ENCRIPTADOR Y DESENCRIPTADOR DE IMÁGENES USANDO AUTÓMATAS CELULARES

MÁSTER UNIVERSITARIO EN LÓGICA, COMPUTACIÓN E INTELIGENCIA ARTIFICIAL
MÉTODOS COMPUTACIONALES EN VIDA ARTIFICIAL

DAVID PÉREZ RIVERO

INDICE

1. INTRODUCCIÓN

Hoy en día, la seguridad es uno de los aspectos más importantes del mundo tecnológico. Por ello, a partir del uso de autómatas celulares, he construido un encriptado y desencriptador de imágenes.

Después de estudiar y comprobar distintas variantes de encriptadores de imágenes basados en autómatas celulares, he implementado uno capaz de encriptar imágenes a color, que utiliza el juego de la vida como regla para pasar de un estado actual al estado siguiente, y que usa la vecindad de Moore para comprobar el estado de las células que se encuentran alrededor de la célula a estudiar.

Algunos de los encriptadores de imágenes estudiados realizaban la separación de las componentes del color de cada pixel y otros sólo eran capaz de trabajar con imágenes en escala de grises. En cuanto a las reglas que utilizaban eran distintas, pero me he decantado por éste porque utiliza el juego de la vida, que ha sido explicado durante la asignatura. Por lo que me he decantado por implementar el encriptador y desencriptador de imágenes basado en autómatas celulares que se describe a continuación.

2. AUTÓMATA CELULAR

Un autómata celular es una dupla que contiene los siguientes objetos:

- Una **rejilla** donde cada celda es reconoce como una célula. En este caso, es una rejilla en dos dimensiones de tamaño m x n. Donde m es el alto y n el ancho. A partir de ahora, se referirá a esta rejilla como tablero.

- Cada célula toma un **valor.** El valor que puede tomar en este caso es 1 o 0.

- Tiene un **estado actual** (S) y un **estado siguiente** (S+).

- Cada célula se caracteriza por su **vecindad**. (Apartado 5)

- Una **función de transición** que toma el valor de la célula actual y el valor de las células de su vecindad, devolviendo el nuevo valor de la célula en el estado siguiente. En este caso,

3. ENCRIPTAR IMAGEN

El encriptador implementado consta de cuatro pasos:

1- Se comienza generando una matriz binaria pseudoaleatoria de tamaño m x n a partir de una semilla. El tamaño de la matriz es igual al del número de píxeles de la imagen donde m es el alto y n el ancho. Esta matriz pasará a ser el estado actual (S_o) del autómata celular.

```java
/**
 * Genera una matriz aleatoria a partir de la semilla
 */
private void generateRandomMatrix() {
    bitInit = new int[m][n];
    Random random = new Random();
    random.setSeed(seed);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            bitInit[i][j] = (random.nextBoolean()) ? 1 : 0;
        }
    }
}
```

2- Una vez generado el estado actual, se obtiene el estado siguiente (S+) a través del uso de un autómata celular que sigue las reglas del juego de la vida (las condiciones se explican en el apartado 6). La vecindad utilizada para este encriptador ha sido Moore Neighborhood, que se explica en el apartado 5.

```
// Obtiene el numero de celulas vivas en la vecindad
//      de la celula actual
int alive = getNumOfNeighborsAlive(i, j);
// Obtiene el estado siguiente
getNextState(alive, i, j, nextState);
```

3- A medida que vamos generando el estado siguiente, se comprueba si la celda actual está viva o no. En caso afirmativo se almacena el color del pixel de la celda actual en un array de colores de células vivas. En caso negativo, se almacena el color del pixel de la celdas actual en un array de colores de células muertas.

```
if (nextState[i][j] == 1) {
    // Si la celula en el estado siguiente esta viva
    //      se copia en la estructura de celulas vivas
    aliveCell.add(image.getRGB(i, j));
} else {
    // Si esta muerta en el estado siguiente,
    //      se copia en la estructura de celulas muertas
    deadCell.add(image.getRGB(i, j));
}
```

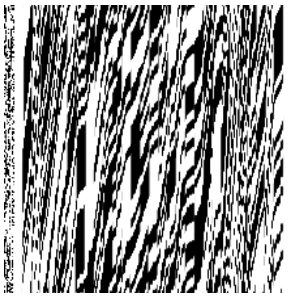
4- A continuación, se van colocando todos los colores del array de colores de células vivas en primer lugar, y después, los colores del array de colores de células muertas.

```
// Va estableciendo en los n primeros pixeles los colores de las
//      n celulas vivas y a continuacion los colores de las restantes
//      celulas muertas
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        if (!aliveCell.isEmpty()) {
            image.setRGB(i, j, aliveCell.remove(0));
        } else {
            image.setRGB(i, j, deadCell.remove(0));
        }
    }
}
```

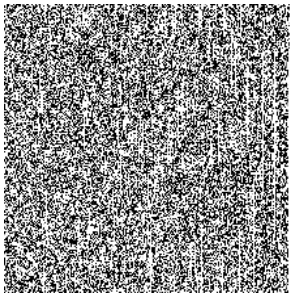
Se repiten los pasos 2-4 tantas veces como vueltas (rounds) se hayan establecido.

3.1 Ejemplos de encriptación

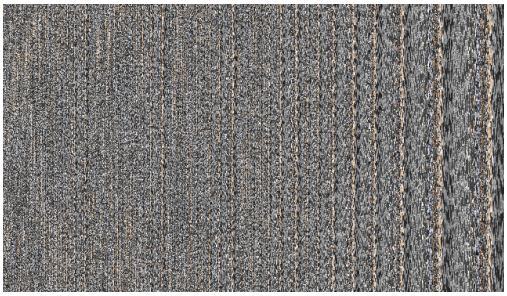
A continuación, se muestra un ejemplo del algoritmo de encriptación con una imagen de un código bidi. Tenemos por un lado la imagen original (imagen a la izquierda) y por otro lado, la imagen encriptada (imagen a la derecha). Usando el algoritmo descrito anteriormente, utilizando los siguientes valores para los parámetros: el algoritmo realiza 4 vueltas, la vecindad tiene un radio igual a 2 y el valor de la semilla es igual a 29121994.

Podemos ver otro ejemplo utilizando distintos parámetros y comprobar como cambia la imagen encriptada (imagen a la derecha). En este caso, se utiliza un radio igual a 1, 10 vueltas del algoritmo y la misma semilla (29121994).

Por último, tenemos un ejemplo con una imagen de un cachorro de tigre a color, para comprobar que el algoritmo acepta este tipo de imágenes. Utilizando como valor de radio un 1, 7 vueltas y una semilla como valor igual a 999888, obtenemos el siguiente resultado:

4. DESENCRIPTAR IMAGEN

Para desencriptar la imagen se realiza el algoritmo de forma inversa.

Antes de comenzar, es necesario seleccionar la imagen y el fichero de texto encriptado que contiene la información sobre los valores de los parámetros. Este fichero es desencriptado y se establecen los parámetros. A continuación, comienza el algoritmo de desencriptar la imagen, que se compone por los siguientes pasos:

1- En primer lugar, se genera la matriz binaria pseudoaleatoria así como tantos estados siguientes como vueltas se hayan establecido, y se van almacenando.

```
/**
 * Obtiene todos los estados, incluyendo el inicial que es el
 * obtenido mediante la generacion de la matriz aleatoria
 */
private void getStates() {
    // Genera la matriz aleatoria
    generateRandomMatrix();
    // Estado siguiente
    nextState = new int[m][n];
    // Bucles para obtener los rounds estados siguientes
    for (int k = 0; k < rounds; k++) {
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                // Obtiene vecinos vivos de la celula actual
                int alive = getNumOfNeighborsAlive(i, j);
                // Obtiene el estado siguiente de la celula actual
                getNextState(alive, i, j, nextState);
            }
        }
        // Realiza una copia del estado siguiente y la establece
        // como estado actual
        bitInit = copyState(nextState);
        // Almacena el estado actual
        states.add(new State(bitInit, 0, new ArrayList<Integer>()));
    }
}
```

2- En segundo lugar, se recorren en sentido inverso al generado, es decir, se comienza por el último estado siguiente generado. Luego se van contando el número de células vivas y almacenando los colores de la imagen en el orden en el que se van presentando.

```
// Obtiene el ultimo estado no utilizado
State state = states.get(rounds - 1 - k);
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        // Obtiene los colores de la imagen actual
        state.getColors().add(image.getRGB(i, j));
        if (state.getState()[i][j] == 1) {
            // Si la celula esta viva, incrementa un contador
            state.increaseCount();
        }
    }
}
```

3- Por último, si la célula está viva se coge el color del principio, en caso contrario el color en la posición que marca el contador de células vivas.

```
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        // Si la celula esta viva, saca el primer color,
        // decrementando el contador
        if (state.getState()[i][j] == 1) {
            image.setRGB(i, j, state.getColors().remove(0));
            state.decreaseCount();
        } else {
            // Si la celula esta muerta, saca el color apuntado
            // por el contador
            image.setRGB(i, j, state.getColors().remove(state.getCount()));
        }
    }
}
```

4.1 Ejemplo de desencriptación y encriptación

Suponemos que tenemos una imagen y una matriz binaria aleatoria como se muestra a continuación:

1	0	1
0	1	0
1	1	1

R	G	B
R	G	B
R	G	B

Al ejecutar el algoritmo de encriptación, se obtiene la matriz de estado siguiente y la imagen encriptada con la siguiente disposición:

0	1	0
1	0	1
0	0	0

G	R	B
R	B	G
R	G	B

Ahora, queremos obtener la imagen original. Para ello, en primer lugar, generamos la matriz aleatoria con la misma semilla y los estados siguientes, en este caso, solo uno. Se muestran a continuación por orden, la matriz aleatoria, la de estado siguiente y la imagen encriptada.

1	0	1
0	1	0
1	1	1

0	1	0
1	0	1
0	0	0

G	R	B
R	B	G
R	G	B

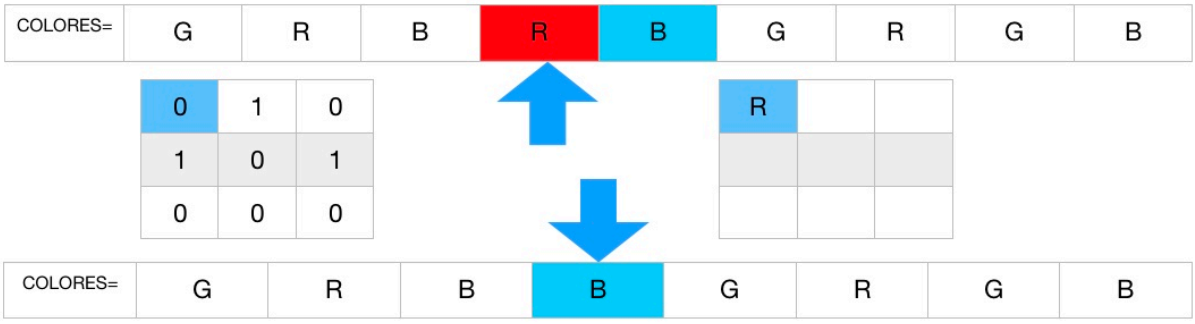
Una vez generados los estados, almacenamos los colores de la imagen encriptada, y contamos el número de células vivas del último estado generado. En el caso que nos ocupa, el número de células vivas es igual a 3.

COLORES=	G	R	B	R	B	G	R	G	B

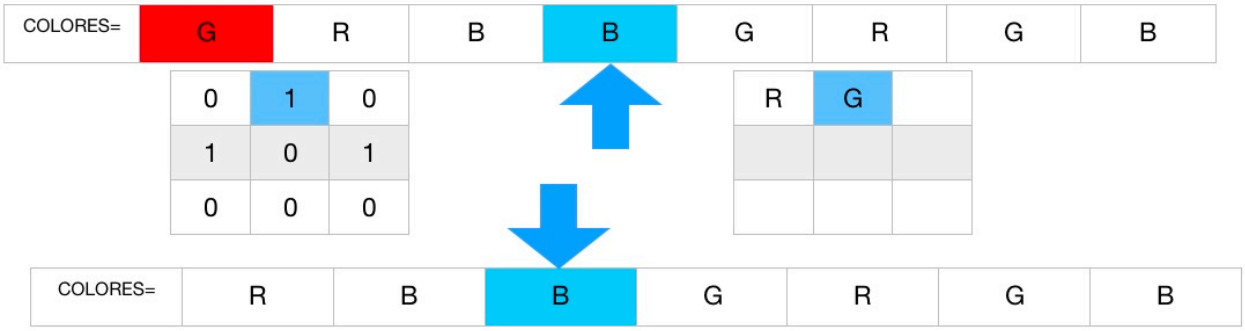
Por último, lo único que queda hacer es ir comprobando cada célula del estado actual (el último en ser generado). Si está viva, se coge el color que esté en primer lugar, se pone en la imagen, se elimina del almacenamiento y se decrementa en una unidad el valor del contador de células vivas. Si está muerta, se coge el color en la posición apuntada por el contador de células vivas, teniendo en cuenta que comienza a indexarse por 0.

A continuación, se muestra como se comportaría el algoritmo para la primera fila de la imagen:

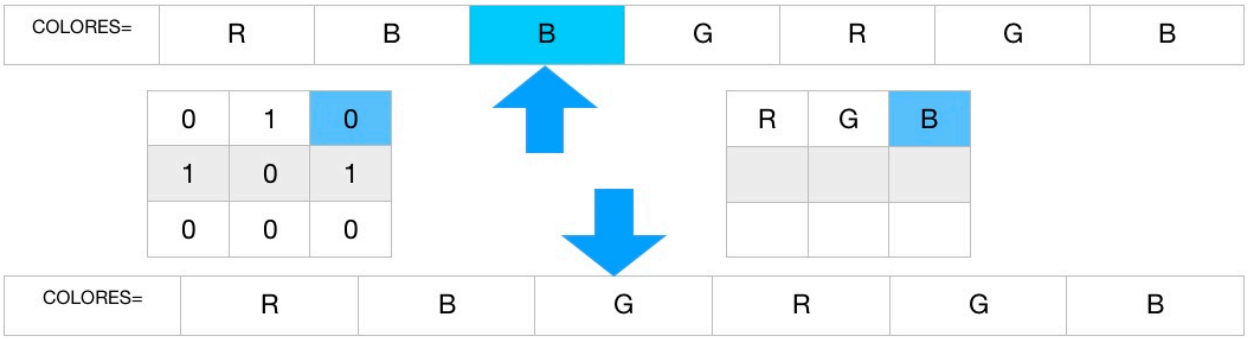
1- Pixel (1,1): El valor del contador es igual a 3. Por lo que el elemento apuntado es el cuarto. Al estar la célula muerta (tiene valor 0 en la matriz de estado), se coge el elemento en la posición apuntada por el contador y se establece en la imagen. Se elimina el elemento en la posición apuntada por el contador, por lo que será el siguiente elemento quien ocupe su lugar.



2- Pixel (1,2): El valor del contador es igual a 3. Al estar la célula viva (tiene valor 1 en la matriz de estado), se coge el elemento en primer lugar y se establece en la imagen. Se elimina el elemento de la primera posición y se decrementa en una unidad el contador.

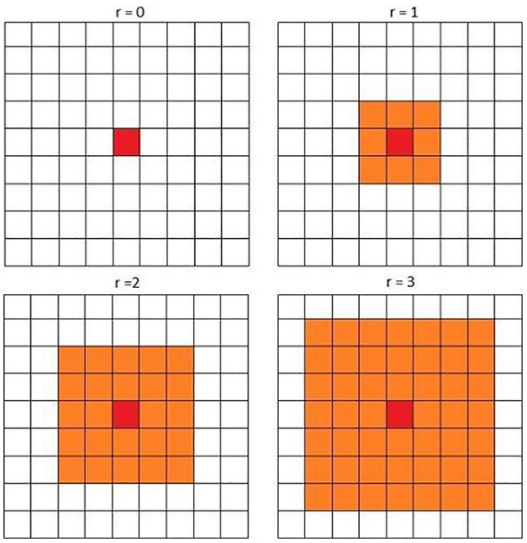


3- Pixel (1,3): El valor del contador es igual a 2, ya que sufrió un decremento en el paso anterior, pero sigue apuntando al mismo elemento que al principio del paso anterior. Al estar la célula muerta, se coge el elemento en la posición apuntada por el contador y se establece en la imagen. Luego, se elimina del almacenamiento.

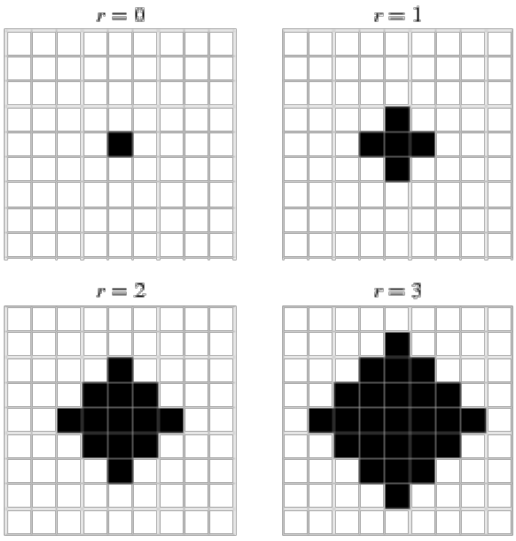


5. VECINDARIO

En este proyecto se ha utilizado la vecindad de Moore (Moore Neighbordhood), que comprueba un cuadrado del radio establecido alrededor de la célula actual. En la imagen que se muestra a continuación, podemos ver en color rojo la célula actual y en naranja las células a comprobar según el radio que varía entre 0 y 3. En este proyecto, solo se comprobaba si la célula vecina estaba viva o no en el siguiente estado.



Durante el desarrollo de este proyecto, también se estudió la vecindad de Von Neumann (Von Neumann Neighbordhood), que consiste en comprobar los elementos a una distancia de Manhattan igual al radio establecido, formando un rombo. En la imagen que se muestra debajo de estas líneas, puede verse las distintas vecindades con el radio variando entre 0 y 3.



En primer lugar, me decanté por la vecindad de Moore por la facilidad de implementación. Una vez implementada, se implementó la vecindad de Von Neumann pero no se percibió ninguna mejora, por lo que se descartó al dar las dos vecindades un resultado muy similar.

A continuación, se puede observar el código implementado para contabilizar el número de células vivas alrededor de la actual. Para ello, se comprueba si existe o no la casilla vecina, ya que puede ser, por ejemplo, que se encuentre en una casilla situada en la esquina superior izquierda y no tenga casillas vecinas hacia arriba ni hacia la izquierda.

```
/**
 * Obtiene el numero de vecinos vivos en la vecindad de la celula i,j.
 *
 * Para obtener este numero, se utiliza la vecindad de moore, que observa
 * un cuadrado de radio radius alrededor de la celula
 */
public int getNumOfNeighborsAlive(int i, int j) {
    int alive = 0;
    for (int r = radius * -1 + i; r <= radius + i; r++) {
        for (int c = radius * -1 + j; c <= radius + j; c++) {
            if (r != i && c != j) {
                if (r >= 0 && r < m && c >= 0 && c < n && bitInit[r][c] == 1) {
                    alive++;
                }
            }
        }
    }
    return alive;
}
```

6. FUNCIÓN DE TRANSICIÓN

En cuanto al autómata celular, se trata de un autómata celular bidimensional, ya que se mueve en dos direcciones para comprobar la vecindad, y sigue las reglas del juego de la vida.

El juego de la vida consiste en un tablero formado por m x n células, cada una con un estado actual (viva o muerta). A partir del estado actual de cada célula y el de sus vecinas, se genera el estado siguiente mediante turnos, es decir, se actualiza simultáneamente en cada ronda. Las reglas que sigue el juego de la vida son las siguientes:

- Una célula muerta con exactamente tres células vecinas vivas nace en el siguiente turno.
- Una célula viva con dos o tres células vecinas vivas, sigue viva en el siguiente turno.
- Una célula viva con menos de dos o más de tres células vecinas vivas muere en el siguiente turno.

Podemos observar, que al utilizar un radio igual o mayor que tres, la imagen a encriptar se mantiene prácticamente igual. Esto es debido a que la probabilidad de encontrarse con más de tres células vivas en 49 celdas (utilizando la vecindad de Moore con radio igual a tres) es prácticamente 1, por lo que todas las células mueren en el siguiente turno y no se produce el desorden por colores de células vivas y muertas. Al utilizar la vecindad de Von Neumann, aunque se tienen en cuenta menos células, 25 células para ser exactos, la probabilidad de encontrar más de tres células vivas es muy alta y no se producen cambios en la siguiente ronda del algoritmo.

A continuación, se muestra el código relativo a la generación del estado siguiente:

```
/**
 * Genera el siguiente estado a partir de la celula actual
 * y los vecinos vivos en su vecindad.
 *
 * La generacion del estado siguiente se basa en el algoritmo
 * del juego de la vida
 *
 * @param alive     Numero de celulas vivas
 * @param i         Fila de la celula actual
 * @param j         Columna de la celula actual
 * @param next      Matriz de estados
 */
public void getNextState(int alive, int i, int j, int[][] next) {
    if ((bitInit[i][j]) == 0 && (alive == 3)) {
        next[i][j] = 1;
    } else if (bitInit[i][j] == 1 && (alive == 2 || alive == 3)) {
        next[i][j] = 1;
    } else {
        next[i][j] = 0;
    }
}
```

7. USOS Y APLICACIONES

El proyecto desarrollado, como función principal, actúa como encriptador de imágenes para poder ser transferidas de forma segura, evitando que si algún usuario la intercepta, sepa el contenido de la misma.

A partir de esta idea, un uso que se le puede dar sería el utilizar esta herramienta con códigos BIDI. De esta manera, incorporándola en una aplicación móvil, obligaría al usuario a descargarse la aplicación específica que contenga el desencriptador de un código BIDI dado. Por ejemplo, si una cadena de comida rápida incluyera de descuentos mediante códigos BIDI encriptados, obligaría al usuario a utilizar la aplicación de esta cadena de comida rápida para desencriptar el código BIDI y obtener el descuento.

8. REPOSITORIO

El código entero del proyecto puede ser encontrado en un repositorio GitHub, junto a las instrucciones de uso de la herramienta, así como este documento.

El repositorio puede encontrarse en el siguiente enlace: **https://github.com/.**

9. REFERENCIAS

- **Enlace** al pdf que explica la técnica del encriptador de imágenes.

- **Enlace** a la Wikipedia sobre el Juego de la Vida.

- **Enlace** a la Wikipedia sobre Autómatas Celulares.

- **Enlace** al encriptador de textos utilizado en la herramienta.