Informe: Primer Reto de Informática II

1st Daniela Escobar Velandia
Dpto. de Ingeniería Electrónica y Telecomunicaciones
Universidad de Antioquia
Medellín, Colombia
daniela.escobarv@udea.edu.co

2nd Yonathan López Mejía
Dpto. de Ingeniería Electrónica y Telecomunicaciones
Universidad de Antioquia
Medellin, Colombia
harley.lopez@udea.edu.co

***Abstract*—Este informe presenta una estrategia para la recuperación de imágenes digitales modificadas mediante operaciones a nivel de bits. A partir de una imagen cifrada y archivos auxiliares generados durante el proceso de transformación, se plantea un enfoque de ingeniería inversa que permite identificar y aplicar las operaciones inversas en el orden adecuado. Se detallan los criterios utilizados para evaluar la similitud entre imágenes, así como los algoritmos empleados para revertir transformaciones como rotaciones, desplazamientos y operaciones XOR. El documento incluye análisis, diagramas y resultados experimentales que validan la efectividad del método propuesto.**

***Keywords*—Ingeniería inversa, transformaciones bit a bit, rotación de bits, operación XOR, rotaciones de bits, desplazamientos de bits, enmascaramiento de imágenes, distancia de Hamming.**

I. Introducción

El presente trabajo aborda un reto centrado en la recuperación de imágenes digitales que han sido alteradas mediante transformaciones bit a bit. El proceso involucra operaciones como *XOR*, *rotaciones* y *desplazamientos*, aplicadas de forma secuencial pero en un orden no conocido previamente.

El objetivo consiste en aplicar técnicas de ingeniería inversa para identificar la naturaleza y el orden de las transformaciones efectuadas, apoyándose en archivos auxiliares generados durante el cifrado, como máscaras y datos de enmascaramiento. Una vez identificadas las operaciones, se procede a revertirlas de manera sistemática hasta obtener una imagen lo más cercana posible a la original.

A lo largo del informe se describe el camino que se siguió para restaurar las imágenes, mostrando cómo se pensó la solución, qué métodos se usaron y cuáles fueron los principales retos a la hora de llevar la idea a la práctica. También se presentan los resultados obtenidos y se analizan las situaciones en las que el programa funcionó bien y aquellas en las que hubo dificultades.

II. Propuesta de solución

A. Condiciones y restricciones

Para la realización del reto se conoce de forma directa o derivada lo siguiente:

- El número de operaciones realizadas sobre la imagen original.
- La imagen resultado de la última operación: $\texttt{P}_\texttt{N}\texttt{.bmp}$.
- Los archivos de enmascaramiento: $M_{n-1}\texttt{.txt}$, que incluyen la semilla aplicada y el resultado.
- La imagen máscara: `M.bmp`.
- La imagen de entropía: $\texttt{I}_\texttt{M}\texttt{.bmp}$.
- El número máximo de operaciones aplicadas: XOR, desplazamientos o rotaciones a izquierda o derecha de hasta 8 bits.
- La operación de enmascaramiento $s(k) = I_D(k+s) + M(k)$ con s una semilla aleatoria y positiva, I_D la imagen transformada al momento del enmascaramiento y M la imagen máscara.

Se desconoce:

- El orden de las operaciones aplicadas.
- La imagen objetivo.
- Las imagenes resultado de aplicar las operaciones intermedias hasta llegar a $\texttt{P}_\texttt{N}\texttt{.bmp}$.

B. Operaciones que mantienen la integridad de los datos

En cualquier sistema de cifrado, se debe cumplir que cualquier conjunto de N datos debe ser íntegramente recuperable. En el caso del reto, no se implementan estrategias durante el cifrado que garanticen la integridad de los datos, como sumas de verificación o códigos de detección de errores. Por lo tanto, la integridad de los datos recuperados estará garantizada únicamente si las operaciones a nivel de bits son completamente reversibles, esto es: al aplicar su inversa se recuperan todos y cada uno de los bits originales.

Un ejemplo de operación reversible es la operación `XOR`. Por ejemplo:

```
1011 XOR 0011 = 1000
1000 XOR 0011 = 1011
```

Como se observa, al aplicar la misma operación `XOR` con el mismo operando (`0011`) sobre el resultado, se recuperan los 4 bits originales (`1011`).

Otro ejemplo de operación reversible es la rotación de bits. Supongamos una rotación hacia la izquierda (`ROL`) de un valor de 4 bits:

```
1011 ROL 2 = 1110
1110 ROR 2 = 1011
```

En este caso, al rotar `1011` dos bits a la izquierda, se obtiene `1110`. Luego, al aplicar una rotación de dos bits a la derecha (`ROR`) sobre `1110`, se recupera el valor original `1011`. Por lo tanto, la operación es reversible siempre que se conozcan la dirección y la cantidad de desplazamiento.

A diferencia de las operaciones anteriores, el desplazamiento de bits (`shift`) no siempre es reversible, ya que puede provocar pérdida de información. Por ejemplo, consideremos un desplazamiento lógico a la derecha de un valor de 4 bits:

`1011 SHR 2 = 0010`

En este caso, al desplazar `1011` dos bits a la derecha, se obtiene `0010`. Sin embargo, no es posible recuperar los dos bits más significativos que fueron descartados (los bits que estaban originalmente en las posiciones altas). Por lo tanto, esta operación en general no es reversible pues usualmente al realizar desplazamientos a izquierda o derecha se perderán bits 1s no recuperables. La Tabla I resume las operaciones.

Tabla I
REVERSIBILIDAD DE LAS OPERACIONES OPERACIONES BIT A BIT

Operación	¿Es reversible?
XOR	Sí
Rotación de bits (ROL/ROR)	Sí
Desplazamiento (SHL/SHR)	No

C. Criterio de verosimilitud

1) Diferencia absoluta: Una imagen, al igual que un vector en álgebra lineal, será igual a otra si coinciden en todos sus píxeles. En el contexto del reto, es necesario definir un criterio que permita determinar si dos imágenes (o regiones de una imagen) pueden considerarse equivalentes.

Una forma común de abordar este problema consiste en realizar una resta pixel a pixel: dos imágenes se consideran iguales si la suma total de las diferencias es cero. Sin embargo, este método resulta insuficiente cuando las operaciones aplicadas durante el procesamiento no garantizan la integridad de los datos, ya que pequeñas alteraciones pueden impedir una coincidencia exacta incluso si la estructura general se conserva.

Algorithm 1: Comparación pixel a pixel entre dos porciones de imagen

Input: ImagenA, ImagenB
Output: SonIguales (booleano)

if *dimensiones(ImagenA)* $\neq$ *dimensiones(ImagenB)* **then**
  return `falso`
suma_diferencias $\leftarrow$ 0
for *cada píxel* (i, j) *en las imágenes* **do**
  diferencia $\leftarrow$ |ImagenA[i][j] $-$ ImagenB[i][j]| ;
  suma_diferencias $\leftarrow$ suma_diferencias $+$ diferencia ;
if *suma_diferencias* $=$ *0* **then**
  return `verdadero`
else
  return `falso`

2) Uso de la distancia de Hamming como criterio de similitud: Para afrontar el problema derivado de la pérdida de datos, se considera que dos bytes son más similares entre sí si su **distancia de Hamming** es mínima.

Dada dos secuencias de bits de igual longitud, la **distancia de Hamming** se define como el número de posiciones en las que los bits correspondientes son diferentes. Por ejemplo:

`1011` y `1100` $\Rightarrow$ distancia de Hamming = 2

En el contexto del reto, esta aproximación es más robusta frente a la pérdida de información que una simple resta, ya que tolera diferencias mínimas a nivel de bit. Sin embargo, pueden ocurrir *colisiones*, es decir, múltiples tramas con la misma distancia mínima respecto a una trama de referencia. Por ejemplo:

- `1111` vs `0111` $\Rightarrow$ distancia = 1
- `1111` vs `1110` $\Rightarrow$ distancia = 1

En este caso, ¿cuál de las dos tramas es más similar a `1111`? Este tipo de colisiones puede ocurrir en el reto. El criterio de desempate adoptado será seleccionar aquella operación que realice la mayor cantidad de desplazamientos manteniendo la distancia de Hamming mínima.

Desde el punto de vista computacional, el cálculo de la distancia de Hamming es más costoso que una resta directa, ya que requiere contar el número de bits en 1 tras una operación XOR. El algoritmo de fuerza bruta para este conteo tiene una complejidad de $O(n)$, siendo n el número de bits de la trama.

Nosotros proponemos una optimización mediante el **algoritmo de Brian Kernighan**, que reduce la complejidad a $O(k)$, siendo k el número de bits en 1 del resultado, lo cual representa una mejora considerable en eficiencia cuando los valores son dispersos.

Algorithm 2: Conteo de bits en 1 usando el algoritmo de Brian Kernighan

Input: n: entero sin signo (representa la trama de bits)
Output: count: número de bits en 1

count $\leftarrow$ 0 ;
while *n* $\neq$ *0* **do**
  n $\leftarrow$ n **AND** (n $-$ 1) ;
  count $\leftarrow$ count $+$ 1 ;
return count

D. Propuesta de ingeniería inversa

Aunque no se conocen las imágenes resultantes de las transformaciones intermedias, sí se dispone de los archivos generados durante el proceso de enmascaramiento.

Con esto en mente, se propone el siguiente procedimiento para cada imagen transformada:

Sea $\texttt{P}_{\texttt{N}}$`.bmp` la última imagen obtenida tras aplicar una secuencia de transformaciones. A partir del archivo $\texttt{M}_{\texttt{N-1}}$`.txt`, se extraen la semilla y el resultado del enmascaramiento $s(k)$. Conociendo $s(k)$ y $m(k)$, es posible —utilizando la

fórmula de enmascaramiento— obtener $I_D(k+s)$, es decir, un fragmento de la imagen $\texttt{P}_{\texttt{N-1}}\texttt{.bmp}$ de la cual se derivó $\texttt{P}_{\texttt{N}}\texttt{.bmp}$.

A continuación, se realizan operaciones de bit sobre un segmento de $\texttt{P}_{\texttt{N}}\texttt{.bmp}$ que comienza en la posición $P(s)$ y tiene la misma longitud que $\texttt{M.bmp}$. Tras aplicar cada operación candidata, se calcula la distancia de Hamming total entre el resultado obtenido y el esperado. La operación seleccionada será aquella que produzca la menor distancia de Hamming. (Tabla II)

Una vez identificada la operación más probable, se aplica su inversa sobre $\texttt{P}_{\texttt{N}}\texttt{.bmp}$ para obtener $\texttt{P}_{\texttt{N-1}}\texttt{.bmp}$ en su totalidad. Este proceso se repite iterativamente hasta recuperar la imagen original.

Tabla II
OPERACIONES Y SUS INVERSAS

Operación	**Operación inversa**
XOR con clave I_M	XOR con clave I_M
Rotación a la izquierda ($\texttt{ROL}\ n$)	Rotación a la derecha ($\texttt{ROR}\ n$)
Rotación a la derecha ($\texttt{ROR}\ n$)	Rotación a la izquierda ($\texttt{ROL}\ n$)
Desplazamiento a la izquierda ($\texttt{SHL}\ n$)	$\texttt{SHR}\ n$ (parcialmente reversible)
Desplazamiento a la derecha ($\texttt{SHR}\ n$)	$\texttt{SHL}\ n$ (no reversible)

E. Algoritmo planteado

El diagrama del algoritmo planteado es el siguiente:

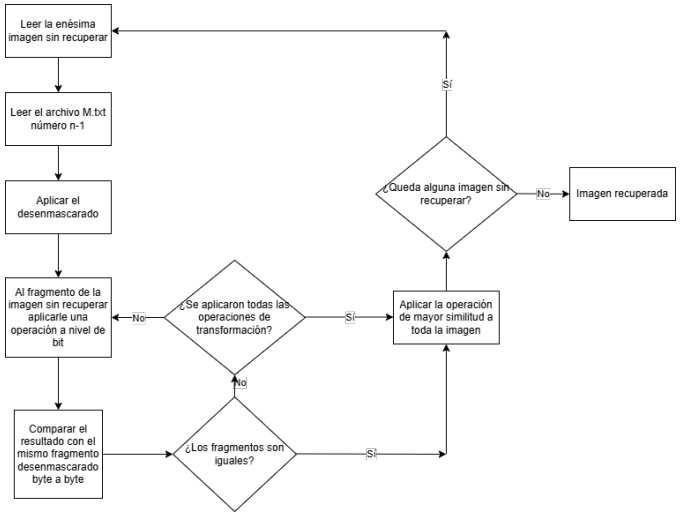

Fig. 1. Algoritmo planteado.

III. IMPLEMENTACIÓN

El código de la solución implementa 2 librerías propias, a saber:

A. Librería: bitwise pixel

Implementa las funciones asociadas a las operaciones de bits sobre las imagenes y el algoritmo de distancia de hamming. La librería incluye las siguientes funciones.

1) ***xor_byte, rotate_right_byte, rotate_left_byte, shift_right_byte, shift_left_byte****:* Estas subrutinas implementan las operaciones lógicas a nivel de bit indicadas en sus nombres. Cada una recibe dos operandos de 8 bits sin signo y devuelve un valor, también de 8 bits sin signo, que corresponde al resultado de aplicar la operación sobre el byte de datos. Estas funciones constituyen las operaciones bit a bit más elementales que se realizarán sobre las imágenes.

2) ***hamming_distance****:* Esta subrutina calcula la distancia de Hamming entre dos bytes, utilizando el algoritmo de Brian Kernighan. Retorna un valor entero que representa la cantidad de bits diferentes entre ambos operandos. Un resultado de 0 indica que los bytes son idénticos, mientras que valores mayores reflejan una menor similitud entre ellos.

3) ***validate_xor****:* Esta función valida el proceso de descifrado sobre una sección de la imagen usando XOR y utilizando una máscara revertida como referencia.

Para cada byte de la imagen, aplica una operación XOR entre los datos de la imagen transformada y la imagen con ruido. Luego, calcula la distancia de Hamming entre el resultado de dicha operación y el byte correspondiente de la máscara revertida. Este procedimiento se repite para todos los píxeles definidos por el tamaño de la máscara, considerando los tres canales de color (RGB).

Finalmente, retorna la suma total de las distancias de Hamming obtenidas, la cual representa el grado de diferencia entre los datos procesados y la máscara original. Un valor bajo indica una alta similitud, mientras que valores más altos reflejan mayor diferencia.

4) ***validate_rotate_shift_process****:* Esta función permite validar una operación bit a bit aplicada sobre una máscara invertida, comparando el resultado con los datos originales de la imagen. La operación específica (rotación o desplazamiento) se pasa como parámetro mediante una función op, la cual actúa sobre cada byte de la máscara utilizando un valor adicional n (el número de bits a rotar o desplazar).

Para cada posición, se aplica la operación op al byte correspondiente de la máscara invertida y se calcula la distancia de Hamming entre el resultado obtenido y el byte original de la imagen (desplazado una cantidad definida por seed). El procedimiento se repite para todos los bytes asociados a la máscara, incluyendo los tres canales de color RGB por píxel.

Finalmente, la función retorna la suma total de las distancias de Hamming, valor que cuantifica la discrepancia entre la máscara procesada y la imagen original. Un valor bajo indica una alta similitud, mientras que valores más elevados reflejan una mayor diferencia.

5) ***apply_complete_xor****:* Esta función aplica una operación XOR entre los bytes de dos imágenes del mismo tamaño: una imagen original y una imagen con ruido. El resultado de la operación se escribe directamente sobre los datos de la imagen original, modificándola.

Esta es la función que descifra las imagenes cifradas con XOR.

6) ***apply_complete_xor****:* Esta función aplica una operación bit a bit sobre cada byte de una imagen RGB. La operación

específica es proporcionada mediante una función op (rotación o desplazamiento), que recibe como parámetros el byte a procesar y un valor entero n que indica el número de bits a rotar o desplazar.

B. Librería: process_data

Esta librería se encarga de aplicar las operaciones de comparación necesarias durante el proceso de descifrado. Su objetivo principal es evaluar el nivel de similitud entre diferentes versiones de la imagen transformada, utilizando métricas como la distancia de Hamming. Con base en estos resultados, se determinan las operaciones inversas más adecuadas que deben aplicarse para reconstruir, de forma progresiva y precisa, la imagen original.

Las funciones que la conforman son:

1) ***reverse_mask****:* Esta función revierte el proceso de enmascaramiento contenido en los archivos M.txt en formato RGB. A partir de un arreglo de bytes enmascarados (`bytes_masked`) y la máscara utilizada (`mask`), ambos del mismo tamaño, reconstruye los valores originales del trozo de imagen antes del enmascaramiento.

El algoritmo asume que el enmascaramiento se llevó a cabo mediante la relación:

$$s(k) = ID(k+s) + M(k)$$

donde $s(k)$ es el byte enmascarado, $ID(k+s)$ el pixel desplazado de la imagen original, y $M(k)$ es el valor del pixel de la imagen máscara. La operación inversa se aplica de forma directa como:

$$ID(k+s) = s(k) - M(k)$$

La función devuelve un nuevo arreglo dinámico de tamaño $size \times 3$ (donde el factor 3 corresponde a los canales RGB), que contiene los valores desenmascarados.

Importante: la memoria utilizada para el arreglo resultante se asigna dinámicamente. Es responsabilidad del usuario liberar esta memoria mediante `delete[]` para evitar pérdidas.

2) ***get_reversed_mask****:* Se encarga de recuperar la máscara original utilizada durante el proceso de enmascaramiento de una imagen. Para lograrlo, la función carga un archivo de texto que contiene la semilla y la secuencia de píxeles enmascarados, y luego invoca a la función reverse_mask, que revierte el efecto del enmascaramiento.

El primer paso del proceso consiste en invocar la función loadSeedMasking, la cual extrae tanto la semilla como el número de píxeles afectados, y devuelve un arreglo que representa los valores enmascarados. A partir de esta información, la subrutina aplica la operación inversa del enmascaramiento utilizando la imagen de máscara original y la función reverse_mask.

En esta función se libera la memoria dinámica del arreglo retornado por loadSeedMasking porque al recuperar la sección enmascarada ya no es necesario.

3) ***validate_ro_sh****:* Tiene como propósito evaluar distintas configuraciones de operaciones bit a bit, como desplazamientos y rotaciones, aplicadas sobre la máscara previamente invertida. A través de esta evaluación, busca identificar aquella configuración que produzca la mayor similitud posible entre la máscara transformada y los datos originales de la imagen (`img_data`), en un rango de posibles desplazamientos de 0 a 8 bits.

Durante su ejecución, la función aplica iterativamente una operación dada (`op`), que puede representar, por ejemplo, una rotación a la izquierda o un desplazamiento lógico, a cada byte de los datos desenmascarados. Luego, los resultados obtenidos se comparan con la imagen original utilizando la función auxiliar `validate_rotate_shift_process`, la cual devuelve una métrica de similitud, usualmente basada en la distancia de Hamming.

Si una determinada configuración genera una similitud superior (es decir, una distancia más baja que la previamente registrada en `max_op_sim`), los parámetros de salida `op_code`, `max_op_sim` y `op_n` (número de bits de la operación) se actualizan con los valores correspondientes a esta nueva mejor configuración.

El valor de retorno de la función es el número de bits que proporcionó la mejor coincidencia entre los datos procesados y la imagen original. Esta función resulta esencial dentro del flujo de descifrado de imágenes, ya que permite determinar de forma automática el grado de desplazamiento o rotación aplicado originalmente a la máscara, facilitando así su correcta inversión.

4) ***apply_ops****:* Tiene como objetivo determinar cuál de varias operaciones bit a bit aplicadas a una máscara invertida produce la mayor similitud con una versión ruidosa de una imagen original. Esta función es fundamental dentro del proceso de análisis de imágenes alteradas mediante operaciones bit a bit, ya que permite recuperar información sobre la operación original aplicada durante el proceso de enmascaramiento.

El flujo de trabajo consiste en aplicar secuencialmente distintas operaciones como `XOR`, rotaciones a la derecha (`ROR`) e izquierda (`ROL`), así como desplazamientos lógicos a la derecha (`SHR`) e izquierda (`SHL`), a una máscara invertida. Para cada operación, se calcula su similitud con respecto a la imagen original, comparándola con la imagen ruidosa a través de funciones especializadas como `validate_xor` y `validate_ro_sh`. La métrica empleada es típicamente la distancia de Hamming.

Si se alcanza una coincidencia perfecta (denotada como `MAX_SIMILARITY`), la función retorna de inmediato, registrando el tipo de operación y, si aplica, la cantidad de bits usados. En caso contrario, se selecciona la operación que haya producido la mejor similitud registrada y se devuelve la cantidad de bits que optimizó dicho resultado.

Además, la función informa por consola cuál fue la operación de mayor similitud para propósitos de depuración o análisis, diferenciando entre coincidencia exacta o mejor coincidencia aproximada. Esta capacidad de evaluación automa-

tizada es fundamental para revertir transformaciones aplicadas de manera desconocida a una imagen.

5) ***reverse_operations****:* Se encarga de revertir una transformación aplicada previamente sobre una imagen, restaurando sus datos originales a partir del código de operación y la cantidad de bits utilizados. Esta función modifica los datos de la imagen `img_data` directamente, con base en la operación identificada previamente mediante la función `apply_ops`. Las operaciones inversas están definidas por la Tabla II

Si el código de operación no es reconocido, se imprime un mensaje de advertencia en consola.

6) ***app_img****:* Aplica un proceso de desenmascaramiento y reversión de transformaciones bit a bit sobre una imagen previamente codificada. Realiza `n` iteraciones de reversión sobre la imagen (`I_D.bmp`), que ha sido sometida a operaciones de enmascaramiento y transformaciones binarias. La función utiliza una imagen de referencia (`I_M.bmp`), una máscara (`M.bmp`) y archivos de datos de enmascaramiento secuenciales (`M(n-1).txt`) para deshacer paso a paso las transformaciones aplicadas.

En cada iteración, la función realiza lo siguiente:

- Carga y valida los datos de la máscara y la semilla desde el archivo correspondiente (`M(n-1).txt`).
- Calcula la operación aplicada utilizando la similitud con la imagen original enmascarada.
- Aplica la operación inversa correspondiente a la imagen transformada.

Finalmente, la imagen restaurada se exporta como `I_O.bmp`.

a) Parámetros::

- `n`: Número de transformaciones (y archivos `Mx.txt`) a revertir. Se asume que las transformaciones fueron aplicadas en orden.

b) Dependencias:: Esta función depende de varias funciones auxiliares, como:

- `loadPixels`: para cargar los píxeles de las imágenes.
- `get_reversed_mask`: para obtener la máscara invertida.
- `apply_ops`: para determinar la operación aplicada.
- `reverse_operations`: para revertir la operación sobre la imagen.
- `exportImage`: para exportar la imagen restaurada.

c) Notas:: Si algún archivo no puede abrirse o si las dimensiones de las imágenes son inconsistentes, la función se aborta inmediatamente, liberando la memoria utilizada hasta ese momento.

Las funciones entregadas como suministro inicial para el reto no se incluyen porque son bien conocidas.

C. ***Archivo: main.cpp***

En este documento hay tres funciones simples que validan la entrada del usuario y ejecutan la aplicación principal.

1) ***str_len****:* Esta función cuenta la cantidad de caracteres que tiene el string ingresado por el usuario vía consola.

2) ***get_int16_t****:* Esta función valida que la entrada del usuario esté dentro de los límites del tipo de dato `int16_t`. El proceso de validación se realiza en varios pasos:

- Primero se evalúa si la cantidad de caracteres de la cadena ingresada es menor que `MAX_NUMBER_DIGITS`. Si lo es, se asume que es posible que el número esté dentro del rango permitido para el tipo.
- Luego se utiliza la función `atoi()` para convertir la cadena a un número entero con signo de 32 bits. Esta conversión garantiza que el valor no sobrepase los límites de almacenamiento del tipo `int`, y permite compararlo con los límites de `int16_t`.
- Posteriormente, se verifica si el número convertido está dentro del rango definido por `INT16_MIN` y `INT16_MAX`.
- Si todas las condiciones anteriores se cumplen, se asigna el valor al puntero proporcionado y se retorna `true`, indicando que la conversión y validación fueron exitosas. Si alguna validación falla, se retorna `false` inmediatamente.
- En cuanto a `atoi()`, si detecta una cadena que no es estrictamente numérica, retorna 0. En este caso, no se realiza una validación explícita para asegurar que la cadena contenga exclusivamente números, ya que el valor 0 resultante no es problemático en el contexto de esta validación.

3) Función principal ***main****:* La función `main` representa el punto de entrada del programa. Su objetivo es recibir un único argumento desde la línea de comandos que indica el número de operaciones a revertir sobre una imagen codificada. La función realiza los siguientes pasos:

  - Primero verifica que se haya proporcionado exactamente un argumento adicional al nombre del ejecutable. Si no es así, muestra un mensaje indicando el modo correcto de uso y termina el programa con un código de error.
  - Luego declara una variable de tipo `int16_t` para almacenar el número de operaciones. Esta variable se pasa como referencia a la función `get_int16_t()`, encargada de validar que el argumento sea un número entero en el rango de `int16_t`.
  - Si la conversión no es válida, el programa muestra un mensaje de error indicando que el número ingresado no es válido y termina con código de error.
  - Si el número ingresado es menor que uno, se muestra un mensaje de advertencia adicional y se finaliza la ejecución. Esto impide continuar con una cantidad de operaciones nula o negativa.
  - Finalmente, si todas las validaciones son exitosas, se invoca la función `app_img()` con el número de operaciones proporcionado. Esta función se encarga de aplicar el proceso de reversión de transformaciones sobre la imagen.

IV. Ejecución y resultados

La ejecución del programa se realiza desde una consola en sistemas basados en Linux. Para ello, es necesario ubicarse en el directorio donde se encuentra el archivo ejecutable y ejecutar la siguiente instrucción:

```
./reto_1 [num_operaciones]
```

Donde `[num_operaciones]` representa la cantidad de operaciones que deben revertirse sobre la imagen procesada. Este valor debe ser un número entero positivo dentro del rango del tipo `int16_t`.

Es importante que las imágenes utilizadas por la aplicación (`I_D.bmp`, `I_M.bmp` y `M.bmp`), así como los archivos de máscara (`M0.txt`, `M1.txt`, etc.), se encuentren en el mismo directorio que el ejecutable.

El programa generará como salida una imagen denominada `I_O.bmp`, la cual representa la versión restaurada de la imagen original tras revertir las transformaciones aplicadas.

A. Resultados Caso 1

Al ejecutar el comando correspondiente para las imágenes del caso 1:

```
./reto_1 3
```

se obtuvo la siguiente salida en consola:

Salida del programa

```
La operación #3 fue: XOR
La operación #2 fue: rotación a la derecha de 3 bits
La operación #1 fue: XOR
Imagen BMP modificada guardada como I_O.bmp
```

Estas operaciones corresponden con las indicadas en la guía del reto.

Fig. 2. Imagen restaurada después de aplicar las operaciones inversas.

B. Resultados Caso 2

Para el segundo conjunto de imágenes se ejecutó el siguiente comando:

```
./reto_1 7
```

El programa arrojó la siguiente salida:

Salida del programa

```
La operación #7 fue: XOR
La operación #6 fue: rotación a la derecha de 2 bits
La operación #5 fue: XOR
La operación #4 fue: rotación a la derecha de 5 bits
La operación #3 fue: XOR
La operación #2 fue: rotación a la derecha de 4 bits
La operación #1 fue: XOR
Imagen BMP modificada guardada como I_O.bmp
```

Fig. 3. Imagen restaurada para el caso 2 después de revertir 7 operaciones.

V. Discusión

Las dos imágenes recuperadas son visualmente **idénticas** a las originales. Esto se debe a dos razones fundamentales:

- **Funcionamiento correcto del programa:** Los objetivos de desarrollo planteados en el reto fueron alcanzados exitosamente. El algoritmo implementado se comporta de forma precisa y produce exactamente la salida esperada.
- **Operaciones completamente reversibles:** Durante el proceso de transformación no se utilizaron desplazamientos (shifts) que implicaran pérdida de información. Las únicas operaciones aplicadas fueron del tipo `XOR` y rotaciones (`ROR`), las cuales, como se discutió en la primera entrega, son completamente reversibles. Gracias a esto, las imágenes mantienen intactas sus propiedades visuales: color, textura y forma.

A. Comportamiento del algoritmo con operaciones reversibles y no reversibles

En este contexto, se puede afirmar que el algoritmo funcionará de manera adecuada cuando se utilicen **operaciones completamente reversibles**, tales como `XOR` y rotaciones. Estas operaciones permiten una recuperación exacta de los datos sin pérdida de información, y, por lo tanto, la imagen original puede ser restaurada de manera fiel.

Por otro lado, el comportamiento del algoritmo al emplear **operaciones no reversibles** es impredecible. Esto se debe a que las operaciones de desplazamiento (`SHL`, `SHR`) implican la pérdida de información, especialmente

cuando se aplican varias veces en el proceso de transformación. En particular, la capacidad para recuperar la imagen depende de la cantidad y la etapa en la que los desplazamientos ocurran:

- **Si los desplazamientos ocurren en la primera transformación:** Aunque se perderá parte de la información, es posible recuperar la imagen con cierto nivel de ruido, ya que la información perdida únicamente será relevante en el último paso de la recuperación.
- **Si los desplazamientos ocurren en etapas posteriores:** La recuperación se vuelve más difícil, pues al perder información en las etapas finales, también se pierde la capacidad de rastrear eficazmente las transformaciones previas. Esto hace que la recuperación de la imagen sea menos precisa y más ruidosa, ya que la información original puede haber sido distorsionada de manera irreversible.

Por lo tanto, el uso de operaciones no reversibles en las primeras etapas del procesamiento hace que la recuperación sea más factible, pero aún así dependerá de cuántos datos se hayan perdido y cuándo se hayan perdido.

Con esto en mente, y suponiendo que las operaciones aplicadas son siempre reversibles, el algoritmo se encarga de finalizar cualquier comparación cuando se detecta una operación que recupera exactamente la información. Evaluando inicialmente las operaciones reversibles y dejando para casos atípicos las comparaciones con operaciones no reversibles.

B. Retos afrontados

El principal reto fue en la etapa de análisis donde se debió entender de manera concienzuda el funcionamiento del algoritmo de cifrado y plantear el posible descifrado a partir de los datos de entrada. Una vez comprendido esto el trabajo de codificación fue más fluido. En la etapa de codificación fue interesante la solución del problema asociado a la detección del procedimiento de mayor similitud.

VI. Conclusiones

La solución desarrollada permitió recuperar correctamente las imágenes originales aplicando ingeniería inversa sobre secuencias de transformaciones bit a bit. Las pruebas realizadas confirmaron que el algoritmo es efectivo cuando se utilizan únicamente operaciones reversibles como *XOR* y *rotaciones*.

Se comprobó que la distancia de Hamming es una métrica poderosa para evaluar la similitud entre imágenes, especialmente en presencia de ligeras alteraciones en los datos.

Por otro lado, se identificó que el uso de desplazamientos (*shifts*) puede comprometer la recuperación, dado que implican pérdida de información. En tales casos, la calidad de la imagen reconstruida se ve afectada y depende del momento en que la operación fue aplicada.

En general, el enfoque propuesto demostró ser flexible, eficiente y adecuado para escenarios donde es necesario revertir transformaciones aplicadas a datos visuales.