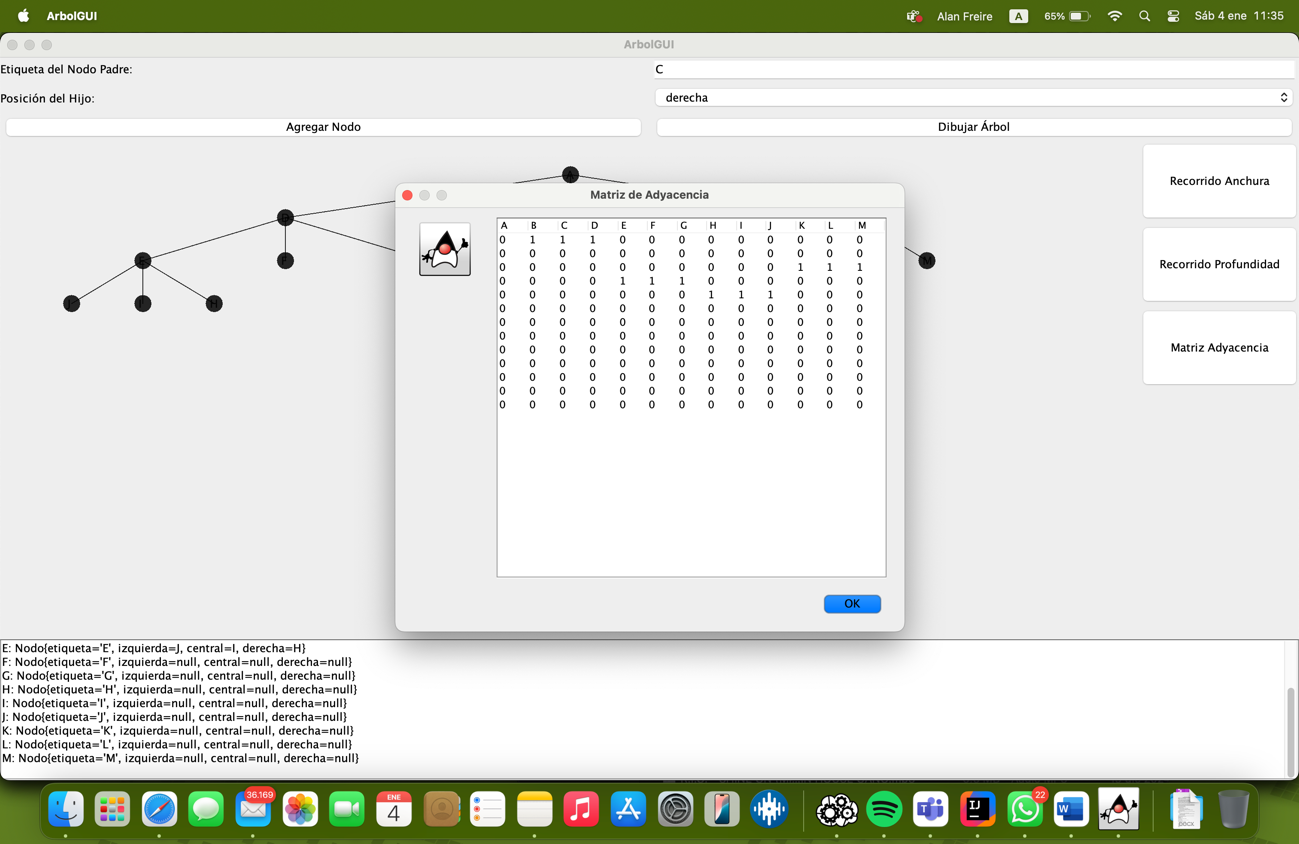
import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.Queue;  
import java.util.Stack;  
  
class ArbolTrinario {  
 private Nodo raiz;  
 private ArrayList<Nodo> nodos;  
 private int numNodos;  
 private static final int *DISTANCIA\_VERTICAL* = 50;  
 private static final int *ESPACIO\_INICIAL* = 300;  
  
 public ArbolTrinario() {  
 raiz = null;  
 nodos = new ArrayList<>();  
 numNodos = 0;  
 }  
  
 public void anadirNodo(Nodo nodo, Nodo padre, String posicion) {  
 if (padre == null) {  
 if (raiz == null) {  
 raiz = nodo; // El nodo es la raíz  
 nodos.add(nodo);  
 return;  
 } else {  
 throw new IllegalArgumentException("La raíz ya existe");  
 }  
 }  
  
 if (posicion == null || (!posicion.equalsIgnoreCase("izquierda") && !posicion.equalsIgnoreCase("central") && !posicion.equalsIgnoreCase("derecha"))) {  
 throw new IllegalArgumentException("Posición inválida. Use 'izquierda', 'central' o 'derecha'");  
 }  
  
 switch (posicion.toLowerCase()) {  
 case "izquierda":  
 if (padre.izquierda == null) {  
 padre.izquierda = nodo;  
 } else {  
 throw new IllegalArgumentException("Hijo izquierdo ya existe");  
 }  
 break;  
 case "central":  
 if (padre.central == null) {  
 padre.central = nodo;  
 } else {  
 throw new IllegalArgumentException("Hijo central ya existe");  
 }  
 break;  
 case "derecha":  
 if (padre.derecha == null) {  
 padre.derecha = nodo;  
 } else {  
 throw new IllegalArgumentException("Hijo derecho ya existe");  
 }  
 break;  
 }  
 nodos.add(nodo);  
 }  
  
 public void ajustarPosiciones() {  
 int anchoTotal = calcularAnchoTotal(raiz);  
 recalcularPosiciones(raiz, 0, 0, anchoTotal / 2);  
 }  
  
 private int calcularAnchoTotal(Nodo nodo) {  
 if (nodo == null) return 0;  
 int anchoIzquierdo = calcularAnchoTotal(nodo.izquierda);  
 int anchoCentral = calcularAnchoTotal(nodo.central);  
 int anchoDerecho = calcularAnchoTotal(nodo.derecha);  
 return Math.*max*(anchoIzquierdo + anchoCentral + anchoDerecho, *ESPACIO\_INICIAL*);  
 }  
  
 private void recalcularPosiciones(Nodo nodo, int x, int y, int espacio) {  
 if (nodo == null) return;  
  
 nodo.x = x;  
 nodo.y = y;  
  
 int nuevoEspacio = espacio / 2;  
 if (nodo.izquierda != null) {  
 recalcularPosiciones(nodo.izquierda, x - espacio, y + *DISTANCIA\_VERTICAL*, nuevoEspacio);  
 }  
  
 if (nodo.central != null) {  
 recalcularPosiciones(nodo.central, x, y + *DISTANCIA\_VERTICAL*, nuevoEspacio);  
 }  
  
 if (nodo.derecha != null) {  
 recalcularPosiciones(nodo.derecha, x + espacio, y + *DISTANCIA\_VERTICAL*, nuevoEspacio);  
 }  
 }  
  
 public Nodo getRaiz() {  
 return raiz;  
 }  
  
 public ArrayList<Nodo> getNodos() {  
 return nodos;  
 }  
  
 public String getEtiquetaNodoSiguiente() {  
 return String.*valueOf*((char) ('A' + numNodos++));  
 }  
  
 // Recorrido en Anchura (BFS)  
 public String bfs() {  
 if (raiz == null) return "";  
  
 StringBuilder resultado = new StringBuilder();  
 Queue<Nodo> queue = new LinkedList<>();  
 queue.add(raiz);  
  
 while (!queue.isEmpty()) {  
 Nodo nodo = queue.poll();  
 resultado.append(nodo.etiqueta).append(" ");  
 if (nodo.izquierda != null) queue.add(nodo.izquierda);  
 if (nodo.central != null) queue.add(nodo.central);  
 if (nodo.derecha != null) queue.add(nodo.derecha);  
 }  
  
 return resultado.toString().trim();  
 }  
  
 // Recorrido en Profundidad (DFS)  
 public String dfs() {  
 if (raiz == null) return "";  
  
 StringBuilder resultado = new StringBuilder();  
 Stack<Nodo> stack = new Stack<>();  
 stack.push(raiz);  
  
 while (!stack.isEmpty()) {  
 Nodo nodo = stack.pop();  
 resultado.append(nodo.etiqueta).append(" ");  
 if (nodo.derecha != null) stack.push(nodo.derecha);  
 if (nodo.central != null) stack.push(nodo.central);  
 if (nodo.izquierda != null) stack.push(nodo.izquierda);  
 }  
  
 return resultado.toString().trim();  
 }  
  
 public Object[][] getMatrizAdyacencia() {  
 int tam = nodos.size();  
 Object[][] matriz = new Object[tam][tam];  
  
 for (int i = 0; i < tam; i++) {  
 for (int j = 0; j < tam; j++) {  
 matriz[i][j] = 0;  
 }  
 }  
  
 for (Nodo nodo : nodos) {  
 int desdeIndice = nodos.indexOf(nodo);  
 if (nodo.izquierda != null) matriz[desdeIndice][nodos.indexOf(nodo.izquierda)] = 1;  
 if (nodo.central != null) matriz[desdeIndice][nodos.indexOf(nodo.central)] = 1;  
 if (nodo.derecha != null) matriz[desdeIndice][nodos.indexOf(nodo.derecha)] = 1;  
 }  
  
 return matriz;  
 }  
}

class Nodo {  
 int x, y;  
 String etiqueta;  
 Nodo izquierda, central, derecha;  
  
 public Nodo(int x, int y, String etiqueta) {  
 this.x = x;  
 this.y = y;  
 this.etiqueta = etiqueta;  
 this.izquierda = null;  
 this.central = null;  
 this.derecha = null;  
 }  
  
 @Override  
 public String toString() {  
 return "Nodo{" + "etiqueta='" + etiqueta + '\'' + ", izquierda=" + (izquierda != null ? izquierda.etiqueta : "null") + ", central=" + (central != null ? central.etiqueta : "null") + ", derecha=" + (derecha != null ? derecha.etiqueta : "null") + '}';  
 }  
}

import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.\*;  
  
public class ArbolGUI {  
 private JTable tbMatrizAdyacencia;  
 private JTextArea textArea;  
 private JTextField txtRaiz;  
 private JButton btnAgregarNodo;  
 private JButton btnDibujarArbol;  
 private JButton btnRecorridoAnchura;  
 private JButton btnRecorridoProfundidad;  
 private JLabel lblNodo;  
 private JLabel lblRaiz;  
 private JLabel lblHoja;  
 private JComboBox<String> cbIzqDer;  
 private JPanel panelArbol;  
 private JPanel panelGeneral;  
 private JPanel panelDatos;  
 private JLabel lblRecorridos;  
 private JButton btnMatrizAdyacencia;  
  
 // Cambiar Arbol por ArbolTrinario  
 private ArbolTrinario arbol = new ArbolTrinario();  
 private ArbolGrafico arbolGrafico = new ArbolGrafico(arbol);  
 private DefaultTableModel modeloTabla = new DefaultTableModel();  
  
 public ArbolGUI() {  
 // Inicialización de componentes  
 panelGeneral = new JPanel(new BorderLayout());  
 panelDatos = new JPanel(new GridLayout(3, 2, 5, 5));  
 panelArbol = new JPanel();  
 tbMatrizAdyacencia = new JTable();  
 textArea = new JTextArea(10, 30);  
 txtRaiz = new JTextField(15);  
 cbIzqDer = new JComboBox<>(new String[]{"izquierda", "central", "derecha"});  
 btnAgregarNodo = new JButton("Agregar Nodo");  
 btnDibujarArbol = new JButton("Dibujar Árbol");  
 btnRecorridoAnchura = new JButton("Recorrido Anchura");  
 btnRecorridoProfundidad = new JButton("Recorrido Profundidad");  
 btnMatrizAdyacencia = new JButton("Matriz Adyacencia");  
 lblNodo = new JLabel("Etiqueta del Nodo Padre:");  
 lblRecorridos = new JLabel("Recorridos:");  
  
 // Configuración del área de texto  
 textArea.setEditable(false);  
 JScrollPane scrollPane = new JScrollPane(textArea);  
  
 // Panel de datos  
 panelDatos.add(lblNodo);  
 panelDatos.add(txtRaiz);  
 panelDatos.add(new JLabel("Posición del Hijo:"));  
 panelDatos.add(cbIzqDer);  
 panelDatos.add(btnAgregarNodo);  
 panelDatos.add(btnDibujarArbol);  
  
 // Agregar componentes al panel general  
 panelGeneral.add(panelDatos, BorderLayout.*NORTH*);  
 panelGeneral.add(panelArbol, BorderLayout.*CENTER*);  
 panelGeneral.add(scrollPane, BorderLayout.*SOUTH*);  
  
 // Barra lateral para botones  
 JPanel panelBotones = new JPanel(new GridLayout(6, 1, 5, 5));  
 panelBotones.add(btnRecorridoAnchura);  
 panelBotones.add(btnRecorridoProfundidad);  
 panelBotones.add(btnMatrizAdyacencia);  
 panelGeneral.add(panelBotones, BorderLayout.*EAST*);  
  
 // Acción de los botones  
 btnAgregarNodo.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 try {  
 String etiqueta = arbol.getEtiquetaNodoSiguiente();  
 Nodo nuevoNodo = new Nodo(0, 0, etiqueta);  
 String etiquetaPadre = txtRaiz.getText().trim();  
 String posicion = cbIzqDer.getSelectedItem().toString(); // Obtener la posición como String  
  
 Nodo nodoPadre = null;  
 for (Nodo nodo : arbol.getNodos()) {  
 if (nodo.etiqueta.equals(etiquetaPadre)) {  
 nodoPadre = nodo;  
 break;  
 }  
 }  
 arbol.anadirNodo(nuevoNodo, nodoPadre, posicion);  
 imprimirArbol();  
 dibujarArbolEnPanel();  
  
 } catch (Exception ex) {  
 JOptionPane.*showMessageDialog*(null, "Error al agregar nodo: " + ex.getMessage());  
 }  
 }  
 });  
 btnDibujarArbol.addActionListener(e -> {  
 try {  
 arbol.ajustarPosiciones(); // Ajusta las posiciones de los nodos  
 dibujarArbolEnPanel(); // Dibuja el árbol con las posiciones actualizadas  
 } catch (Exception ex) {  
 JOptionPane.*showMessageDialog*(null, "Error al dibujar el árbol: " + ex.getMessage());  
 }  
 });  
 btnRecorridoAnchura.addActionListener(e -> textArea.append("Recorrido en Anchura (BFS): " + arbol.bfs() + "\n"));  
 btnRecorridoProfundidad.addActionListener(e -> textArea.append("Recorrido en Profundidad (DFS): " + arbol.dfs() + "\n"));  
 btnMatrizAdyacencia.addActionListener(e -> mostrarMatrizAdyacencia());  
 btnDibujarArbol.addActionListener(e -> dibujarArbolEnPanel());  
 }  
  
 private void dibujarArbolEnPanel() {  
 Graphics g = panelArbol.getGraphics();  
 if (g != null) {  
 Graphics2D g2d = (Graphics2D) g;  
 g2d.clearRect(0, 0, panelArbol.getWidth(), panelArbol.getHeight());  
 int x = panelArbol.getWidth() / 2;  
 int y = 40;  
 arbolGrafico.dibujarArbol(g2d, arbol.getRaiz(), x, y, panelArbol.getWidth() / 4, 50);  
 }  
 }  
 public void dibujarArbol(Graphics2D g2d, Nodo nodo, int x, int y, int dimensionX, int dimensionY) {  
 if (nodo != null) {  
 // Ajustar el tamaño del nodo (bolas más pequeñas)  
 int radio = 10; // Tamaño del radio del nodo  
 g2d.fillOval(x - radio, y - radio, radio \* 2, radio \* 2);  
 g2d.drawString(nodo.etiqueta, x - radio / 2, y + radio / 2);  
  
 if (nodo.izquierda != null) {  
 g2d.drawLine(x, y, x - dimensionX, y + dimensionY);  
 dibujarArbol(g2d, nodo.izquierda, x - dimensionX, y + dimensionY, dimensionX / 2, dimensionY);  
 }  
  
 if (nodo.central != null) {  
 g2d.drawLine(x, y, x, y + dimensionY);  
 dibujarArbol(g2d, nodo.central, x, y + dimensionY, dimensionX / 2, dimensionY);  
 }  
  
 if (nodo.derecha != null) {  
 g2d.drawLine(x, y, x + dimensionX, y + dimensionY);  
 dibujarArbol(g2d, nodo.derecha, x + dimensionX, y + dimensionY, dimensionX / 4, dimensionY);  
 }  
 }  
 }  
  
  
 private void imprimirArbol() {  
 textArea.setText("Nodos:\n");  
 for (Nodo nodo : arbol.getNodos()) {  
 textArea.append(nodo.etiqueta + ": " + nodo.toString() + "\n");  
 }  
 }  
  
 private void mostrarMatrizAdyacencia() {  
 Object[][] matriz = arbol.getMatrizAdyacencia();  
 String[] nombreColumnas = new String[matriz.length];  
 for (int i = 0; i < matriz.length; i++) {  
 nombreColumnas[i] = String.*valueOf*((char) ('A' + i));  
 }  
 modeloTabla.setDataVector(matriz, nombreColumnas);  
 JOptionPane.*showMessageDialog*(null, new JScrollPane(new JTable(modeloTabla)), "Matriz de Adyacencia", JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
  
 public static void main(String[] args) {  
 JFrame frame = new JFrame("ArbolGUI");  
 frame.setContentPane(new ArbolGUI().panelGeneral);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setSize(1000, 600);  
 frame.setVisible(true);  
 }  
}







Link GitHub:

<https://github.com/apfreire1997/arboltrinario.git>