**INFORME DE ANALISIS Y DESARROLLO DEL DESAFIO 1**

informática II

Profesor: Aníbal Guerra

María Virginia Castro Herrera

[maría.castroh@udea.edu.co](mailto:maría.castroh@udea.edu.co)

*Departamento de Ingeniería en Telecomunicaciones*

*Universidad de Antioquia*

Medellín-Colombia

Giselle Juliana Gómez Zúñiga

[giselle.gomez1@udea.edu.co](mailto:giselle.gomez1@udea.edu.co)

*Departamento de Ingeniería Electrónica*

*Universidad de Antioquía*

Medellín-Colombia

12 de abril de 2025

1. **Introducción**

Este proyecto aborda el desafío de reconstruir una imagen BMP original a partir de una versión transformada mediante operaciones a nivel de bits (XOR, rotaciones)y enmascaramiento. Con solo la imagen alterada, una máscara y archivos de rastreo como entrada, el objetivo es aplicar ingeniería inversa para revertir las transformaciones en el orden correcto. La solución combina técnicas de programación en C++ (manejo de memoria dinámica, operaciones de bits) con un enfoque lógico-iterativo, validando cada paso contra los datos disponibles.

**II. Análisis del Problema y Consideraciones para la Alternativa de Solución Propuesta**  
El desafío consiste en reconstruir una imagen BMP original a partir de una imagen transformada mediante operaciones a nivel de bits (XOR, rotación, desplazamiento) y enmascaramiento con una máscara aleatoria. Los archivos de rastreo (.txt) contienen información clave para revertir las transformaciones, pero el orden de estas es desconocido.

**Consideraciones:**

* La imagen final (ID) es el resultado de múltiples transformaciones aplicadas en un orden desconocido.
* Cada transformación pudo involucrar operaciones XOR con una imagen aleatoria (IM), rotaciones o desplazamientos de bits.
* El enmascaramiento se aplicó después de cada transformación, usando una máscara M y un desplazamiento s.
* Los archivos .txt contienen las sumas S(k) resultantes del enmascaramiento, lo que permite inferir el estado intermedio de la imagen en cada etapa.

**Alternativa de solución:**

1. **Análisis de archivos**.txt**:** Extraer los desplazamientos s y las sumas S(k) para cada etapa.
2. **Reversión del enmascaramiento:** Usar S(k) y M para recuperar las versiones intermedias de la imagen antes del enmascaramiento.
3. **Identificación de transformaciones:** Probar combinaciones de operaciones (XOR, rotación, desplazamiento) en distintos órdenes hasta revertir la imagen a su estado original.
4. **Validación:** Comparar los resultados intermedios con los archivos de rastreo para confirmar el orden correcto de las operaciones.

**III. ESQUEMA DONDE DESCRIBE LAS TAREAS EN EL DESARROLLO DE LOS ALGORITMOS**

El siguiente esquema muestra la funcionalidad del código y las tareas y técnicas empleadas para su operación

**IV. ALGORITMOS USADOS EN LA IMPLEMENTACION.**

**1. Cargar Datos**

* Para cargar los datos (imágenes BMP) se leen los pixeles RGB y se guardan en arreglos dinámicos utilizando estructuras unsigned char\*(punteros)
* Leer los archivos .txt usando ifstream para abrirlo, leemos la primera línea como int s, ya que corresponde al desplazamiento, y el resto de las líneas en grupos de tres valores (R, G, B) para guardarlos en arreglos dinámicos.

**2. Revertir Enmascaramiento**

* Calcular la Porción Transformada (la parte ID (k + s)) utilizando un ciclo for que recorra de tres en tres los arreglos, desde k=0 hasta el tamaño de la máscara, cuidando los limites y valores menores a 0

**3. Operaciones Inversas**: El código realiza cada operación por separado hasta coincidir con los archivos .txt, para validar cual es la correcta.

* XOR: Recorrer el arreglo de datos encriptados aplicando una operación XOR inversa para recuperar los datos originales, anteriormente modificados durante el enmascaramiento.
* ROTACION CIRCULAR: Deshacer una rotación circular (Izquierda o Derecha), cambiando el signo del desplazamiento, para que la rotación aplicada sea inversa y lo guardamos en un arreglo temporal.
* DESPLAZAMIENTO: Reordenar los datos de pixeles desplazados aplicando el inverso de la operación aplicada del desplazamiento

**4. validación**

* Comparar el resultado del enmascaramiento con el archivo .txt, si coincide es correcta la transformación.

**5. Exportar imagen**

* Convertir el arreglo RGB final a un archivo .bmp, para guardar la imagen desde un arreglo dinámico

**6. Liberar Memoria**

* Liberar los arreglos dinámicos usando delete[], para evitar fugas de memoria.

**V. Problema de desarrollo que encontramos Y soluciones**

Uno de los principales problemas que encontramos fue el manejo de la memoria dinámica. Al trabajar con imágenes representadas como arreglos tridimensionales, Podemos tener dificultades para gestionar correctamente la asignación y liberación de memoria. De pronto El programa puede presentar fugas de memoria o accesos inválidos a posiciones de memoria no asignadas, lo que pude causar comportamientos erráticos o fallos en la ejecución. Otro Problema que es importante fue la implementación correcta de las operaciones a nivel de bits, particularmente las rotaciones. Al principio, no estaba considerando adecuadamente la naturaleza circular de las rotaciones de bits, lo que resultaba en pérdida de información cuando los bits "salían" por un extremo y no "entraban" por el otro. Esto puede afectar la calidad de la imagen reconstruida, ya que los colores no se recuperaban correctamente. A veces Pueden faltar datos o el orden de la información no era el esperado, lo que hacía difícil extraer correctamente los valores necesarios para revertir las transformaciones. Además, el desplazamiento indicado en los archivos no siempre era intuitivo de aplicar, especialmente cuando trabajaba con porciones específicas de la imagen. Uno de los mayores retos fue determinar el orden correcto en que se habían aplicado las transformaciones originales. No poder tener esta información de antemano, tenemos que probar diferentes combinaciones de operaciones inversas, lo que consumir mucho tiempo recursos y de pronto la eficiencia del programa. Finalmente, podemos encontrar con problemas aritméticos al revertir el enmascaramiento. Al restar los valores de la máscara a los valores almacenados en los archivos de rastreo, en ocasiones obtenía resultados negativos que, al ser convertidos a valores de píxel (que deben estar entre 0y 255), causaban distorsiones en la imagen reconstruida.

VI. **La solución y consideraciones**

Manejo de memoria dinámica:

* Verificar asignación de memoria y liberar correctamente al finalizar.
* Usar herramientas para detectar fugas o errores de acceso.

Rotaciones de bits:

* Asegurar que las rotaciones sean circulares para no perder información.
* Diferenciar entre rotación y desplazamiento simple.

Transformaciones inversas:

* Guardar el orden de transformaciones si es posible.
* En caso contrario, probar combinaciones de forma automatizada y evaluar resultados.

Reversión del enmascaramiento:

* Controlar que los valores resultantes estén dentro del rango [0, 255].
* Prevenir distorsiones por valores negativos o fuera de rango.

**VII. Conclusión**

El desarrollo de este proyecto de reconstrucción de imágenes nos permitió enfrentar y superar diversos desafíos complejos en programación, especialmente en el manejo de memoria dinámica, operaciones a nivel de bits, procesamiento de archivos y determinación del orden de transformaciones. Estos problemas nos impulsaron a crear soluciones innovadoras y meticulosas que dieron como resultado un sistema eficiente y confiable. Implementamos una gestión robusta de memoria para evitar fugas y accesos inválidos, operaciones de bits precisas mediante rotaciones circulares bien controladas, un sistema inteligente para interpretar archivos de rastreo incompletos, un motor de inferencia para descubrir la secuencia de transformaciones, y un modelo aritmético que asegura la integridad de los valores de píxel.