

filas, columnas : Dimensiones de la imagen

IMG\_original: Almacena la imagen original (Triple puntero para RGB)

IMG\_transformada: Almacena la imagen durante las transformaciones.

IMG\_aleatoria: Imagen aleatoria para operaciones XOR.

mascara: La mascara a usar para el enmascaramiento de la imagen.

Operaciones Basicas: cargar\_IMG, guardar\_IMG

Transformaciones: aplicarXOR. rotarBITS, aplicar\_mascara

Logica Principal: revertir\_TRANSFOR

Verificación: verificar\_mascara

Este, sería el programa “Ideal” según los requisitos planteados en el PDF.

A el programa planteado, le falta lo siguiente:

**Operaciones de bits:**

Función para aplicar XOR entre imágenes (Io e IM)

Función para rotar bits (hasta 8 bits)

Función para desplazamientos de bits

**Procesamiento de transformaciones:**

Algoritmo para revertir las transformaciones en orden inverso

Implementación del enmascaramiento (suma con la máscara M)

Función para verificar los archivos de enmascaramiento (M1.txt, M2.txt, etc.)

**Lógica principal:**

Mecanismo para determinar el orden correcto de las transformaciones aplicadas

Implementación del proceso de reconstrucción de la imagen original

**Manejo de datos:**

Carga y procesamiento de múltiples archivos de enmascaramiento (no solo M1.txt)

Manejo de la imagen aleatoria (IM) y la máscara (M)

**Validaciones:**

Verificación de consistencia entre operaciones y archivos de rastreo

Mecanismo para probar diferentes combinaciones de operaciones

**Optimización:**

Liberación adecuada de memoria para todas las estructuras dinámicas

Manejo de errores para casos donde las transformaciones no coincidan

## 21/04/2025

Decidí, debido a que no entendí el codigo hecho por el profesor, reescribir el codigo desde cero, basandome en los requisitos.

## Funciones a crear:

unsigned char cargar\_imagen(const char ruta, int& ancho, int& alto);

bool guardar\_imagen(unsigned char datos, int ancho, int alto, const char ruta);

void aplicar\_xor(unsigned char imagen1, unsigned char imagen2, int tamaño);

void rotar\_bits(unsigned char imagen, int tamaño, int bits);

unsigned int cargar\_archivos\_enmascaramiento(const char rutas, int num\_archivos, int semillas, int num\_pixeles);

void liberar\_datos(unsigned int datos, int num\_archivos);

void procesar\_imagen(unsigned char io, unsigned char im, unsigned char m, int ancho, int alto);

## 24/04/25

## PROYECTO COMPLETADO.

Despues de 2 reescrituras de codigo, volví a el archivo planteado por el profesor, el cual modifiqué completamente (Basicamente reescribiendolo), hasta llegar al resultado actual.

AVISO:  
  
La “Imagen reconstruida” no es la imagen real, ya que esta se corrompe, y no pude ver porqué. Sin embargo, el archivo “prueba\_paso” con el numero acompañando a paso siendo menor a n (siendo n el número final de pasos) es el resultado que más se acerca. La verdad creo que pasa, debido a el manejo dado cuando no se determina la transformación para cierto paso.

**Desafíos**

Operaciones no perfectamente reversibles (desplazamientos)

Posibles errores de redondeo en enmascaramiento

Combinaciones explosivas de posibilidades

**Soluciones implementadas**

Comparación con tolerancia para pequeñas diferencias

Algoritmo de búsqueda optimizado para encontrar la transformación correcta

Guardado de pasos intermedios para depuración

## DOCUMENTACIÓN DE EL PROGRAMA

Cómo funciona el programa paso a paso

1. Estructura del programa

El programa usa estas cosas importantes:

- Enum TipoTransformacion: Aquí defino qué tipos de transformaciones se pueden aplicar (XOR, rotaciones, desplazamientos)

- Struct Transformacion: Guarda el tipo de transformación y cuántos bits afecta

- Main: El punto de entrada que procesa todos los casos que encuentre

- ReconstruirImagen: La función principal que hace todo el trabajo de detectar y revertir transformaciones

2. Archivos que necesita

Para funcionar, mi programa necesita estos archivos:

- Imagen transformada: La foto toda rara que hay que arreglar (I\_O.bmp o I\_D.bmp)

- Imagen aleatoria: Una imagen que se usó para las operaciones XOR (I\_M.bmp)

- Archivos de máscara: Varios archivos de texto (M1.txt, M2.txt, etc.) que tienen información para reconstruir la imagen

3. Lo que hace el programa

1. Busca archivos: Primero revisa qué archivos hay disponibles en la carpeta del caso

2. Carga las imágenes: Lee la imagen transformada y la aleatoria

3. Analiza transformaciones: Por cada archivo de máscara, intenta adivinar qué transformación se aplicó

4. Prueba todas las posibilidades: Va probando diferentes combinaciones (XOR, rotaciones de 1-7 bits) hasta encontrar la correcta

5. Reconstruye la imagen: Va aplicando las transformaciones inversas en el orden correcto

6. Guarda el resultado: Al final guarda la imagen reconstruida y también imágenes intermedias de cada paso

4. Funciones principales

- cargarImagen: Lee una imagen BMP y la convierte en un arreglo de bytes

- guardarImagen: Guarda un arreglo de bytes como imagen BMP

- rotarDerecha/Izquierda: Rota los bits de un byte (para deshacer la rotación original)

- desplazarDerecha/Izquierda: Desplaza bits (como << y >>)

- cargarDatosEnmascaramiento: Lee los archivos M1.txt, M2.txt, etc.

- aplicarTransformacion/aplicarTransformacionInversa: Aplica o deshace una transformación

- compararImagenes: Compara dos imágenes para ver si son iguales

Cómo usar el programa

Para usar el programa:

1. Crea una carpeta llamada "casos" en el mismo directorio del ejecutable

2. Dentro de esta carpeta, crea subcarpetas para cada caso (como "caso1", "caso2", etc.)

3. En cada subcarpeta, coloca:

- La imagen transformada (con "\_D" o "\_O" en el nombre)

- La imagen aleatoria (con "\_M" en el nombre)

- Los archivos de máscara (M1.txt, M2.txt, etc.)

4. Ejecuta el programa y listo

El programa va a crear una carpeta "salida" con las imágenes reconstruidas y los pasos intermedios para cada caso.

Implementaciones importantes.

- Detección automática: El programa prueba todas las transformaciones posibles y usa las máscaras para verificar cuál es la correcta

- Reconstrucción inversa: Empiezo desde la última transformación y voy hacia atrás

- Verificación de resultados: En cada paso, verifico si el resultado tiene sentido aplicando la transformación otra vez

- Tolerancia a errores: El comparador de imágenes permite pequeñas diferencias (por si hay errores de redondeo)

Problemas que puede tener

- Si las transformaciones son muy raras o hay más de 6, puede fallar

- A veces la automatización implementada puede equivocarse si hay transformaciones muy parecidas

- Las operaciones de desplazamiento no son totalmente reversibles (se pierden bits)

Hecho por : EMANUEL GUERRA URREA

Hecho por : EMANUEL GUERRA URREA