ANÁLISIS Y DISEÑO – DESAFÍO 1

Informática II – 2025_1

Desafío 1 – Informática II

En este documento se presenta el análisis realizado al documento con toda la información relacionada con el examen #1 de la asignatura. Se presenta por secciones, cada una encargada de mostrar algunos detalles importantes del análisis realizado y el diseño de la implementación próxima a realizar.

Juan Felipe Orozco Londoño – Davielys Astrid Assia Carrasquero
1000411042 - 1102900188

Descripción general del problema

El desafío consiste en desarrollar una aplicación en C++ utilizando Qt que permita procesar imágenes BMP mediante máscaras de transformación. El sistema debe:

- Cargar una imagen BMP (sin compresión) y acceder a sus datos RGB (24 bits, 3 bytes por píxel).
- Aplicar transformaciones artificiales a la imagen a nivel de bits (operaciones de XOR, desplazamiento y rotación de bits, con un máximo de 8 bits de movimiento).
- Leer archivos de enmascaramiento generados en cada etapa (archivos .txt que contienen, en la primera línea, un desplazamiento y, a partir de la segunda línea, tripletas RGB resultantes de sumar una porción de la imagen transformada con una máscara).
- Finalmente, mediante ingeniería inversa, identificar qué operaciones a nivel de bits fueron realizadas (y en qué orden) para, eventualmente, reconstruir la imagen original.

Las transformaciones se aplican sobre la imagen de forma secuencial (por ejemplo:

1. Aplicar XOR entre la imagen original y una imagen de distorsión, generando una imagen intermedia P1.bmp y el archivo M0.txt,
2. Rotar 3 bits a la derecha P1.bmp para obtener P2.bmp y generar M1.txt,
3. Aplicar nuevamente XOR entre P2.bmp y la imagen de distorsión para obtener la imagen final I_d.bmp, sin enmascaramiento en el último paso).

Revisión del código base proporcionado

El código base en C++ con Qt que realiza lo siguiente:

- **Carga de imágenes BMP:** Función `loadPixels()` que utiliza QImage para leer y convertir una BMP en un arreglo dinámico de bytes (unsigned char*), sin padding.
- **Exportación de imágenes:** Función `exportImage()` que copia los datos desde un buffer a un objeto QImage para guardar la imagen resultante.
- **Lectura de archivos de enmascaramiento:** Función `loadSeedMasking()` para cargar desde un archivo .txt la semilla (offset) y las tripletas RGB resultantes del proceso de enmascaramiento.

Análisis del problema y alternativa de solución:

El reto implica dos grandes bloques:

- **Transformaciones y enmascaramiento:** Se aplican operaciones a nivel de bits a una imagen (XOR, desplazamiento y rotación de bits) y, tras algunas de estas operaciones, se aplica un enmascaramiento que consiste en sumar (pixel a pixel) los valores RGB de la imagen transformada y una máscara de color.

- **Reconstrucción inversa:** Mediante los archivos de rastreo (M*.txt), se deberá analizar y deducir el orden y tipo de operaciones aplicadas para, posteriormente, aplicar las operaciones inversas y así reconstruir la imagen original.

La solución planteada se basa en el diseño modular de funciones libres en C++ que, mediante el uso exclusivo de punteros, arreglos y memoria dinámica, se encarguen de cada tarea:

- Lectura y procesamiento de imágenes BMP.
- Lectura y almacenamiento de las máscaras.
- Aplicación de transformaciones (XOR, desplazamiento, rotación) a nivel de bits.
- Verificación del enmascaramiento comparando datos obtenidos con los archivos de rastreo.
- Implementación del algoritmo inverso para identificar el orden de operaciones y revertirlas.

Tareas:

Las tareas fundamentales para el desarrollo del sistema son:

- **Lectura de imagen BMP**: Crear una función que cargue una imagen BMP y almacene sus datos RGB en un arreglo dinámico utilizando punteros.
- **Lectura del archivo de máscara**: Implementar una función para leer el archivo de enmascaramiento (.txt) y almacenar el desplazamiento junto con las tripletas RGB en arreglos dinámicos.
- **Aplicación de máscaras**: Desarrollar un algoritmo que aplique, a nivel de píxel (y a nivel de bit), la transformación definida por las máscaras sobre la imagen cargada.
- **Implementación de operaciones a nivel de bits**: Crear funciones libres para:
  - Realizar la operación XOR entre dos bytes.
  - Desplazar bits (shift a la izquierda o derecha, hasta 8 posiciones).
  - Rotar bits (rotación circular de un byte, en ambas direcciones, hasta 8 posiciones).
- **Verificación del enmascaramiento**: Implementar un algoritmo que compare la salida de la aplicación de la máscara (sumas de píxeles transformados con la máscara) con los valores registrados en los archivos de rastreo.
- **Algoritmo de ingeniería inversa**: Diseñar un algoritmo que, dada la información de las transformaciones (archivos de rastreo M*.txt), identifique qué operaciones se aplicaron y en qué orden, para poder invertirlas y reconstruir la imagen original.
- **Integración y validación**: Integrar todos los módulos, realizar pruebas con los casos entregados y ajustar la solución.

Algoritmos considerados

Funciones de Operaciones a Nivel de Bits:

- XOR:
  Implementar una función que aplique XOR entre dos valores de 8 bits. Dado que el XOR es reversible, se usará tanto para encriptar como para desencriptar.
- Desplazamiento (Shift):
  Función para desplazar bits hacia la izquierda o derecha, hasta 8 posiciones. Se debe asegurar que los bits que se pierdan se manejen correctamente.
- Rotación de Bits: Función que realice rotaciones circulares, de modo que los bits que salen por un extremo se reintroduzcan por el otro. Permitido hasta 8 posiciones.

Experimentación y análisis:

- Se realizarán pruebas sobre bloques de datos (píxeles) para medir el efecto de cada operación, su reversibilidad y su impacto en la imagen.
- Se documentarán los resultados experimentales y se analizará la utilidad de cada operación en el contexto de encriptación básica.

Verificación del enmascaramiento:

- Se desarrollará un algoritmo que, a partir de la imagen transformada y la máscara, y usando el desplazamiento s extraído del archivo de enmascaramiento, verifique que:

$$S(k) = Id(k+s) + M(k)S(k) \;=\; I\_d(k+s) \;+\; M(k)S(k) \\ = Id(k+s) + M(k)$$

  Para cada píxel k (o tripleta RGB), permitiendo validar que el enmascaramiento se realizó correctamente.

Algoritmo inverso de identificación y reconstrucción:
A partir de los archivos de rastreo (M*.txt) y la imagen final transformada (I_D.bmp), desarrollar un algoritmo que determine el tipo de operaciones a nivel de bits que se aplicaron y su orden.

Estrategia Propuesta:

- Análisis de Patrones: Comparar los resultados en los archivos de rastreo con los valores esperados en función de las operaciones conocidas (XOR, shift, rotación).
- Búsqueda del Orden de Operaciones: Implementar un algoritmo iterativo o heurístico que evalúe distintas combinaciones de operaciones (respetando la restricción de máximo 8 bits de movimiento) para identificar la secuencia que, al invertirse, recupere los datos originales.
- Aplicación de Operaciones Inversas: Una vez identificado el orden, aplicar la versión inversa de cada función (por ejemplo, rotación en sentido contrario, desplazamiento inverso, aplicar XOR nuevamente) para intentar reconstruir la imagen original.
- Uso Obligatorio de Memoria Dinámica y Punteros: Todas las operaciones se realizarán sobre arreglos dinámicos (por ejemplo, para almacenar la imagen en

memoria, la máscara y los resultados intermedios), usando punteros para acceder y modificar los datos a nivel de bit.