
# UNIVERSIDAD DE GUADALAJARA

## CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

## DIVISIÓN DE ELECTRÓNICA Y COMPUTACIÓN

## DEPARTAMENTO DE CIENCIAS COMPUTACIONALES.

## INGENIERÍA EN INFORMÁTICA.

## INTELIGENCIA ARTIFICIAL

## BUSQUEDA POR ANCHURA

## LUNES 15:00 – 19:00

**Ocampo Orozco Alan Vidal - 217451073**

## FECHA: 10/03/2024

## Prof. RAMIRO LUPERCIO CORONEL

OBSERVACIONES:

_____

_____

# Índice.

## Contenido

## Introducción:

La implementación de algoritmos de búsqueda en inteligencia artificial es fundamental para resolver problemas complejos y tomar decisiones en distintos contextos. En este proyecto, nos enfocamos en la simulación de movimiento a través de laberintos utilizando dos algoritmos de búsqueda: profundidad y anchura. El objetivo es comprender y aplicar estos algoritmos para encontrar el camino óptimo desde un punto de inicio hasta la salida en un laberinto predefinido.

## Desarrollo

En el desarrollo de este proyecto, se ha utilizado el lenguaje de programación Java para la implementación de la simulación. Se creó una interfaz gráfica que permite interactuar con la aplicación y visualizar el proceso de búsqueda en el laberinto. Para la búsqueda por profundidad, se empleó el método de recursión para explorar las distintas rutas posibles hasta alcanzar la salida.

Posteriormente, se extendió el proyecto para incluir la búsqueda por anchura. Este algoritmo utiliza una cola para explorar los nodos vecinos de manera gradual, priorizando la exploración en amplitud. Se ajustó la interfaz gráfica para reflejar el recorrido realizado por el algoritmo y visualizar la solución encontrada.

Además, se incorporó la capacidad de cambiar entre diferentes laberintos, lo que permite evaluar la eficacia de los algoritmos en distintos escenarios. Se realizaron ajustes para reiniciar el laberinto y la solución visualizada al cambiar de escenario.

## CODIGO

## MAIN

```
*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package org.busquedaprofunda;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
```

```
import java.util.Queue;
import javax.swing.JFrame;
import java.util.Random;


/**
 *
 * @author Medaf
 */

public class Main extends javax.swing.JFrame {


    private final List<Integer> camino = new ArrayList<>();
    private int[][] mapa;
    public int indiceLaberinto = 1; // Laberinto inicial

    /**
     * Creates new form Main
     */
    public Main() {
        initComponents();
        //mapa = xd.generarLaberinto();
        mapa = laberintosPred.obtenerLaberinto(indiceLaberinto);

    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jButton1 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jButton1.setText("ENCONTRAR");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jButton2.setText("Cambiar Laberinto");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
```

```
        }
    });

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(46, 46, 46)
            .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 108,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 49,
Short.MAX_VALUE)
            .addComponent(jButton2)
            .addGap(31, 31, 31))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
            .addContainerGap(500, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 49,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 49,
javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(12, 12, 12))
    );

    pack();
}// </editor-fold>

  private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// Utiliza el laberinto específico para la búsqueda
    mapa = laberintosPred.obtenerLaberinto(indiceLaberinto);

    boolean encontrado = BusquedaAnchura(mapa, 1, 1);

    if (encontrado) {
        repaint();
        System.out.println("¡Camino encontrado!");
    } else {
        System.out.println("No se encontró un camino.");
    }

    // Actualiza al siguiente laberinto
```

```
        //indiceLaberinto = (indiceLaberinto % laberintosPred.obtenerCantidadLaberintos()) + 1;
    }

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        indiceLaberinto++;
        reiniciarLaberinto();
        repaint(); // Para redibujar el nuevo laberinto
    }

    private void reiniciarLaberinto() {
    //mapa = xd.generarLaberinto(); // Utiliza tu método para generar un nuevo laberinto
    //camino.clear(); // Limpia el camino anterior
    //System.out.println("Laberinto reiniciado.");

    mapa = laberintosPred.obtenerLaberinto(indiceLaberinto); // Utiliza tu método para obtener el
laberinto actual
    camino.clear(); // Limpia el camino anterior

    System.out.println("Laberinto reiniciado.");

    if(indiceLaberinto==7){
        indiceLaberinto=1;
        mapa = laberintosPred.obtenerLaberinto(indiceLaberinto); // Utiliza tu método para obtener
el laberinto actual
        camino.clear(); // Limpia el camino anterior

    }
}
    @Override
public void paint(Graphics g) {
    super.paint(g);

    int cellSize = 20;

    // Ajustar posición del laberinto
    int offsetX = 30;
    int offsetY = 30;

    // Draw the maze
    for (int row = 0; row < mapa.length; row++) {
        for (int col = 0; col < mapa[0].length; col++) {
            Color color;
            switch (mapa[row][col]) {
                case 1:
                    color = Color.BLACK; // Pared
                    break;
```

```
                case 9:
                    color = Color.RED; // Salida
                    break;
                default:
                    color = Color.WHITE; // Camino
            }
            g.setColor(color);
            g.fillRect(offsetX + col * cellSize, offsetY + row * cellSize, cellSize, cellSize);

            // Dibujar bordes del camino en rojo
            if (camino.contains(col) && camino.contains(row)) {
                g.setColor(Color.RED);
                g.drawRect(offsetX + col * cellSize, offsetY + row * cellSize, cellSize, cellSize);
            }
        }
    }

    // Draw the path in green
    for (int i = 0; i < camino.size(); i += 2) {
        int caminoX = camino.get(i);
        int caminoY = camino.get(i + 1);

        // Ajustar las coordenadas del camino según el desplazamiento
        int adjustedX = offsetX + caminoX * cellSize;
        int adjustedY = offsetY + caminoY * cellSize;

        g.setColor(Color.green);
        g.fillRect(adjustedX, adjustedY, cellSize, cellSize);
    }
}


     public void actionPerformed(ActionEvent e) {
        repaint();
    }


public boolean BusquedaAnchura(int[][] mapa, int x, int y) {
    return BusquedaAnchuraHelper(mapa, x, y);
}

private boolean BusquedaAnchuraHelper(int[][] mapa, int x, int y) {
    Queue<Integer> queue = new LinkedList<>();
    Map<Integer, Integer> visited = new HashMap<>();

    int startKey = getKey(x, y, mapa[0].length);
    queue.add(startKey);
```

```
    visited.put(startKey, -1); // Usamos -1 para representar el inicio

    while (!queue.isEmpty()) {
        int currentKey = queue.poll();
        int currentX = getX(currentKey, mapa[0].length);
        int currentY = getY(currentKey, mapa[0].length);

        if (mapa[currentY][currentX] == 9) { // Encontró la salida
            reconstructPath(visited, currentKey);
            return true;
        }

        int[] Nodos = {0, 1, 0, -1, 1, 0, -1, 0}; // Movimientos posibles (derecha, abajo, izquierda,
arriba)

        for (int i = 0; i < Nodos.length; i += 2) {
            int nextX = currentX + Nodos[i];
            int nextY = currentY + Nodos[i + 1];
            int nextKey = getKey(nextX, nextY, mapa[0].length);

            if (isValid(nextX, nextY, mapa) && !visited.containsKey(nextKey) &&
mapa[nextY][nextX] != 1) {
                queue.add(nextKey);
                visited.put(nextKey, currentKey);
            }
        }
    }

    return false; // No se encontró un camino
}

// Métodos auxiliares para manejar las coordenadas y claves
private int getKey(int x, int y, int width) {
    return y * width + x;
}

private int getX(int key, int width) {
    return key % width;
}

private int getY(int key, int width) {
    return key / width;
}

private boolean isValid(int x, int y, int[][] mapa) {
    return x >= 0 && x < mapa[0].length && y >= 0 && y < mapa.length;
}
```

```
private void reconstructPath(Map<Integer, Integer> visited, int endKey) {
    int currentKey = endKey;
    List<Integer> reversedCamino = new ArrayList<>(); // Lista temporal para almacenar el
camino invertido

    while (currentKey != -1) {
        int currentX = getX(currentKey, mapa[0].length);
        int currentY = getY(currentKey, mapa[0].length);
        reversedCamino.add(currentX);
        reversedCamino.add(currentY);
        currentKey = visited.get(currentKey);
    }

    // Invertir la lista antes de agregarla al camino
    for (int i = reversedCamino.size() - 2; i >= 0; i -= 2) {
        camino.add(reversedCamino.get(i));
        camino.add(reversedCamino.get(i + 1));
    }
}
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (IllegalAccessException ex) {
```

```
java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Main.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }

        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Main().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    // End of variables declaration
}
```

## LaberintoPred

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package org.busquedaprofunda;
import java.util.Random;
/**
 *
 * @author Medaf
 */
public class laberintosPred {
    public static int[][] obtenerLaberintoAleatorioDeLista() {
    Random random = new Random();
    int numeroAleatorio = random.nextInt(obtenerCantidadLaberintos()) + 1;
    return obtenerLaberinto(numeroAleatorio);
}
    public static int[][] obtenerLaberinto(int numero) {
      if (numero < 1 || numero > 10) {
        throw new IllegalArgumentException("Número de laberinto no válido");
```

```
    }

   switch (numero) {
    case 1:
       return laberinto1();
    case 2:
       return laberinto2();
    case 3:
       return laberinto3();
    case 4:
       return laberinto4();
    case 5:
       return laberinto5();
    case 6:
       return laberinto6();
    case 7:
       return laberinto7();
    case 8:
       return laberinto8();
    case 9:
       return laberinto9();
    case 10:
       return laberinto10();
    default:
       throw new IllegalArgumentException("Número de laberinto no válido");
}
}

private static int[][] laberinto1() {
    int[][] laberinto = {
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
        {1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1},
        {1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1},
        {1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1},
        {1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1},
        {1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1},
        {1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1},
        {1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1},
        {1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 9, 1},
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
    };
    return laberinto;
}
private static int[][] laberinto2() {
    int[][] laberinto = {
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
```

```
    {1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 9, 1},
    {1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1},
    {1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1},
    {1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1},
    {1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1},
    {1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1},
    {1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
    };
    return laberinto;
}

private static int[][] laberinto3() {
    int[][] laberinto = {
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1},
    {1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1},
    {1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1},
    {1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1},
    {1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1},
    {1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1},
    {1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1},
    {1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1},
    {1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 9, 1}
    };
    return laberinto;
}
private static int[][] laberinto4() {
    int[][] laberinto = {
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 9, 1},
    {1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1},
    {1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1},
    {1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1},
    {1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1},
    {1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1},
    {1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1},
    {1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1},
    {1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1}
    };
    return laberinto;
}
private static int[][] laberinto5() {
    int[][] laberinto = {
```

```
    {1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1},
    {1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1},
    {1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1},
    {1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1},
    {1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1},
    {1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1},
    {1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1},
    {9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
    {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
    };
    return laberinto;
}
private static int[][] laberinto6() {
    int[][] laberinto = {
        {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
        {1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1},
        {1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1},
        {1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1},
        {1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1},
        {1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1},
        {1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1},
        {1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1},
        {1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1},
        {1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1},
        {1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1},
        {1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1},
        {1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1},
        {1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1},
        {1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1},
        {1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1},
        {1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1},
        {1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1},
        {1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1},
        {1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 9, 1}

    };
    return laberinto;
}
private static int[][] laberinto7() {
    int[][] laberinto = {
        // Definición del laberinto 7
        // ...
    };
    return laberinto;
}
private static int[][] laberinto8() {
```

```
    int[][] laberinto = {
        // Definición del laberinto 8
        // ...
    };
    return laberinto;
}
private static int[][] laberinto9() {
    int[][] laberinto = {
        // Definición del laberinto 9
        // ...
    };
    return laberinto;
}



    private static int[][] laberinto10() {
        int[][] laberinto = {
            // Definición del laberinto 10
            // ...
        };

        return laberinto;
    }

     public static int obtenerCantidadLaberintos() {
        return 6; // Número total de laberintos predeterminados
    }
}
```

## Conclusión.

El proyecto ha proporcionado una experiencia práctica en la implementación de algoritmos de búsqueda en el ámbito de la inteligencia artificial. Se logró una comprensión más profunda de los algoritmos de búsqueda por profundidad y anchura, así como su aplicación en la resolución de problemas complejos, como el recorrido en laberintos.

La interfaz gráfica ha demostrado ser una herramienta valiosa para la visualización y comprensión del proceso de búsqueda, lo que facilita el análisis del rendimiento de cada algoritmo. La capacidad de cambiar entre diferentes laberintos enriquece la aplicación al permitir pruebas y comparaciones exhaustivas.