**UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS**

Universidad del Perú, Decana de América

**Facultad de Ingeniería de Sistemas e Informática**

Escuela Profesional de Ingeniería de Sistemas

****

**SOFTWARE DE ÁRBOL BINARIO DE BÚSQUEDA**

**ASIGNATURA:**  Inteligencia Artificial

**DOCENTE:** Dr. Hugo Vera Huerta

**ALUMNOS:**

* GONZALES ROJAS, Manuel Hernan
* CORDOVA SILVA, Giuseppe Jefferson
* DE LA CRUZ GUILLEN, Ivan Paolo
* PEREZ GRADOS, Jose Luis
* RAFAEL JAVIER, Hector Imanol
* ROJAS HURTADO, Karen Antonia
* TORRES ESPINOZA, Alejandro Paul
* VILCHEZ GIRALDO, Jamie Edinso

Lima - Perú

**2023**

# ÍNDICE

[**ÍNDICE 2**](#_memdc4iavo5z)

[**MARCO TEÓRICO 4**](#_8beefcj0zz27)

[Definición de árbol 4](#_w40fke3bb1q6)

[Árbol binario de búsqueda 4](#_5wt5fwkap11l)

[Búsqueda de un elemento en un Árbol Binario de Búsqueda 4](#_pjsyh8f4qiat)

[Búsqueda por profundidad 4](#_amfsok2qr0kn)

[Búsqueda por anchura 4](#_3yvtguunv5bw)

[**IMPLEMENTACIÓN DEL SOFTWARE 5**](#_rg3s2i1hnma1)

[CÓDIGO 5](#_2vnm9erx0jtz)

[Clase principal 5](#_q25pgdrk5otd)

[Método principal 5](#_waoe324ii2l2)

[Método crearSensores 5](#_mpx1mq5dczm)

[Método agregarSensoresArbol 6](#_abmx6y3i4bmt)

[Método listarSensores 6](#_h0jxqfw5833y)

[Método mostrarSensoresArbol 6](#_1kj5fcfbp5tl)

[Método btnGenerarArbol 8](#_kf0w2gm1v42)

[Método btnListarSensores 8](#_n0is9ovsu6pr)

[Método btnLimpiarArbol 8](#_lz66wg8xof7k)

[Método btnBusquedaProfundidad 9](#_wat9jstwmev2)

[Método btnBusquedaAnchura 10](#_gt5u6tcag7gq)

[Clase ABBSensores 11](#_40oy8r91yof3)

[Método agregarSensor 11](#_p5aop7c01bw0)

[Método busquedaProfundidad 12](#_q0ainq4ogni4)

[Método busquedaAnchura 13](#_yfqll8l83wxd)

[Clase Arbol 14](#_rysgu2sd4a0q)

[Método dibujarArbol 15](#_xqwi2f9pkgbn)

[Clase Linea 16](#_9m36cuzblerg)

[Método dibujarLinea 16](#_ygp0affd95g0)

[Clase NodoABB 17](#_aajr6ncbnp00)

[Clase Rectangulo 17](#_o24cnlyn62xb)

[Método dibujarRectangulo 17](#_uic269g2yefo)

[INTERFAZ 19](#_rqolf8bepuip)

[Interfaz principal 19](#_63tj44mjbhes)

[Botón Listar sensores 20](#_d7py9ea050i)

[Botón Generar Árbol Binario de Búsqueda 21](#_v2m0wme1eliw)

[Botón Limpiar árbol 22](#_5a8z64od1phd)

[Botón Búsqueda por profundidad 23](#_oy0uqs3090tj)

[Botón Búsqueda por anchura 24](#_krf61ig9ffpd)

# MARCO TEÓRICO

## Definición de árbol

Los árboles se utilizan comúnmente en informática para organizar y almacenar datos de manera eficiente y permiten una rápida búsqueda, inserción y eliminación de elementos. Los nodos en un árbol están conectados por bordes o enlaces.

## Árbol binario de búsqueda

Es un tipo específico de árbol en el que cada nodo tiene como máximo dos hijos, uno a la izquierda y otro a la derecha.La característica principal de este árbol es que para cada nodo en el árbol, todos los nodos en su subárbol izquierdo tienen un valor menor que el nodo en sí, y todos los nodos en su subárbol derecho tienen un valor mayor.

## Búsqueda de un elemento en un Árbol Binario de Búsqueda

### Búsqueda por profundidad

-La búsqueda por profundidad es un algoritmo de recorrido que se caracteriza por explorar una rama completa del grafo antes de retroceder y explorar otras ramas

### Búsqueda por anchura

-La búsqueda por anchura se caracteriza por explorar todos los nodos vecinos de un nodo antes de avanzar al siguiente nivel.

# IMPLEMENTACIÓN DEL SOFTWARE

## CÓDIGO

### Clase principal

public class frmABBSensores extends javax.swing.JFrame {

String[] sensores;

ABBSensores abbs;

public frmABBSensores() {

initComponents();

sensores = new String[15];

abbs = new ABBSensores();

crearSensores();

agregarSensoresArbol();

}

}

* La clase frmABBSensores es la clase principal y es donde se implementan todos los códigos de los principales procesos.
* En el momento de la ejecución del programa se crea el arreglo de cadenas para los sensores y se agregan de manera inmediata al árbol binario de búsqueda.

#### Método principal

public static void main(String args[]) {

java.awt.EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

frmABBSensores frmABBS = new frmABBSensores();

frmABBS.setLocationRelativeTo(null);

frmABBS.setVisible(true);

}

});

}

* Crea la clase principal y se ejecuta la visualización de la interfaz.

#### Método crearSensores

public void crearSensores(){

sensores[0] = "MOV";

sensores[1] = "TEM";

sensores[2] = "LUZ";

sensores[3] = "PRE";

sensores[4] = "PRO";

sensores[5] = "HUME";

sensores[6] = "GAS";

sensores[7] = "SON";

sensores[8] = "VEL";

sensores[9] = "FUE";

sensores[10] = "ACE";

sensores[11] = "HUMO";

sensores[12] = "PH";

sensores[13] = "CO2";

sensores[14] = "OXI";

}

* Inserta los nombres de los sensores en el arreglo de cadenas.

#### Método agregarSensoresArbol

public void agregarSensoresArbol(){

for(int i=0; i<15; i++){

abbs.agregarSensor(sensores[i]);

}

}

* Agrega el arreglo de sensores al árbol binario de búsqueda.

#### Método listarSensores

public String listarSensores(String[] sensores){

String listar = "";

for(int i=0; i<15; i++){

listar = listar + sensores[i] + "\n";

}

return listar;

}

* Retorna una cadena llamada “listar” en el que se encuentran los nombres de los sensores.

#### Método mostrarSensoresArbol

public void mostrarSensoresArbol(){

lblMov.setText("MOV");

lblLuz.setText("LUZ");

lblTem.setText("TEM");

lblHume.setText("HUME");

lblPre.setText("PRE");

lblVel.setText("VEL");

lblGas.setText("GAS");

lblHumo.setText("HUMO");

lblPh.setText("PH");

lblPro.setText("PRO");

lblFue.setText("FUE");

lblOxi.setText("OXI");

lblSon.setText("SON");

lblAce.setText("ACE");

lblCO2.setText("CO2");

lblMov.setBounds(350, 41, 39, 28);

lblLuz.setBounds(250, 121, 39, 28);

lblTem.setBounds(450, 121, 39, 28);

lblHume.setBounds(195, 201, 44, 28);

lblPre.setBounds(400, 201, 39, 28);

lblVel.setBounds(500, 201, 39, 28);

lblGas.setBounds(150, 281, 39, 28);

lblHumo.setBounds(245, 281, 44, 28);

lblPh.setBounds(355, 281, 34, 28);

lblPro.setBounds(450, 281, 39, 28);

lblFue.setBounds(100, 361, 39, 28);

lblOxi.setBounds(300, 361, 39, 28);

lblSon.setBounds(500, 361, 39, 28);

lblAce.setBounds(50, 441, 39, 28);

lblCO2.setBounds(100, 521, 39, 28);

}

* Muestra los nombres de los sensores en la interfaz principal del árbol binario de búsqueda.

#### Método btnGenerarArbol

private void btnGenerarArbolActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

jPanelArbol.setLayout(null);

Arbol a = new Arbol();

Graphics g = jPanelArbol.getGraphics();

a.dibujarArbol(g);

mostrarSensoresArbol();

}

* Crea una clase de tipo Arbol y utilizamos el método dibujarArbol de este para poder graficar el árbol binario de búsqueda en la interfaz principal.

#### Método btnListarSensores

private void btnListarSensoresActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

txtListaSensores.setText(listarSensores(sensores));

}

* Muestra el arreglo de sensores utilizado en un txtArea de la interfaz principal.

#### Método btnLimpiarArbol

private void btnLimpiarArbolActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

jPanelArbol.repaint();

lblMov.setText("");

lblLuz.setText("");

lblTem.setText("");

lblHume.setText("");

lblPre.setText("");

lblVel.setText("");

lblGas.setText("");

lblHumo.setText("");

lblPh.setText("");

lblPro.setText("");

lblFue.setText("");

lblOxi.setText("");

lblSon.setText("");

lblAce.setText("");

lblCO2.setText("");

}

* Elimina el árbol binario de búsqueda de la interfaz principal.

#### Método btnBusquedaProfundidad

private void btnBusquedaProfundidadActionPerformed(java.awt.event.ActionEvent evt) {

String busquedaSensor = txtBusquedaSensor.getText();

if(lblCO2.getText().equals("")){

txtResultado.setText("ERROR. GENERE EL ÁRBOL DE SENSORES.");

}else{

if(txtListaSensores.getText().equals("")){

txtResultado.setText("ERROR. GENERE LA LISTA DE SENSORES.");

}else{

if(txtBusquedaSensor.getText().equals("")){

txtResultado.setText("ERROR. INGRESE UN SENSOR POR BUSCAR.");

}else{

List<String> comparaciones = abbs.busquedaProfundidad(busquedaSensor);

if(comparaciones.contains(busquedaSensor)) {

txtResultado.setText("Sensor encontrado tras " + comparaciones.size() + " comparaciones. Nodos: " + comparaciones);

} else {

txtResultado.setText("Sensor no encontrado tras " + comparaciones.size() + " comparaciones. Nodos: " + comparaciones);

}

}

}

}

}

* Primero, obtenemos el valor del sensor por buscar y lo llamaremos busquedaSensor.
* Luego, comprobamos que en la interfaz principal estén correctamente generados el árbol binario de búsqueda y la lista de sensores, sino nos mostrará un mensaje de error.
* Una vez verificado lo anterior, nos aseguramos que busquedaSensor no esté vacío, sino nos aparecerá un mensaje de error.
* Si busquedaSensor no está vacío, entonces el software realizará la búsqueda por profundidad del sensor en el árbol binario de búsqueda.
* Por último, nos mostrará un mensaje si es que se ha encontrado o si es que no se ha encontrado, además del recorrido que se ha realizado en la búsqueda.

#### Método btnBusquedaAnchura

private void btnBusquedaAnchuraActionPerformed(java.awt.event.ActionEvent evt) {

String busquedaSensor = txtBusquedaSensor.getText();

if(lblCO2.getText().equals("")){

txtResultado.setText("ERROR. GENERE EL ÁRBOL DE SENSORES.");

}else{

if(txtListaSensores.getText().equals("")){

txtResultado.setText("ERROR. GENERE LA LISTA DE SENSORES.");

}else{

if(txtBusquedaSensor.getText().equals("")){

txtResultado.setText("ERROR. INGRESE UN SENSOR POR BUSCAR.");

}else{

List<String> comparaciones = abbs.busquedaAnchura(busquedaSensor);

if(comparaciones.contains(busquedaSensor)) {

txtResultado.setText("Sensor encontrado tras " + comparaciones.size() + " comparaciones. Nodos: " + comparaciones);

} else {

txtResultado.setText("Sensor no encontrado tras " + comparaciones.size() + " comparaciones. Nodos: " + comparaciones);

}

}

}

}

}

* Primero, obtenemos el valor del sensor por buscar y lo llamaremos busquedaSensor.
* Luego, comprobamos que en la interfaz principal estén correctamente generados el árbol binario de búsqueda y la lista de sensores, sino nos mostrará un mensaje de error.
* Una vez verificado lo anterior, nos aseguramos que busquedaSensor no esté vacío, sino nos aparecerá un mensaje de error.
* Si busquedaSensor no está vacío, entonces el software realizará la búsqueda por anchura del sensor en el árbol binario de búsqueda.
* Por último, nos mostrará un mensaje si es que se ha encontrado o si es que no se ha encontrado, además del recorrido que se ha realizado en la búsqueda.

### Clase ABBSensores

public class ABBSensores {

NodoABB raiz;

public ABBSensores(){

this.raiz = null;

}

}

* Clase principal del árbol binario de búsqueda.
* Contiene los métodos para agregar un elemento al árbol binario de búsqueda y realizar la búsqueda de un elemento por profundidad o por anchura.

#### Método agregarSensor

public void agregarSensor(String sensor){

NodoABB nuevo = new NodoABB();

nuevo.setSensor(sensor);

if(raiz == null){

raiz = nuevo;

}else{

NodoABB anterior = null, reco;

reco = raiz;

while(reco != null){

anterior = reco;

if(sensor.compareToIgnoreCase(reco.getSensor()) < 0){

reco = reco.izquierdo;

}else{

reco = reco.derecho;

}

}

if(sensor.compareToIgnoreCase(anterior.getSensor()) < 0){

anterior.izquierdo = nuevo;

}else{

anterior.derecho = nuevo;

}

}

}

* Agrega un elemento al árbol binario de búsqueda, este elemento es recibido como parámetro.
* Si es el primer elemento ingresado al árbol, este será considerado como raíz del árbol.
* Si no es el primer elemento, se realizará una comparación con los elementos de los árboles para saber la posición que tomará.
* Los elementos del árbol binario de búsqueda están ordenados de manera alfabética.

#### Método busquedaProfundidad

// Búsqueda por Profundidad (DFS)

public List<String> busquedaProfundidad(String sensor) {

List<String> comparaciones = new ArrayList<>();

return dfs(this.raiz, sensor, comparaciones);

}

private List<String> dfs(NodoABB nodo, String sensor, List<String> comparaciones) {

if (nodo == null) {

return comparaciones;

}

comparaciones.add(nodo.getSensor());

if (nodo.getSensor().equals(sensor)) {

return comparaciones;

}

comparaciones = dfs(nodo.izquierdo, sensor, comparaciones);

if (comparaciones.get(comparaciones.size() - 1).equals(sensor)) {

return comparaciones;

}

return dfs(nodo.derecho, sensor, comparaciones);

}

* Función principal busquedaProfundidad:

Esta función simplemente inicia el proceso de búsqueda DFS desde la raíz del árbol binario y retorna una lista de comparaciones.

* Función auxiliar dfs:
  + Si el nodo actual es null, se retorna la lista de comparaciones hasta el momento.
  + Se añade el valor del nodo actual (nodo.getSensor()) a la lista de comparaciones.
  + Si el valor del nodo actual es igual al valor buscado (sensor), se retorna la lista de comparaciones.
  + Se realiza una llamada recursiva al hijo izquierdo.
  + Si el último valor en la lista de comparaciones es el valor buscado, se retorna la lista de comparaciones. Esto es para evitar que la búsqueda continúe innecesariamente después de encontrar el valor.
  + Finalmente, se realiza una llamada recursiva al hijo derecho.
* Lógica:

DFS explora tan profundamente como sea posible a lo largo de una rama antes de retroceder. Así que para un ABB, primero explora todo el subárbol izquierdo de un nodo y luego explora el subárbol derecho.

#### Método busquedaAnchura

// Búsqueda por Anchura (BFS)

public List<String> busquedaAnchura(String sensor) {

List<String> comparaciones = new ArrayList<>();

Queue<NodoABB> cola = new LinkedList<>();

cola.add(this.raiz);

while (!cola.isEmpty()) {

NodoABB nodoActual = cola.poll();

if (nodoActual == null) {

continue;

}

comparaciones.add(nodoActual.getSensor());

if (nodoActual.getSensor().equals(sensor)) {

return comparaciones;

}

cola.add(nodoActual.izquierdo);

cola.add(nodoActual.derecho);

}

return comparaciones;

}

* Función busquedaAnchura:
  + Se inicializa una cola y se añade la raíz del árbol a la cola.
  + Mientras la cola no esté vacía:
    - Se extrae (dequeues) el nodo del frente de la cola.
    - Si el nodo extraído es null, se continúa al siguiente ciclo del bucle.
    - Se añade el valor del nodo actual (nodoActual.getSensor()) a la lista de comparaciones.
    - Si el valor del nodo actual es igual al valor buscado (sensor), se retorna la lista de comparaciones.
    - Se añade el hijo izquierdo y el hijo derecho del nodo actual a la cola.
* Lógica:

BFS explora todos los nodos a la actual profundidad antes de moverse a los nodos del siguiente nivel de profundidad. Así que para un ABB, primero explora un nodo y luego explora todos sus hijos antes de explorar los hijos de sus hijos.

### Clase Arbol

public class Arbol {

public Arbol(){

}

}

* Clase auxiliar que nos va a permitir graficar el árbol binario de búsqueda en la interfaz principal.

#### Método dibujarArbol

public void dibujarArbol(Graphics g){

Rectangulo[] r = new Rectangulo[15];

r[0] = new Rectangulo(339, 40, 50, 30); //MOV

r[1] = new Rectangulo(239, 120, 50, 30); //LUZ

r[2] = new Rectangulo(439, 120, 50, 30); //TEM

r[3] = new Rectangulo(189, 200, 50, 30); //HUME

r[4] = new Rectangulo(389, 200, 50, 30); //PRE

r[5] = new Rectangulo(489, 200, 50, 30); //VEL

r[6] = new Rectangulo(139, 280, 50, 30); //GAS

r[7] = new Rectangulo(239, 280, 50, 30); //HUMO

r[8] = new Rectangulo(339, 280, 50, 30); //PH

r[9] = new Rectangulo(439, 280, 50, 30); //PRO

r[10] = new Rectangulo(89, 360, 50, 30); //FUE

r[11] = new Rectangulo(289, 360, 50, 30); //OXI

r[12] = new Rectangulo(489, 360, 50, 30); //SON

r[13] = new Rectangulo(39, 440, 50, 30); //ACE

r[14] = new Rectangulo(89, 520, 50, 30); //CO2

for(int i=0; i<15; i++){

r[i].dibujarRectangulo(g);

}

Linea[] l = new Linea[14];

l[0] = new Linea(364, 70, 264, 120);

l[1] = new Linea(364, 70, 464, 120);

l[2] = new Linea(264, 150, 214, 200);

l[3] = new Linea(464, 150, 414, 200);

l[4] = new Linea(464, 150, 514, 200);

l[5] = new Linea(214, 230, 164, 280);

l[6] = new Linea(214, 230, 264, 280);

l[7] = new Linea(414, 230, 364, 280);

l[8] = new Linea(414, 230, 464, 280);

l[9] = new Linea(164, 310, 114, 360);

l[10] = new Linea(364, 310, 314, 360);

l[11] = new Linea(464, 310, 514, 360);

l[12] = new Linea(114, 390, 64, 440);

l[13] = new Linea(64, 470, 114, 520);

for(int i=0; i<14; i++){

l[i].dibujarLinea(g);

}

}

* Dibuja el árbol binario de búsqueda para el problema planteado y para los sensores utilizados.

### Clase Linea

public class Linea {

int x1;

int y1;

int x2;

int y2;

public Linea(int x1, int y1, int x2, int y2){

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

}

}

* Clase que nos va a ayudar a representar la descendencia de un nodo en el árbol binario de búsqueda.

#### Método dibujarLinea

public void dibujarLinea(Graphics g){

g.drawLine(x1, y1, x2, y2);

}

* Dibuja una línea en la interfaz principal, recibe como parámetros la posición inicial(x1, y1) y la posición final(x2, y2).

### Clase NodoABB

public class NodoABB {

NodoABB izquierdo, derecho;

String sensor;

public NodoABB(){

this.izquierdo = null;

this.derecho = null;

}

}

* Clase que representa a los nodos del árbol binario de búsqueda, aquí se incluye la creación de los nodos izquierdo y derecho, que representan a los hijos de un nodo.

### Clase Rectangulo

public class Rectangulo {

int x1;

int y1;

int x2;

int y2;

public Rectangulo(int x1, int y1, int x2, int y2){

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

}

}

* Clase que nos va a ayudar a representar los nodos del árbol binario de búsqueda.

#### Método dibujarRectangulo

public void dibujarRectangulo(Graphics g){

g.drawRect(x1, y1, x2, y2);

}

* Dibuja un rectángulo en la interfaz principal, recibe como parámetros la posición inicial(x1, y1), la anchura(x2) y la altura(y2).

## INTERFAZ

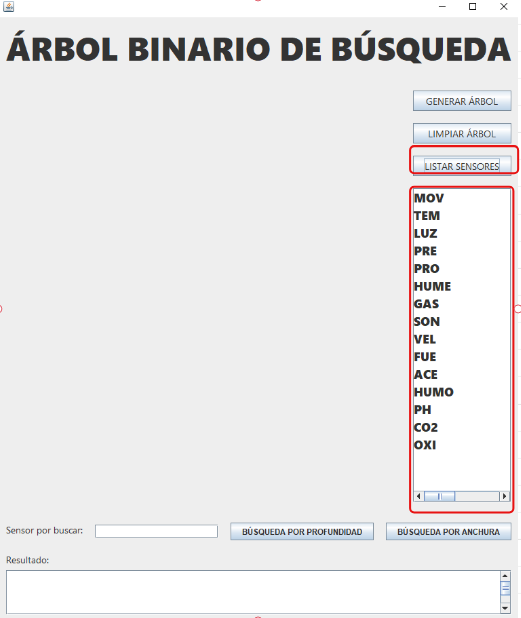
### Interfaz principal



En la interfaz principal observamos diferentes botones que nos van a ayudar a realizar los procesos de búsqueda, ya sea por profundidad o por anchura.

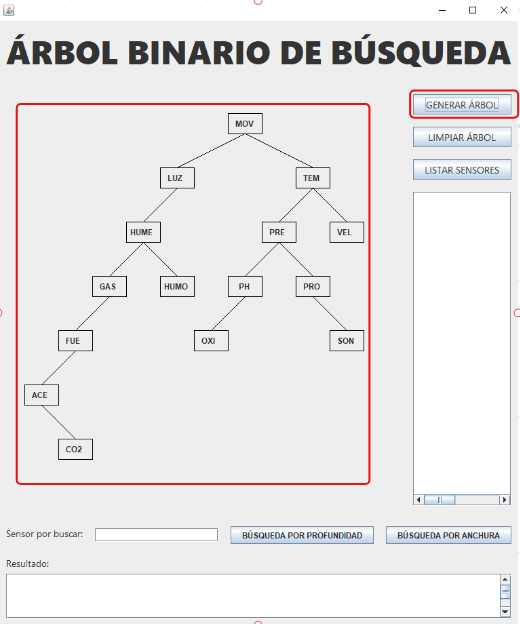
A continuación, se procederá a explicar la función de cada uno de estos botones.

#### Botón Listar sensores



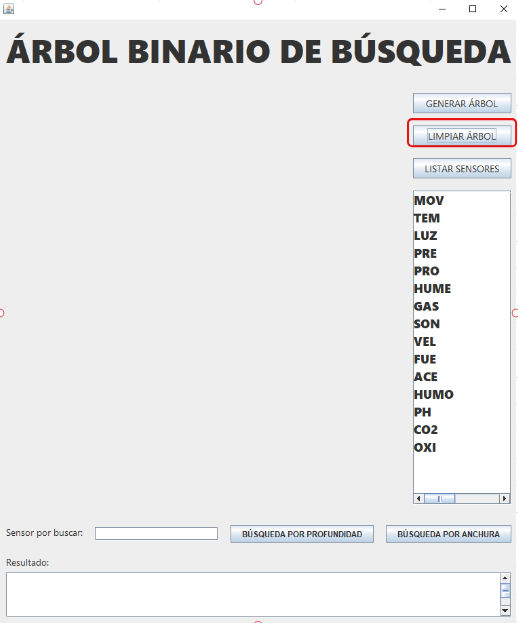
Muestra la lista de elementos utilizados para el árbol binario de búsqueda en la parte derecha de la interfaz principal-

#### Botón Generar Árbol Binario de Búsqueda



Genera el árbol binario de búsqueda para el problema planteado y los elementos utilizados.

#### Botón Limpiar árbol



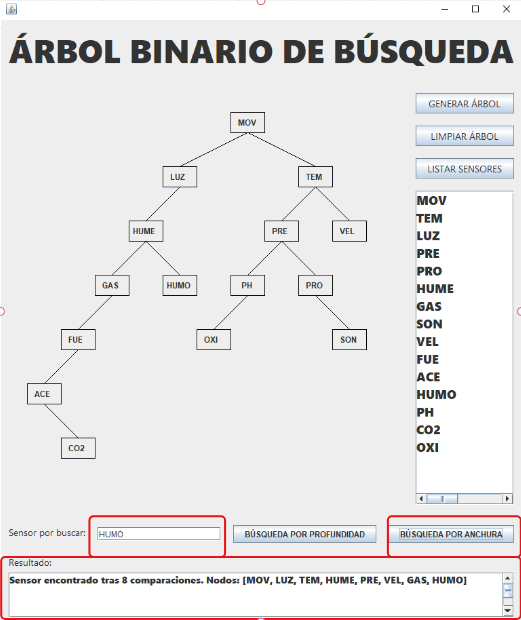
Se encarga de eliminar el árbol binario de búsqueda generado anteriormente.

#### Botón Búsqueda por profundidad



Para que este botón realice su acción, es necesario que haya un sensor por buscar, es decir, que el espacio esté rellenado correctamente, sino nos mostrará un mensaje de error. En el caso de que todo esté correcto, se realizará la búsqueda por profundidad y en la parte inferior nos mostrará las comparaciones realizadas y el recorrido que ha realizado.

#### Botón Búsqueda por anchura



Igual que en el caso anterior, es necesario cumplir ciertas condiciones para que este botón realice su acción correctamente. La diferencia con el anterior, es el método que utiliza para realizar la búsqueda, siendo este por anchura.