Estructura del Programa de desencriptado de imagen BMP – Desafío 1 Informática II (UdeA)

Este documento describe la estructura lógica, técnica y funcional del programa desarrollado para resolver el Desafío 1 de la asignatura Informática II, correspondiente al semestre 2025-1 del programa de Ingeniería Electrónica de la Universidad de Antioquia.

El programa tiene como objetivo la reconstrucción de una imagen BMP transformada mediante operaciones binarias a nivel de bits, como XOR, ROTACIÓN y SHIFT, aplicadas en secuencia desconocida. Para lograrlo, se analizan y revierten transformaciones codificadas en archivos auxiliares .txt, empleando técnicas de verificación por máscara y validación progresiva.

La implementación se realiza en C++ utilizando exclusivamente memoria dinámica, sin uso de estructuras ni STL, respetando las restricciones del reto. A lo largo del documento se detallan las decisiones de diseño, fases del algoritmo, manejo de memoria, funciones utilizadas y un análisis profundo de los riesgos asociados a transformaciones destructivas y ambigüedades lógicas.

Trabajo realizado por:  
Juan Paulo Moncada Morales - CC 1059699215  
López Ramírez Emmanuel - CC 1038866909  
Grupo 3

## 

## Estructura del programa

### 1. Variables principales y estructuras de datos

| **Variable / puntero** | **Tipo** | **Descripción** |
| --- | --- | --- |
| bufferA / bufferB | unsigned char\* | Arreglos para almacenar la imagen transformada (ID) y la imagen resultado de la inversa aplicada. |
| actual, siguiente | unsigned char\* | Punteros para manejar los roles de entrada y salida de los buffers. |
| IM | unsigned char\* | Imagen aleatoria usada en operaciones XOR, del mismo tamaño que ID. |
| M | unsigned char\* | Máscara de tamaño menor a la imagen (i x j x 3). Se libera tras desenmascarar los .txt. |
| txt\_data[n] | unsigned int\*\* | Matriz de n apuntadores que guardan los datos RGB desenmascarados de los .txt. |
| semilla\_actual | int | Valor extraído de cada .txt, indica el desplazamiento para aplicar la máscara. |
| operaciones[n-1] | int\* o arreglo de structs | Arreglo que almacena el tipo de operación detectada en cada paso (XOR, ROT, SHIFT) y su parámetro. |
| flag\_shift\_detectado | int | Indicador si ya se detectó una operación SHIFT (para activar lógicas especiales). |
| n\_txt | int | Cantidad total de archivos .txt (etapas del proceso). |
| height, width | int | Dimensiones de la imagen (m x n). |
| t | int | Contador de prueba de operaciones dentro del bucle de desencriptado. |

### 2. Flujo general del programa (Main)

#### Paso 1: Carga de recursos

1. Se inicializan los punteros bufferA, bufferB y se reserva espacio usando new.
2. Se invoca loadPixels() con la ruta del archivo BMP original ("I\_D.bmp") para cargar la imagen transformada.  
   * Esta se guarda en bufferA y se capturan width y height.
3. Se invoca nuevamente loadPixels() para cargar la imagen IM desde su archivo.
4. Se carga la máscara M con loadPixels() y se guardan ancho\_mask, alto\_mask.

#### Paso 2: Carga y desenmascaramiento de archivos .txt

1. Se inicializa txt\_data como una matriz dinámica de n filas (una por cada .txt).  
   * Cada txt\_data[i] apunta a un arreglo de i \* j \* 3 enteros.
2. Se invoca una función personalizada basada en loadSeedMasking() que:  
   * Lee el .txt, extrae la semilla.
   * Resta la máscara a los datos RGB del archivo.
   * Guarda el resultado en txt\_data[i].
   * Devuelve el total de archivos .txt cargados (n).

#### Paso 3: Inicialización del algoritmo de desencriptado

1. Se inicializa el arreglo operaciones[n-1] para guardar las operaciones resueltas.
2. Se crea una variable flag\_shift\_detectado = 0.

#### Paso 4: Bucle principal sobre los .txt

1. Se inicia un for (i = 0; i < n\_txt; ++i) para procesar cada archivo en orden inverso:  
   * Se define semilla\_actual = txt\_data[i][0].
   * Se determinan los roles de actual y siguiente según i % 2.
2. Se ejecuta un ciclo while (flag\_op\_valida == false && t <= 33):

* Se evalúa el valor de t para seleccionar la operación:  
  + t == 1 → XOR
  + 1 < t <= 9 → ROT\_L
  + 9 < t <= 17 → ROT\_R
  + 17 < t <= 25 → SHIFT\_L
  + 25 < t <= 33 → SHIFT\_R
* Se aplica la operación inversa en actual y se guarda en siguiente.
* Se invoca verificar\_enmascaramiento() con siguiente, txt\_data[i] y semilla\_actual.
* Si hay coincidencias:  
  + Si son **totales** → se acepta la operación.
  + Si son **parciales** y es un SHIFT → se marca flag\_shift\_detectado = 1 y se acepta igual.
* Se guarda la operación encontrada en operaciones[i].
* Se hace intercambiar\_buffers(actual, siguiente) para continuar al siguiente paso.

#### Paso 5: Finalización y salida

1. Al terminar el bucle principal:

* Se imprime por consola la secuencia de operaciones detectadas.
* Se exporta la imagen resultante final (reconstruida) con exportImage().
* Se libera la memoria:  
  + delete[] bufferA, bufferB, IM, M, txt\_data[i], etc.

### 3. Consideraciones especiales

* La verificación de cada transformación se hace **solo sobre la región cubierta por la máscara**, no en toda la imagen.
* Si un SHIFT genera coincidencia parcial, se acepta bajo sospecha y se marca para control posterior.
* El uso de intercambiar\_buffers() evita la sobreescritura de memoria durante las transformaciones sucesivas.
* La aplicación del XOR siempre usa IM, que ya está garantizado que cubre toda la imagen (tamaño igual a IO).

### 4. Análisis de transformaciones, ambigüedades y riesgo de corrupción por SHIFT

Este análisis es el más crítico del sistema. A lo largo del proceso se identificaron diversos riesgos algorítmicos relacionados con:

* Las transformaciones destructivas como los desplazamientos (SHIFT)
* El enmascaramiento parcial
* Y la imposibilidad de verificar toda la imagen

#### ▶ El problema fundamental: ver solo una parte de la imagen

Cada archivo .txt representa una muestra parcial de la imagen transformada en cierto punto del proceso. Esto significa que:

* **No se puede verificar la totalidad de la imagen**, solo la región cubierta por la máscara.
* **Un SHIFT puede destruir información en partes que no fueron observadas**, y nunca recuperarse.
* **Operaciones distintas (ej. ROT y XOR) pueden parecer correctas si solo observamos un subconjunto de bytes**.

#### ▶ Ambigüedad entre transformaciones

En muchos casos:

* ROT\_L, ROT\_R y XOR pueden producir el mismo resultado en una sección de bytes.
* Por ejemplo, al aplicar una rotación y un XOR sobre ciertos valores, los bits resultantes coinciden **en la máscara**, aunque no lo hagan en toda la imagen.

Por ello, **la veracidad de una transformación solo puede asegurarse dentro de la ventana que cubre la máscara**.

#### ▶ SHIFT como punto de no retorno

Las operaciones SHIFT son destructivas:

* En cada desplazamiento, se pierden bits y se insertan ceros.
* Esta pérdida **no puede ser revertida** sin información adicional.
* Si **la región perdida no fue registrada en el .txt anterior**, esa información se pierde para siempre.

Esto implica que:

* **Si se aplica un SHIFT antes de que la máscara cubra una región determinada**, ya no es posible verificar su contenido original.
* El algoritmo **puede bloquearse**, porque no hay manera de seguir verificando pasos siguientes.

#### ▶ Verificación encadenada como estrategia

Ante ambigüedades, se propone una verificación en cadena:

* Se aplica una transformación candidata.
* Se verifica con el .txt actual.
* Si hay más de una transformación válida, se siguen aplicando las siguientes y verificando los .txt siguientes.
* Solo si toda la cadena coincide, se acepta esa rama del camino.

#### ▶ Riesgo de caminos falsos

Dado que muchas combinaciones distintas pueden coincidir localmente con un .txt, existe el riesgo de:

* Seguir un camino de transformaciones **que pasa todas las verificaciones locales**,
* Pero que **no corresponde a la secuencia real aplicada**,
* Y eventualmente bloquea el algoritmo si aparece un SHIFT sin contexto previo verificable.

#### ▶ Fragmentación de información por máscaras aleatorias

* Las semillas de los .txt son aleatorias.
* Por tanto, las máscaras pueden cubrir regiones diferentes, **no necesariamente contiguas**.
* Esto agrava el problema: si no hay traslape entre las regiones antes y después de un SHIFT, **la información destruida queda fuera de alcance**.

#### ▶ Consecuencia general

El proceso de desencriptado **no garantiza unicidad ni completitud**. Aunque el algoritmo encuentre una secuencia de transformaciones válida localmente, **no se puede asegurar que sea la secuencia real** ni que la imagen reconstruida sea fiel a la original.

Este es el límite teórico más crítico del sistema, y debe tenerse en cuenta para la interpretación de resultados y validación de salida.

### 5. Consideración abierta: posibles soluciones alternativas

Un posible enfoque que se ha discutido es **explorar modelos de aproximación como el método de mínimos cuadrados**. Aunque no es directamente aplicable a transformaciones lógicas bit a bit, podría:

* Inspirar métricas de mínimo error (por ejemplo, conteo de bits distintos entre dos versiones de imagen)
* Ayudar a estimar regiones faltantes si se cuenta con información redundante

Este enfoque se deja **como una línea de investigación abierta**, sin ser implementado en esta versión del sistema.