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;
        }
    }
```