**Informe Desafío I - Análisis y Diseño de la Solución**

**1. Análisis del problema**

**a) Definición general del problema**

Se requiere desarrollar un programa en C++ capaz de reconstruir una imagen original a color (IO), la cual fue modificada mediante una serie de transformaciones a nivel de bits. Estas transformaciones incluyen operaciones como XOR, desplazamientos y rotaciones, todas aplicadas sobre los valores RGB de los píxeles. A cada transformación le sigue un proceso de enmascaramiento, en el cual una máscara de color (M) se suma a una región específica de la imagen resultante (con un desplazamiento s), y los resultados se almacenan en archivos .txt. El orden de estas transformaciones es desconocido, por lo tanto, el objetivo principal es identificarlo y revertir las transformaciones para recuperar la imagen original.

**b) ¿Qué es el formato BMP de 24 bits?**

Es un formato de imagen sin compresión en el cual cada píxel se representa mediante tres bytes (uno para cada componente: rojo, verde y azul). Las imágenes BMP de 24 bits son ideales para manipulación directa a nivel de bits.

**c) ¿Cómo se almacenan los datos en el archivo BMP?**

Cada línea de la imagen está alineada a múltiplos de 4 bytes, pero en el procesamiento se utiliza un arreglo lineal continuo de tamaño ancho \* alto \* 3, donde cada tres valores representan un píxel RGB.

**d) ¿Qué son las operaciones a nivel de bits?**

Son operaciones que modifican directamente los bits individuales de los datos. En este caso se utilizarán:

**XOR:** comparación bit a bit entre dos valores.

**Desplazamiento:** mueve bits hacia la izquierda o derecha.

**Rotación:** similar al desplazamiento, pero los bits que salen se reinsertan por el otro lado.

**e) ¿Qué es una máscara de imagen y cómo se usa para enmascarar?**

Es una pequeña imagen de dimensiones menores o iguales a la original. Se utiliza para modificar temporalmente una parte de la imagen, sumando sus valores RGB a una porción de la imagen transformada.

**f) ¿Qué significa enmascarar con desplazamiento s?**

Es tomar un segmento de la imagen desde la posición s y sumarle los valores RGB de la máscara M, generando un conjunto de resultados que se almacenan en un archivo .txt.

**g) ¿Qué implica hacer ingeniería inversa?**

Implica analizar el resultado final de un proceso sin conocer su estructura interna, para intentar deducir los pasos originales. En este caso, se debe determinar el orden de las transformaciones que llevaron de IO a ID.

**2. Descripción general del proceso**

**Entrada:**

Imagen final ID (BMP 24 bits)

Imagen aleatoria IM (BMP 24 bits)

Máscara M (BMP 24 bits)

Archivos .txt que contienen semilla s y resultados RGB del enmascaramiento.

**Meta:** Descubrir qué operaciones bit a bit fueron aplicadas. Determinar el orden de dichas operaciones. Revertir el proceso para reconstruir la imagen original IO.

**3. Diseño de la solución**

La solución se divide en módulos funcionales:

**Carga de imágenes:** leer archivos BMP y extraer datos RGB usando arreglos dinámicos.

**Operaciones bit a bit:**

xorImagenes (img1, img2)

rotarBits (pixels, sentido, nBits)

desplazarBits (pixels, sentido, nBits)

**Enmascaramiento y verificación:** Simular Enmascaramiento (img, M, s) comparar Con Archivo (txt, resultadoSimulado)

**Búsqueda del orden correcto:** Probar combinaciones posibles de transformaciones. Validar con los archivos .txt en cada paso.

**Reconstrucción:** Revertir operaciones en orden inverso. Exportar imagen final IO como BMP.

**4. Evolución de la solución y consideraciones**

**i) Pensamiento inicial**

Inicialmente se pensó en aplicar todas las transformaciones conocidas de forma aleatoria y verificar el resultado con la imagen final, pero esto resultaba poco eficiente y confiable.

**ii) Mejora en el enfoque**

Se comprendió que los archivos .txt actúan como "rastros" que permiten verificar paso a paso si una transformación fue correcta, lo cual permite reducir el número de combinaciones a probar.

**iii) Decisiones técnicas tomadas**

No se usan estructuras ni STL: solo punteros y arreglos dinámicos.

Se divide el programa en funciones para simplificar el desarrollo y prueba de cada parte.

Se usa Qt por compatibilidad con lectura y escritura de BMP.

**iv) Problemas técnicos previstos**

Fugas de memoria si no se libera correctamente.

Desfase de píxeles por errores en cálculo de offset.

Comparación inexacta si hay errores de redondeo o saturación RGB.

**v) Criterios de implementación**

Mantener el código modular.

Documentar cada función.

Validar entradas antes de operar.

Optimizar el orden de pruebas para reducir tiempo.

**5. Descripción de los algoritmos implementados (avance preliminar)**

El código base ya implementa:

Lectura de una imagen BMP (loadPixels)

Exportación de una imagen BMP modificada (exportImage)

Lectura de un archivo de enmascaramiento con semilla (loadSeedMasking)

Estas funciones servirán como base para agregar:

Las transformaciones XOR, rotación y desplazamiento.

La validación contra archivos .txt.

La reconstrucción completa de la imagen original.