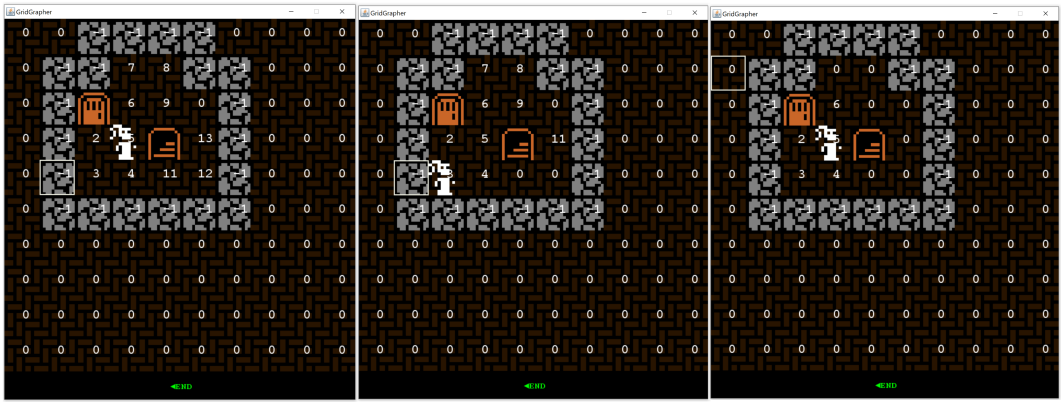
II. Torre, Alfil y Reina

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++;
        }
```

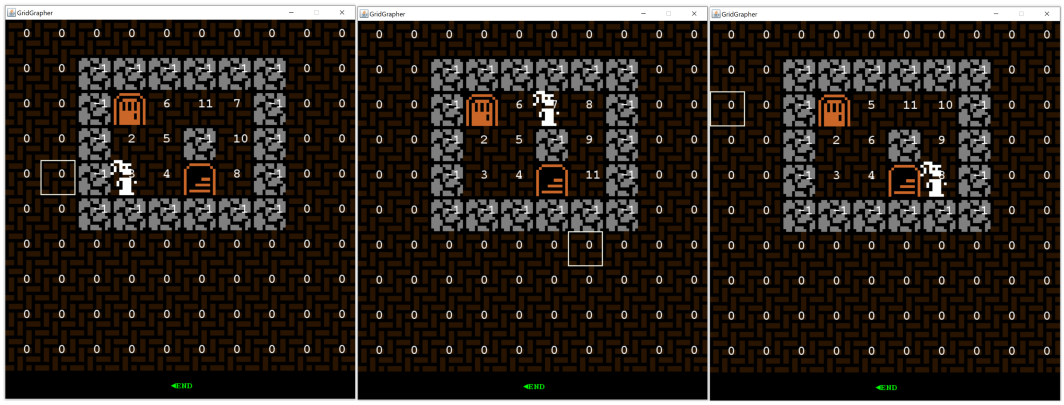
```
        pi=i-1;
        pj=j+1;
        while(posValida(m, pi, pj)){
            L1.add(new Regla(pi,pj));
            pi--;
            pj++;
        }



        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();
    }
```