UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO

FACULTAD DE INGENIERIA Y CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES

INTELIGENCIA ARTIFICIAL

CONTENIDO:

TAREA-3. RECORRIDOS SOBRE EL LABERINTO.

PORCENTAJE TERMINADO: 90%

INTEGRANTES	DT	HG	HI	EVAL
Garcia Taboada Brayan Albaro	1	1	2	90

Fecha de presentación : Lunes, 13 de Mayo de 2024

Fecha Presentada : : Lunes, 13 de Mayo de 2024

Días de Atraso : 0

RECORRIDOS DE LABERINTOS.

RECORRIDOS DE LABERINTOS.

TRABAJO INDIVIDUAL.

PROBLEMA DEL LABERINTO.

Hay dos forma de recorrer los caminos: uno, sin utilizar ciclos para direccionar el camino y otro utilizando ciclos para direccionar la elección de caminos(Lista de Reglas).

Utilizando las dos formas implementar

a. Sin utilizar Lista de Reglas

1. Implementar recorridos en sentido HORARIO, para recorridos:

- ***Laberinto Simple.***

```
public static void laberinto(int m[][], int i1, int j1, int i2, int j2, int paso) {
    if (!posValida(m, i1, j1)) {
        return;
    }
    m[i1][j1] = paso;
    if (i1 == i2 && j1 == j2) {
        mostrar(m);
        soluciones++;
```

```
    }
    laberinto(m, i1, j1 - 1, i2, j2, paso + 1);
    laberinto(m, i1 - 1, j1, i2, j2, paso + 1);
    laberinto(m, i1, j1 + 1, i2, j2, paso + 1);
    laberinto(m, i1 + 1, j1, i2, j2, paso + 1);
    m[i1][j1] = 0;
}
```

- *Laberinto que también recorre por las diagonales.*

```
public static void laberinto(int m[][], int i1, int j1, int i2, int j2, int paso) {
    if (!posValida(m, i1, j1)) {
        return;
    }
    m[i1][j1] = paso;
    if (i1 == i2 && j1 == j2) {
        mostrar(m);
        soluciones++;
    }
    laberinto(m, i1, j1 - 1, i2, j2, paso + 1);
    laberinto(m, i1 - 1, j1 - 1, i2, j2, paso + 1);
    laberinto(m, i1 - 1, j1, i2, j2, paso + 1);
    laberinto(m, i1 + 1, j1 - 1, i2, j2, paso + 1);
    laberinto(m, i1, j1 + 1, i2, j2, paso + 1);
    laberinto(m, i1 + 1, j1 + 1, i2, j2, paso + 1);
    laberinto(m, i1 + 1, j1, i2, j2, paso + 1);
    laberinto(m, i1 + 1, j1 - 1, i2, j2, paso + 1);
```

```
    m[i1][j1] = 0;
  }
```

- *Laberinto con los movimientos del sentido de Salto de Caballo.*

```
  public static void laberinto(int m[][], int i1, int j1, int i2, int j2, int paso) {
    if (!posValida(m, i1, j1)) {
      return;
    }
    m[i1][j1] = paso;
    if (i1 == i2 && j1 == j2) {
      mostrar(m);
      soluciones++;
    }
    laberinto(m, i1-2, j1 - 1, i2, j2, paso + 1);
    laberinto(m, i1 - 2, j1 +1, i2, j2, paso + 1);
    laberinto(m, i1 - 1, j1+2, i2, j2, paso + 1);
    laberinto(m, i1 + 1, j1 +2, i2, j2, paso + 1);
    laberinto(m, i1+ 2, j1+1, i2, j2, paso + 1);
    laberinto(m, i1 + 2, j1 - 1, i2, j2, paso + 1);
    laberinto(m, i1 + 1, j1- 2, i2, j2, paso + 1);
    laberinto(m, i1 - 1, j1 - 2, i2, j2, paso + 1);
    m[i1][j1] = 0;
  }
```

Hacer estos Recorridos para:
a) Sin Atajos.

```
  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;
```

```
    }
```

b) Con Atajos.

```
    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 ;
    }
```

2. Implementar recorridos en sentido ANTI-HORARIO, para recorridos:

- Laberinto Simple.

```
    public static void laberinto(int m[][], int i1, int j1, int i2, int j2, int paso) {

        if (!posValida(m, i1, j1)) {

            return;

        }

        m[i1][j1] = paso;

        if (i1 == i2 && j1 == j2) {

            mostrar(m);

            soluciones++;

        }

        laberinto(m, i1, j1 - 1, i2, j2, paso + 1);

        laberinto(m, i1 + 1, j1, i2, j2, paso + 1);

        laberinto(m, i1, j1 + 1, i2, j2, paso + 1);

        laberinto(m, i1 - 1, j1, i2, j2, paso + 1);

        m[i1][j1] = 0;

    }
```

- *Laberinto que también recorre por las diagonales.*

```
public static void laberinto(int m[][], int i1, int j1, int i2, int j2, int paso) {
    if (!posValida(m, i1, j1)) {
        return;
    }
    m[i1][j1] = paso;
    if (i1 == i2 && j1 == j2) {
        mostrar(m);
        soluciones++;
    }
    laberinto(m, i1, j1 - 1, i2, j2, paso + 1);
    laberinto(m, i1 + 1, j1 - 1, i2, j2, paso + 1);
    laberinto(m, i1 + 1, j1, i2, j2, paso + 1);
    laberinto(m, i1 + 1, j1 + 1, i2, j2, paso + 1);
    laberinto(m, i1, j1 + 1, i2, j2, paso + 1);
    laberinto(m, i1 - 1, j1 + 1, i2, j2, paso + 1);
    laberinto(m, i1 - 1, j1, i2, j2, paso + 1);
    laberinto(m, i1 - 1, j1 - 1, i2, j2, paso + 1);
    m[i1][j1] = 0;
}
```

- *Laberinto con los movimientos del sentido de Salto de Caballo.*

```
public static void laberinto(int m[][], int i1, int j1, int i2, int j2, int paso) {
    if (!posValida(m, i1, j1)) {
        return;
    }
```

```
        m[i1][j1] = paso;
        if (i1 == i2 && j1 == j2) {
            mostrar(m);
            soluciones++;
        }
        laberinto(m, i1-2, j1 + 1, i2, j2, paso + 1);
        laberinto(m, i1 - 2, j1 -1, i2, j2, paso + 1);
        laberinto(m, i1 - 1, j1-2, i2, j2, paso + 1);
        laberinto(m, i1 + 1, j1 -2, i2, j2, paso + 1);
        laberinto(m, i1+ 2, j1 - 1, i2, j2, paso + 1);
        laberinto(m, i1 + 2, j1 + 1, i2, j2, paso + 1);
        laberinto(m, i1 + 1, j1+ 2, i2, j2, paso + 1);
        laberinto(m, i1 - 1, j1 + 2, i2, j2, paso + 1);
        m[i1][j1] = 0;
    }
```

Hacer estos Recorridos para:
a) Sin Atajos.
b) Con Atajos.

b. Utilizando Lista de Reglas

1. Implementar recorridos en sentido HORARIO, para recorridos:

- Laberinto Simple.

```
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 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;
    }
```

- *Laberinto que también recorre por las diagonales.*

```
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 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 - 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;
    }
```

- *Laberinto con los movimientos del sentido de Salto de Caballo.*

```
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 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 - 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;
  }
```

Hacer estos Recorridos para:
a) Sin Atajos.
b) Con Atajos.

2. Implementar recorridos en sentido ANTI-HORARIO, para recorridos:

- Laberinto Simple.

```
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 Regla elegirRegla(LinkedList<Regla> L1) {
    return L1.removeLast();
  }

  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;
    }
```

- *Laberinto que también recorre por las diagonales.*

```
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 Regla elegirRegla(LinkedList<Regla> L1) {
    return L1.removeLast();
}



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;
  }
```

- *Laberinto con los movimientos del sentido de Salto de Caballo.*

```
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 Regla elegirRegla(LinkedList<Regla> L1) {
    return L1.removeLast();
}

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;
    }
```

Hacer estos Recorridos para:
a) Sin Atajos.
b) Con Atajos.