*UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO*

*FACULTAD DE INGENIERIA Y CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES*

INTELIGENCIA ARTIFICIAL

CONTENIDO: LAB-3. LABERINTO CON LISTA DE REGLAS.

PORCENTAJE TERMINADO: 100%

| INTEGRANTES | DT | HG | HI | EVAL |
|---|---|---|---|---|
| | | | | |
| Garcia Taboada Brayan Albaro | 1 | 1 | 1 | 100 |

**Fecha de presentación** : Jueves, 9 de Mayo de 2024

**Fecha Presentada :** : Jueves, 9 de Mayo de 2024

**Días de Atraso : 0**

# EL PROBLEMA DEL LABERINTO.

TRABAJO INDIVIDUAL.

1. Dado una matriz de n x m, inicialmente todas las posiciones con valores de cero, avanzar las casillas en sentido horario con movimientos de izquierda, arriba, derecha y abajo. Resolver este problema utilizando el Algoritmo de llamadas recursivas dentro de otro ciclo.

a) Movimientos en sentido horario, izquierda, arriba, derecha y abajo.

```
public static LinkedList<Regla> reglasAplicables(int m[][], int i, int j) {

    LinkedList<Regla> L1 = new LinkedList();

    if (posValida(m, i, j - 1)) {

        L1.add(new Regla(i, j - 1));

    }

    if (posValida(m, i - 1, j)) {

        L1.add(new Regla(i - 1, j));

    }

    if (posValida(m, i, j + 1)) {

        L1.add(new Regla(i, j + 1));

    }

    if (posValida(m, i + 1, j)) {

        L1.add(new Regla(i + 1, j));

    }

    return L1;

}
```

b) Ampliar los movimientos por las diagonales.

```
public static LinkedList<Regla> reglasAplicables(int m[][], int i, int j) {
    LinkedList<Regla> L1 = new LinkedList();
    if (posValida(m, i, j - 1)) {
        L1.add(new Regla(i, j - 1));
    }
        if (posValida(m, i-1, j - 1)) {
        L1.add(new Regla(i-1, j - 1));
    }
    if (posValida(m, i - 1, j)) {
        L1.add(new Regla(i - 1, j));
    }
        if (posValida(m, i - 1, j+1)) {
        L1.add(new Regla(i - 1, j+1));
    }
    if (posValida(m, i, j + 1)) {
        L1.add(new Regla(i, j + 1));
    }

    if (posValida(m, i+1, j + 1)) {
        L1.add(new Regla(i+1, j + 1));
    }
    if (posValida(m, i + 1, j)) {
        L1.add(new Regla(i + 1, j));
```

```
        }


        if (posValida(m, i + 1, j-1)) {
            L1.add(new Regla(i + 1, j-1));
        }
        return L1;
    }
```

c) Aplicar el movimiento del Caballo.

```
    public static LinkedList<Regla> reglasAplicables(int m[][], int i, int j) {
        LinkedList<Regla> L1 = new LinkedList();
        if (posValida(m, i - 2, j - 1)) {
            L1.add(new Regla(i - 2, j - 1));
        }
        if (posValida(m, i - 2, j + 1)) {
            L1.add(new Regla(i - 2, j + 1));
        }
        if (posValida(m, i - 1, j + 2)) {
            L1.add(new Regla(i - 1, j + 2));
        }
        if (posValida(m, i + 1, j + 2)) {
            L1.add(new Regla(i + 1, j + 2));
        }


        if (posValida(m, i + 2, j + 1)) {
```

```
        L1.add(new Regla(i + 2, j + 1));
    }
    if (posValida(m, i + 2, j - 1)) {
        L1.add(new Regla(i + 2, j - 1));
    }
    if (posValida(m, i + 1, j - 2)) {
        L1.add(new Regla(i + 1, j - 2));
    }
    if (posValida(m, i - 1, j - 2)) {
        L1.add(new Regla(i - 1, j - 2));
    }
    return L1;
  }
```

Mostrar todos los caminos posible desde una posición inicial a una posición final. Las salidas de estos Algoritmos deberán ser idénticos a las salidas del problema del laberinto con la lógica anterior.

```
public class Laberintos {


  public static void main(String[] args) {
    int a = 5, b = 3;
    int m[][] = new int[a][b];
    laberinto(m, 0, 0, a - 1, b - 1, 1);
  }


  public static void mostrar(int m[][]) {
```

```
    for (int i = 0; i < m.length; i++) {
        for (int j = 0; j < m[i].length; j++) {
            System.out.print(m[i][j] + "-");
        }
        System.out.println();
    }
    System.out.println();
}

public static void laberinto(int m[][], int i, int j, int iF, int jF, int paso) {
    m[i][j] = paso;
    if (i == iF && j == jF) {
        mostrar(m);
    }
    LinkedList<Regla> L1 = reglasAplicables(m, i, j);
    while (!L1.isEmpty()) {
        Regla R = elegirRegla(L1);
        laberinto(m, R.fil, R.col, iF, jF, paso + 1);
        m[R.fil][R.col] = 0;
    }
}

public static boolean posValida(int m[][], int i, int j) {
    return i >= 0 && i < m.length
            && j >= 0 && j < m[i].length && m[i][j] == 0;
}
```

```
    public static Regla elegirRegla(LinkedList<Regla> L1) {
        return L1.removeFirst();
    }

    public static LinkedList<Regla> reglasAplicables(int m[][], int i, int j) {
        LinkedList<Regla> L1 = new LinkedList();
        if (posValida(m, i, j - 1)) {
            L1.add(new Regla(i, j - 1));
        }
        if (posValida(m, i - 1, j)) {
            L1.add(new Regla(i - 1, j));
        }
        if (posValida(m, i, j + 1)) {
            L1.add(new Regla(i, j + 1));
        }
        if (posValida(m, i + 1, j)) {
            L1.add(new Regla(i + 1, j));
        }
        return L1;
    }
}

class Regla {

    public int fil;
```

```
    public int col;



    public Regla(int fil, int col) {

        this.fil = fil;

        this.col = col;

    }

}
```