Laboratorio #3

Grupo: 12

Integrantes:

- **Añez Vladimirovna Leonardo Henry**
- Cardona Chavez Alex †
- Mercado Oudalova Erick Denis
- Mollinedo Franco Milena

Materia: Inteligencia Artificial

Fecha: 7 de septiembre de 2020

Porcentaje Completado: 100 %

Comentario(s): En esta practica vimos el problema del laberinto aplicado en multiples situaciones (tal es el caso del alfil, la reina, la torre, el caballo y el rey) y varias formas en su comportamiento (como exploracion de todas las soluciones, de un punto inicial a un final arbitrario y con atajos). Realizamos los problemas a traves de dos enfoques, el primero a punta de `if`s y el segundo a traves de un conjunto de **reglas** que nos permiten flexibilizar el comportamiento en la exploracion del laberinto. Se realizó ademas una implementación para poder simular las soluciones de algunos de los problemas planteados en la tarea de manera mas visual.
Proyecto: https://github.com/toborochi/Personal-Projects/tree/master/Java/IAGridGrapher

A. Todos los caminos posibles

Mostrar todos los caminos posibles que se recorren para los problemas del laberinto, movimiento del rey, movimiento del caballo, con un punto de partida de cualquier posición de la matriz.

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

    public static void laberintoCaballo(int m[][], int i, int j, int i1, int j1,
       int paso) {
        if (!posValida(m, i, j)) {
            return;
        }
        m[i][j] = paso;
        if ((i == i1 && j == j1)) {
            mostrarcaba(m);
        }
        laberintoCaballo(m, i + 1, j - 3, i1, j1, paso + 1);
        laberintoCaballo(m, i - 1, j - 3, i1, j1, paso + 1);
        laberintoCaballo(m, i - 3, j - 1, i1, j1, paso + 1);
        laberintoCaballo(m, i - 3, j + 1, i1, j1, paso + 1);
        laberintoCaballo(m, i - 1, j + 3, i1, j1, paso + 1);
        laberintoCaballo(m, i + 1, j + 3, i1, j1, paso + 1);
        laberintoCaballo(m, i + 3, j + 1, i1, j1, paso + 1);
        laberintoCaballo(m, i + 3, j - 1, i1, j1, paso + 1);
        m[i][j] = 0;
    }
 private 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;
    }

    private static void mostrar(int[][] m, List po) {
        int t = 0;
        for (int i = 0; i < m.length; i++) {
```

```
            for (int j = 0; j < m[i].length; j++) {
                System.out.print("" + m[i][j] + '\t');
                if (m[i][j] != 0) {
                    t++;
                }
            }
            System.out.println(" ");
        }
        po.add(t);
        System.out.println(" ");
        System.out.println("Longuitud :" + t);
        System.out.println(" ");
    }
```

B. Todas las casillas de la matriz

Mostrar las matrices completas con su recorrido, tal que visite a todas las casillas de la matriz, con un punto de partida de cualquier posición de la matriz.

```
// 1 Laberinto ejercicio NRO 2
import java.util.*;

public class Main {
    public static void main(String[] args) {
        int m[][] = new int [2][2];
        laberinto(m, 0, 0, 1);
    }

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

    public static void mostrar(int m[][]){
        for(int i=0; i<m.length; i++){
            for(int j=0; j<m.length; j++){
                System.out.print("\t"+m[i][j]);
            }
            System.out.println("");
        }
        System.out.println("");
    }

    public static void laberinto(int m[][], int i, int j, int paso){
        if(!posvalida(m,i,j)){return;}
        m[i][j] = paso;
        mostrar(m);
        laberinto(m, i, j-1, paso+1);
        laberinto(m, i-1, j, paso+1);
        laberinto(m, i, j+1, paso+1);
        laberinto(m, i+1, j, paso+1);

        m[i][j]=0;
    }

}
```

C. Caminos entre dos puntos

Mostrar todos los caminos desde un punto inicial a un punto final cualquiera.

```java
// 1 Laberinto ejercicio NRO 3
import java.util.*;

public class Main {
    public static void main(String[] args) {
        int m[][] = new int [2][2];
        laberintoIFcualquiera(m, 0, 0, 1, 1, 1);
    }

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

    public static void mostrar(int m[][]){
        for(int i=0; i<m.length; i++){
            for(int j=0; j<m.length; j++){
                System.out.print("\t"+m[i][j]);
            }
            System.out.println("");
        }
        System.out.println("");
    }

    public static void laberintoIFcualquiera(int m[][], int i, int j, int ifin,
       int jfin, int paso){
        if(!posvalida(m,i,j)){return;}
        m[i][j] = paso;
        if(i == ifin && j==jfin){mostrar(m);}

        laberintoIFcualquiera(m, i, j-1, ifin, jfin, paso+1);
        laberintoIFcualquiera(m, i-1, j, ifin, jfin, paso+1);
        laberintoIFcualquiera(m, i, j+1, ifin, jfin, paso+1);
        laberintoIFcualquiera(m, i+1, j, ifin, jfin, paso+1);

        m[i][j]=0;
    }

}
```

D. Atajos

Incluir atajos a la matriz y ejecutar los algoritmos de a), b) y c).

```
{ 0, 0, 0, 0, 0}
{-1, 0, 0, 0, 0}
{ 0,-1,-1, 0, 0}
{ 0,-1,-1, 0, 0}
{ 0, 0,-1, 0,-1}
```

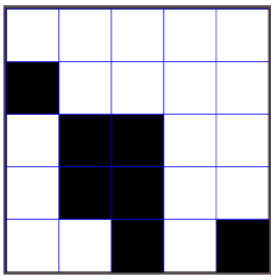


Figura 1: Matriz y su representación mas visual.

En este problema las pruebas se realizaron para la matriz anterior.

Ejercicio 1:

Para el caso del laberinto ordinario.

```
1 0 0 0 0
-1 0 0 0 0
0 -1 -1 0 0
0 -1 -1 0 0
0 0 -1 0 -1

1 2 0 0 0
-1 0 0 0 0
0 -1 -1 0 0
0 -1 -1 0 0
0 0 -1 0 -1

...

1 2 0 0 0
-1 3 4 5 0
0 -1 -1 6 0
0 -1 -1 7 0
0 0 -1 8 -1

Total: 198
```

Ejercicio 2:

Para el caso del rey.

```
1 2 5 4 0
-1 0 3 6 7
0 -1 -1 0 0
0 -1 -1 0 0
0 0 -1 0 -1

1 2 5 4 8
-1 0 3 6 7
0 -1 -1 0 0
0 -1 -1 0 0
0 0 -1 0 -1

...

1 0 0 0 0
-1 2 0 0 0
3 -1 -1 0 0
4 -1 -1 0 0
5 6 -1 0 -1

Total: 43253
```

Ejercicio 3:

Para el caso de los movimientos del caballo.

```
1 0 0 6 3
-1 5 2 0 0
0 -1 -1 4 7
0 -1 -1 0 0
0 0 -1 0 -1

1 0 0 6 3
-1 5 2 0 0
0 -1 -1 4 7
0 -1 -1 0 0
0 0 -1 8 -1

...

1 0 5 0 7
-1 0 2 0 4
0 -1 -1 6 0
0 -1 -1 3 0
0 0 -1 0 -1

Total: 62
```

Ejercicio 4:

Para el caso de todas las celdas se tuvo que realizar un ligero cambio al laberinto, de la siguiente manera:

```
{  0, 0, 0, 0, 0}
{ -1, 0, 0, 0, 0}
{ -1,-1,-1, 0, 0}
{ -1,-1,-1, 0, 0}
{ -1,-1,-1,-1,-1}
```

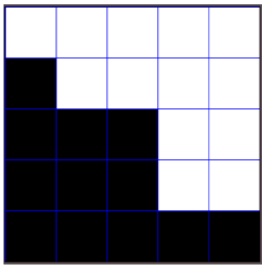


Figura 2: Matriz y su representación mas visual.

```
1 2 3 4 5
-1 13 12 11 6
-1 -1 -1 10 7
-1 -1 -1 9 8
-1 -1 -1 -1 -1
```

```
1 2 5 6 7
-1 3 4 9 8
-1 -1 -1 10 11
-1 -1 -1 13 12
-1 -1 -1 -1 -1
```

```
1 2 5 6 7
-1 3 4 9 8
-1 -1 -1 10 13
-1 -1 -1 11 12
-1 -1 -1 -1 -1
```

```
1 2 5 6 7
-1 3 4 13 8
-1 -1 -1 12 9
-1 -1 -1 11 10
-1 -1 -1 -1 -1
```

```
1 2 5 6 13
-1 3 4 7 12
-1 -1 -1 8 11
-1 -1 -1 9 10
-1 -1 -1 -1 -1
```

```
1 2 13 12 11
-1 3 4 5 10
```

```
-1 -1 -1 6 9
-1 -1 -1 7 8
-1 -1 -1 -1 -1
```

```
Total: 6
```

E. Modificación de los anteriores puntos

Modificar la estructura de los algoritmos anteriores y ejecutar bajo los mismos casos y encontrar las mismas salidas. (El Algoritmos modificado, deberá tener una estructura de llamadas recursivas dentro de un ciclo).

```
private 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;
    }

    private static LinkedList<Regla> reglasAplicablesRey(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;
    }

    private static LinkedList<Regla> reglasAplicablesCaballo(int m[][], int i, int
        j) {
        LinkedList<Regla> L1 = new LinkedList();

        // Izquierda
        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));
        }

        // Arriba
        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));
        }

        // Derecha
        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));
        }

        // Abajo
```

```
        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));
        }

        return L1;
    }

    private 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 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 void laberintoA(int m[][], int i, int j, int paso) {
        if (!posValida(m, i, j)) {
            return;
        }
        m[i][j] = paso;
        mostrar(m);
        LinkedList<Regla> L1 = reglasAplicables(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            laberintoA(m, R.fil, R.col, paso + 1);
            m[R.fil][R.col] = 0;
        }
    }

    public static void laberintoARey(int m[][], int i, int j, int paso) {
        if (!posValida(m, i, j)) {
            return;
        }
        m[i][j] = paso;
```

```
        mostrar(m);
        LinkedList<Regla> L1 = reglasAplicablesRey(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            laberintoARey(m, R.fil, R.col, paso + 1);
            m[R.fil][R.col] = 0;
        }
    }

    public static void laberintoACaballo(int m[][], int i, int j, int paso) {
        if (!posValida(m, i, j)) {
            return;
        }
        m[i][j] = paso;
        mostrar(m);
        LinkedList<Regla> L1 = reglasAplicablesCaballo(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            laberintoACaballo(m, R.fil, R.col, paso + 1);
            m[R.fil][R.col] = 0;
        }
    }



    public static void laberintoB(int m[][], int i, int j, int paso) {
        if (!posValida(m, i, j)) {
            return;
        }
        m[i][j] = paso;
        if (paso==m.length*m[0].length) {
            mostrar(m);
        }
        LinkedList<Regla> L1 = reglasAplicables(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            laberintoB(m, R.fil, R.col, paso + 1);
            m[R.fil][R.col] = 0;
        }
    }

    public static void laberintoC(int m[][], int i, int j, int ifin, int jfin, int
       paso) {
        if (!posValida(m, i, j)) {
            return;
```

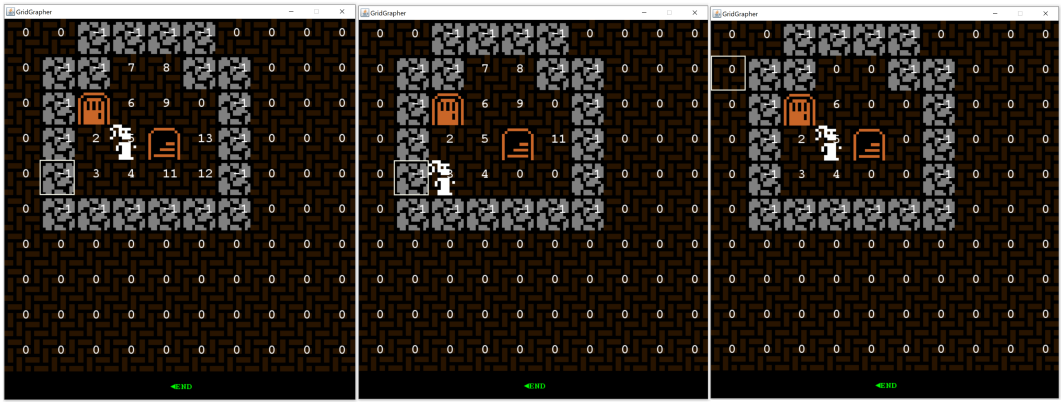
```
        }
        m[i][j] = paso;
        if (i == ifin && j == jfin) {
            mostrar(m);
        }
        LinkedList<Regla> L1 = reglasAplicables(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            laberintoC(m, R.fil, R.col, ifin, jfin, paso + 1);
            m[R.fil][R.col] = 0;
        }
    }
```

II. Torre, Alfil y Reina

Dado una matriz de n x m, inicialmente con valores de ceros. Implementar un Algoritmo para resolver los incisos a), b), c), d) utilizando e). Para los problemas del movimiento de la torre, movimiento del alfil, movimiento de la dama.

Ejercicio 1:

Para la **torre**, realizar todos los recorridos, todos los recorridos tal que visite todas las celdas, todos los caminos desde un punto inicial a un final arbitrario y caminos con atajos.



```
// TORRE
     private LinkedList<Regla> reglasAplicablesTorre(int m[][], int i, int j) {
        LinkedList<Regla> L1 = new LinkedList();

        int pi,pj;

        pi=i-1;
        pj=j;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi--;
        }

        pi=i;
        pj=j-1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pj--;
        }

        pi=i+1;
        pj=j;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
```

```
            pi++;
        }



        pi=i;
        pj=j+1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pj++;
        }







        return L1;
    }

     // TODOS LOS CAMINOS POSIBLES
    private void torreA(int m[][], int i, int j, int paso, LinkedList<Point> acum)
       {

        if (!posValida(m, i, j)) {
            return;
        }
        m[i][j] = paso;

        // ANADIR ESTO
        //mostrar(m);
        steps.add(paso);
        acum.addLast(new Point(i, j));
        int[][] copy = Arrays.stream(m).map(int[]::clone).toArray(int[][]::new);
        results.add(copy);
        for (int k = 0; k < acum.size(); ++k) {
            ins.add((Point) acum.get(k).clone());
        }

        LinkedList<Regla> L1 = reglasAplicablesTorre(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            torreA(m, R.fil, R.col, paso + 1, acum);
            m[R.fil][R.col] = 0;

        }
```

```
        // ANADIR ESTO
        acum.removeLast();
    }

    private void torreB(int m[][], int i, int j, int paso, LinkedList<Point> acum)
        {

        if (!posValida(m, i, j)) {
            return;
        }
        acum.addLast(new Point(i, j));
        m[i][j] = paso;

        if (paso == floodSize) {
            // ANADIR ESTO
            //mostrar(m);
            steps.add(paso);

            int[][] copy =
                Arrays.stream(m).map(int[]::clone).toArray(int[][]::new);
            results.add(copy);
            for (int k = 0; k < acum.size(); ++k) {
                ins.add((Point) acum.get(k).clone());
            }
        }
        LinkedList<Regla> L1 = reglasAplicablesTorre(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            torreB(m, R.fil, R.col, paso + 1, acum);
            m[R.fil][R.col] = 0;

        }

        // ANADIR ESTO
        acum.removeLast();
    }


    private void torreC(int m[][], int i, int j, int ifin, int jfin, int paso,
        LinkedList<Point> acum) {

        if (!posValida(m, i, j)) {
            return;
        }
```

```
        // ANADIR ESTO
        acum.addLast(new Point(i, j));

        m[i][j] = paso;
        if (i == ifin && j == jfin) {
            c++;
            steps.add(paso);
            int[][] copy =
                Arrays.stream(m).map(int[]::clone).toArray(int[][]::new);
            results.add(copy);
            //mostrar(copy);
            for (int k = 0; k < acum.size(); ++k) {
                ins.add((Point) acum.get(k).clone());
            }

        }

        LinkedList<Regla> L1 = reglasAplicablesTorre(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            torreC(m, R.fil, R.col, ifin, jfin, paso + 1, acum);
            m[R.fil][R.col] = 0;

        }

        // ANADIR ESTO
        acum.removeLast();
    }
```

Ejercicio 2:

Para el **alfil**, realizar todos los recorridos, todos los recorridos tal que visite todas las celdas, todos los caminos desde un punto inicial a un final arbitrario y caminos con atajos.

```
// ALFIL
     private LinkedList<Regla> reglasAplicablesAlfil(int m[][], int i, int j) {
        LinkedList<Regla> L1 = new LinkedList();

        int pi,pj;

        pi=i-1;
        pj=j-1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi--;
            pj--;
        }

        pi=i+1;
        pj=j-1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi++;
            pj--;
        }

        pi=i+1;
        pj=j+1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi++;
            pj++;
        }
```

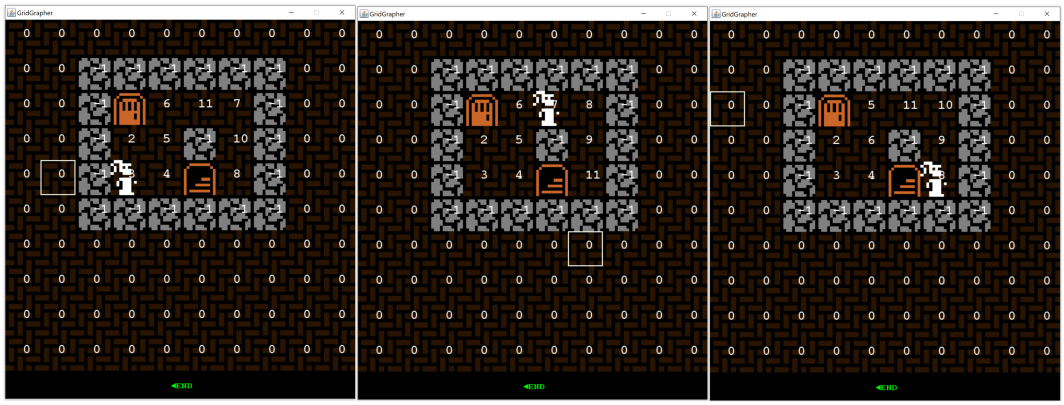
```
        pi=i-1;
        pj=j+1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi--;
            pj++;
        }



        return L1;
    }

      // TODOS LOS CAMINOS POSIBLES
    private void alfilA(int m[][], int i, int j, int paso, LinkedList<Point> acum)
       {

        if (!posValida(m, i, j)) {
            return;
        }
        m[i][j] = paso;

        // ANADIR ESTO
        //mostrar(m);
        steps.add(paso);
        acum.addLast(new Point(i, j));
        int[][] copy = Arrays.stream(m).map(int[]::clone).toArray(int[][]::new);
        results.add(copy);
        for (int k = 0; k < acum.size(); ++k) {
            ins.add((Point) acum.get(k).clone());
        }

        LinkedList<Regla> L1 = reglasAplicablesAlfil(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            alfilA(m, R.fil, R.col, paso + 1, acum);
            m[R.fil][R.col] = 0;

        }

        // ANADIR ESTO
        acum.removeLast();
    }

    // TODOS LOS CAMINOS QUE RECORREN TODAS LAS CELDAS
```

```
    private void alfilB(int m[][], int i, int j, int paso, LinkedList<Point> acum)
        {

        if (!posValida(m, i, j)) {
            return;
        }
        acum.addLast(new Point(i, j));
        m[i][j] = paso;

        if (paso == floodSize) {
            // ANADIR ESTO
            //mostrar(m);
            steps.add(paso);

            int[][] copy =
                Arrays.stream(m).map(int[]::clone).toArray(int[][]::new);
            results.add(copy);
            for (int k = 0; k < acum.size(); ++k) {
                ins.add((Point) acum.get(k).clone());
            }
        }
        LinkedList<Regla> L1 = reglasAplicablesAlfil(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            alfilB(m, R.fil, R.col, paso + 1, acum);
            m[R.fil][R.col] = 0;

        }

        // ANADIR ESTO
        acum.removeLast();
    }

    // TODOS LOS CAMINOS DE PUNTO INICIO A FIN (ARBITRARIOS)
    private void alfilC(int m[][], int i, int j, int ifin, int jfin, int paso,
        LinkedList<Point> acum) {

        if (!posValida(m, i, j)) {
            return;
        }

        // ANADIR ESTO
        acum.addLast(new Point(i, j));

        m[i][j] = paso;
```

```
        if (i == ifin && j == jfin) {
            c++;
            steps.add(paso);
            int[][] copy =
                Arrays.stream(m).map(int[]::clone).toArray(int[][]::new);
            results.add(copy);
            //mostrar(copy);
            for (int k = 0; k < acum.size(); ++k) {
                ins.add((Point) acum.get(k).clone());
            }

        }

        LinkedList<Regla> L1 = reglasAplicablesAlfil(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            alfilC(m, R.fil, R.col, ifin, jfin, paso + 1, acum);
            m[R.fil][R.col] = 0;

        }

        // ANADIR ESTO
        acum.removeLast();
    }
```

Ejercicio 3:

Para la **reina**, realizar todos los recorridos, todos los recorridos tal que visite todas las celdas, todos los caminos desde un punto inicial a un final arbitrario y caminos con atajos.



```
// REINA
     private LinkedList<Regla> reglasAplicablesReina(int m[][], int i, int j) {
        LinkedList<Regla> L1 = new LinkedList();

        int pi,pj;

        pi=i-1;
        pj=j;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi--;
        }

        pi=i-1;
        pj=j-1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi--;
            pj--;
        }

        pi=i;
        pj=j-1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pj--;
        }

        pi=i+1;
        pj=j-1;
```

```
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi++;
            pj--;
        }

        pi=i+1;
        pj=j;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi++;
        }

        pi=i+1;
        pj=j+1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi++;
            pj++;
        }

        pi=i;
        pj=j+1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pj++;
        }

        pi=i-1;
        pj=j+1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi--;
            pj++;
        }



        return L1;
    }

     private void reinaA(int m[][], int i, int j, int paso, LinkedList<Point>
        acum) {

        if (!posValida(m, i, j)) {
```

```
            return;
        }
        m[i][j] = paso;

        // ANADIR ESTO
        //mostrar(m);
        steps.add(paso);
        acum.addLast(new Point(i, j));
        int[][] copy = Arrays.stream(m).map(int[]::clone).toArray(int[][]::new);
        results.add(copy);
        for (int k = 0; k < acum.size(); ++k) {
            ins.add((Point) acum.get(k).clone());
        }

        LinkedList<Regla> L1 = reglasAplicablesReina(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            reinaA(m, R.fil, R.col, paso + 1, acum);
            m[R.fil][R.col] = 0;

        }

        // ANADIR ESTO
        acum.removeLast();
    }

    private void reinaB(int m[][], int i, int j, int paso, LinkedList<Point>
        acum) {

        if (!posValida(m, i, j)) {
            return;
        }
        acum.addLast(new Point(i, j));
        m[i][j] = paso;

        if (paso == floodSize) {
            // ANADIR ESTO
            //mostrar(m);
            steps.add(paso);

            int[][] copy =
                Arrays.stream(m).map(int[]::clone).toArray(int[][]::new);
            results.add(copy);
            for (int k = 0; k < acum.size(); ++k) {
                ins.add((Point) acum.get(k).clone());
```

```
            }
        }
        LinkedList<Regla> L1 = reglasAplicablesReina(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            reinaB(m, R.fil, R.col, paso + 1, acum);
            m[R.fil][R.col] = 0;

        }

        // ANADIR ESTO
        acum.removeLast();
    }


     private void reinaC(int m[][], int i, int j, int ifin, int jfin, int paso,
        LinkedList<Point> acum) {

        if (!posValida(m, i, j)) {
            return;
        }

        // ANADIR ESTO
        acum.addLast(new Point(i, j));

        m[i][j] = paso;
        if (i == ifin && j == jfin) {
            c++;
            steps.add(paso);
            int[][] copy =
                Arrays.stream(m).map(int[]::clone).toArray(int[][]::new);
            results.add(copy);
            //mostrar(copy);
            for (int k = 0; k < acum.size(); ++k) {
                ins.add((Point) acum.get(k).clone());
            }

        }

        LinkedList<Regla> L1 = reglasAplicablesReina(m, i, j);
        while (!L1.isEmpty()) {
            Regla R = L1.removeFirst();
            reinaC(m, R.fil, R.col, ifin, jfin, paso + 1, acum);
            m[R.fil][R.col] = 0;
```

```
        }

        // ANADIR ESTO
        acum.removeLast();
    }
```