Documentación Desafío 1   ingeniería inversa, recuperación de imagen.

Autores:

Santiago Londoño Giraldo C.C:1044530108

Julián Andrés Sánchez Ballesteros C.C:1022002752

Fecha: 27 de abril de 2025

Institución:

Universidad de Antioquia

**Introducción**

Este informe tiene como objetivo documentar de manera detallada el análisis e implementación del problema.

 A través de la manipulación de rotaciones, desplazamientos y operaciones XOR, hacemos un proceso de ingeniería inversa en el cual detectamos por medio de un proceso de fuerza bruta (probando cada caso posible) cuál fue la transformación aplicada y posteriormente aplicamos el proceso inverso para regresar a la imagen o proceso anteriores.

**Esquema de tareas en el algoritmo**

1. Carga de imágenes:  
   - I\_D.bmp: Imagen alterada.  
   - M.bmp: Imagen de enmascaramiento.  
   - I\_M.bmp: Imagen mascara.  
  
2. Lectura de información adicional:  
   - Se cargan archivos M\*N.txt que contienen semillas y valores de enmascaramiento.  
  
3. Procesamiento de cada paso:  
   - Para cada paso:  
     - Se carga la semilla y los valores asociados.  
     - Se analiza qué tipo de transformación se aplicó previamente: XOR, rotaciones o desplazamientos.  
     - Se aplica la transformación inversa correspondiente para restaurar la imagen.  
  
4. Exportación final:  
   - La imagen corregida es exportada a un nuevo archivo BMP con nombre I\_O.bmp.

**Documentación detallada de funciones por módulo**

**Archivo: usoimg.cpp**

Código proporcionado por los docentes

* \*\*loadPixels\*\*

Carga una imagen BMP y extrae los datos de píxeles en formato RGB888 sin canal alfa. Retorna un arreglo dinámico de unsigned char que contiene la representación lineal de la imagen.

* \*\*exportImage\*\*

Exporta una imagen BMP usando un arreglo de píxeles en formato RGB. Copia los datos línea a línea y guarda el archivo especificado.

* \*\*loadSeedMasking\*\*

Carga desde un archivo .txt una semilla de imagen y los valores de píxeles resultantes de un proceso de enmascaramiento.

**Archivo: operaciones bits.cpp**

Definimos las diferentes transformaciones posibles para el problema cada una como una función diferente que puede ser útil para cualquier imagen, en el caso puntual de rotación usamos la funcion\_mascara desplazamiento que nos proporciona una máscara nivel de bits dependiendo de la dirección y cantidad de desplazamiento, al cual posteriormente se le hará un **OR**, y para el caso puntual de desplazamiento usamos recuperar\_con\_mascara para obtener las 2^n máscaras posibles para recuperar los bits perdidos.

* \*\*recuperar\_con\_mascara\*\*

recuperar\_con\_mascara, recibe un unsigned char desplazado, unsigned int original, unsigned int masca, unsigned short int n, bool desplazamiento (true (1) para derecha, false (0) para izquierda) y retorna un booleano true en caso de igualdad false en caso contrario, dado un desplazamiento n la función prueba las 2^n maneras de escribir una máscara a nivel de bits, para que al aplicar **OR** entre desplazado + masca se halle el valor deseado original.

**A Grosso modo la función busca recuperar los bits perdidos en un desplazamiento de n valores a la derecha o izquierda**

* \*\*mascara\_desplazamiento\*\*

La funcion\_mascara\_desplazamiento recibe un booleano (desplazamiento) si es true (1) se desplaza a la derecha y si es false (0) a la izquierda y también un N que va a determinar cuál es la cantidad de bits que se está realizando el desplazamiento esto para definir una máscara que tenga N cantidad de 1 y el resto cero

* \*\*funcion\_rotacion\*\*

Esta función recibe un arreglo con valores RGB de la imagen (img) en forma de puntero para poder acceder a todas las posiciones del arreglo,  un arreglo (arr) en forma de puntero para guardar la rotación directamente en el arreglo arr, recibe la cantidad de bits que se rotó (n) ,recibe un booleano desplazamiento(true (1) para derecha, false (0) para izquierda),  int ancho, int alto de la imagen, y hace uso de una funcion\_mascara\_desplazamiento que es usada para guardar los datos y poder hacer la rotación de manera correcta.

**La función realiza una rotación de bits en los valores de cada píxel, ya sea rotando los bits hacia la derecha o hacia la izquierda una cantidad de i veces.**

* \*\*funcion\_xor\*\*

funcion\_xor recibe tres arreglos unsigned char en forma de puntero img\_D, img\_M y arr, para leer la información en cada posición y modificar el valor en el caso de arr, recibe el ancho y el alto de img\_D.

**Todo esto es para aplicar la operación XOR entre img\_D y img\_M y guardar el resultado en arr.**

* \*\*funcion\_desplazamiento\*\*

funcion\_desplazamiento recibe un arreglo con valores RGB de la imagen en forma de puntero, recibe un arreglo unsigned char arr en forma de puntero que va a ser el arreglo en el que se guarde el desplazamiento, recibe un booleano (desplazamiento) si es true (1) se desplaza a la derecha y si es false (0) a la izquierda y también un N que va a determinar qué cantidad de bits es la que se está realizando el desplazamiento recibe alto y ancho de img\_D.

**Esto, para realizar desplazamiento de bits a cada píxel de una imagen hacia la izquierda o derecha.**

**Archivo: funciones.cpp**

Definimos el proceso de ingeniería inversa por medio de la fucion\_de\_cambio, donde se usan de manera secuencial (menor número de iteraciones a mayor) las diferentes transformaciones a nivel de bits, y se usan funciones auxiliares para la aplicación de la máscara(aplicacion\_mascara) para comparar los valores hallados con los esperados(comparar\_txt) a excepción del caso de desplazamiento dada su complejidad más elevada necesita hacer dicho proceso por aparte y por último para seleccionar una porción de la imagen(seccion\_img\_original) para así mejorar la eficiencia a la hora de comparar.

Nota: a diferencia de I\_D.bmp y M.bmp, que se ingresan como parámetros en la fucion\_de\_cambio leemos I\_M.bmp por su uso limitado solo a la operación XOR.

* \*\*aplicacion\_Mascara\*\*

Esta función recibe dos arreglos tipo unsigned char con los datos de una imagen cada uno, **suma los valores de cada una de sus posiciones y los guarda en otro arreglo.**

* \*\*comparar\_txt\*\*

Esta función recibe un arreglo tipo unsigned int con datos de un txt y un arreglo tipo unsigned int con los datos de una imagen con una máscara aplicada y **compara si son iguales.**

* \*\*seccion\_img\_original\*\*

Esta función recibe un arreglo tipo unsigned char con los datos de una imagen y almacena una parte de la imagen, desde una posición semilla pasado como una variable tipo int en un arreglo tipo unsigned char más pequeño.

**Extrae una porción de una imagen comenzando en la posición de semilla dada.**

* \*\*fucion\_de\_cambio\*\*

Esta función recibe un arreglo tipo unsigned char con los datos de una imagen, un arreglo tipo unsigned char con los datos de una máscara, alto y ancho en variables tipo int de la imagen, un arreglo tipo unsigned int con los datos de un txt y un número (semilla) entero que es la posición en la que se aplicó la máscara.

**Identifica la transformación aplicada a una imagen enmascarada (XOR, rotación, desplazamiento) y aplica su inversa.**

**Archivo: main.cpp**

El main controla el flujo completo del programa:  
1. Solicita al usuario la cantidad de pasos a procesar.  
2. Carga imágenes de entrada (I\_D.BMP) y máscaras (M.BMP) y verifica su correcta apertura y lectura.  
3. Para cada paso, aplica la transformación inversa correspondiente usando fucion\_de\_cambio.  
4. Exporta la imagen corregida como I\_O.bmp.

**Problemas afrontados y evolución.**

Al enfrentarnos al desplazamiento nos encontramos con un problema de pérdida de información dada la naturaleza de la transformación a lo cual planteamos diversas soluciones y nos quedamos, al menos para el análisis de la porción de la imagen con la dada por la función recuperar con máscara, básicamente analizamos y encontramos que la única forma de encontrar los valores perdidos era probar los diferentes bits que podían estar en esa posición por medio de una máscara. Pero dado que no hayamos manera de descubrir cuál es exactamente la máscara lo que hicimos fue definir de manera no continua (solo una máscara existente por iteración) las 2^n posibles máscaras para aplicar **OR** y hallar el valor correcto donde n es el número de desplazamiento y 2 son las formas posibles de escribir los bits 1 o 0, con eso solucionamos el problema de la información perdida para la porción de la cual tenemos información o sea los txt proporcionados, pero con el resto de la imagen no hay como comparar, hallamos 2 posibles soluciones:

1. Simplemente realizar la acción inversa a la imagen sin tener nada más en cuenta, pero esto causa problemas para los siguientes pasos dado que ya no es posible encontrar otra transformación.
2. Aplicar la misma transformación al txt que la hallada anteriormente, pero seguía sin solucionar el problema a lo cual recurrimos a también modificar la imagen lo cual nos dio una respuesta parcialmente positiva, se hallan el resto de las transformaciones, pero no son las correctas dada a la intervención de la información a lo cual concluimos que no hallamos forma de volver a la **MISMA** información que se tenía anteriormente si en el proceso hay un desplazamiento dada la información perdida.

La evolución evidenciada conforme al primer análisis presentado fue un aumento en la robusticidad de las funciones al ahora recibir más datos, usar punteros en las funciones para poder modificar la información del main y la implementación de unas cuantas funciones auxiliares, pero en esencia es el mismo enfoque, lo que más evolución presentó fue la función de desplazamiento dado los diversos problemas enfrentados a la hora de su análisis.