Desafío1

Edwin Piedrahita Piedrahita

Prof. Augusto Enrique Salazar Jiménez

Informática 2

1. Análisis del problema

Se tiene como objetivo la reconstrucción de una imagen en formato BMP de 24 bits, esta imagen fue sometida a varias transformaciones a nivel de bits, de las cuales se desconoce el orden exacto en las que se realizaron las transformaciones. Esto implica un sistema de ingeniería inversa, lo que lleva a deducir un algoritmo algo complejo basado en rastros (archivos) y la implementación de técnicas de manipulación de bits.

La tarea consiste en utilizar la imagen transformada IM, la máscara M y los archivos para poder deducir el orden y el tipo de transformación que se aplicó, y de esta forma poder recuperar la imagen original.

2. Desarrollo de la solución

a. Realizar un esquema o diagrama que me permita ver una posible solución.

b. Desarrollar funciones que realicen las operaciones a nivel de bits, como lo son XOR, desplazamientos y rotaciones.

c. Crear y validar algoritmos para el enmascaramiento y verificar usando los archivos.

d. Diseñar un método que permita deducir el orden de las transformaciones a partir de las pistas incompletas.

e. Documentar el código respectivamente.

3. Esquema de las tareas

1. Proceso Principal de Recuperación (función main)

Cargar imágenes necesarias para el proceso:
I_D.bmp: Imagen distorsionada final
I_M.bmp: Imagen de distorsión utilizada para aplicar XOR
M.bmp: Máscara para validar las operaciones
Revertir operaciones de distorsión en orden inverso:
Revertir último XOR (I_D, I_M)

Validar resultado con archivo M2.txt
Revertir rotación de 3 bits a la derecha
Validar resultado con archivo M1.txt
Revertir primer XOR (I_R, I_M)
Validar resultado con archivo M0.txt
Exportar imagen recuperada como "Io_recuperada.bmp"
Comparar con imagen original "I_O.bmp" para verificar éxito

2. Operaciones de Manipulación de Bits

Operación XOR:
Función AplicarXOR()
Realiza XOR bit a bit entre la imagen procesada y la imagen de distorsión
Operación reversible utilizada para cifrar y descifrar
Operaciones de Rotación:
Función RotarBitsIzquierda(): Rota bits hacia la izquierda para revertir rotación derecha
Función RotarBitsDerecha(): Rota bits hacia la derecha (aunque no se usa en el proceso principal)
Operaciones de Desplazamiento:
Funciones DesplazarBitsIzquierda() y DesplazarBitsDerecha()
Implementadas, pero no utilizadas en el proceso principal

3. Funciones Auxiliares de Manejo de Archivos e Imágenes

Carga de Imágenes:
Función CargarPixels(): Convierte imágenes BMP a arreglos de bytes RGB
Validación:
Función CargarSemilla(): Lee archivos de texto con información de validación
Extrae semilla (posición inicial) y valores RGB para verificación
Exportación:
Función exportImage(): Guarda los datos de píxeles como imagen BMP
Verificación de Resultados:
Función CompararImagenes(): Verifica si dos imágenes son idénticas

4. Flujo de Verificación en Cada Etapa

Después de cada operación de reversión, se valida:
Usando archivos de semilla (M0.txt, M1.txt, M2.txt)
Aplicando fórmula: Suma = Pixel_recuperado + Pixel_máscara
Comparando resultado con valores esperados en archivos de semilla
Verificación final comparando la imagen recuperada con la original

4. Algoritmos Implementados en el Sistema de Recuperación de Imágenes

Algoritmo de Operación XOR

Se implementa un algoritmo que aplica la operación XOR bit a bit entre una imagen y una máscara de distorsión. Esta técnica es fundamental en el proceso tanto de cifrado como de descifrado debido a su naturaleza reversible: al aplicar XOR dos veces con la misma máscara, se recupera la información original. El sistema utiliza esta propiedad para revertir dos operaciones XOR que fueron aplicadas durante el proceso de distorsión.

Algoritmo de Rotación de Bits

El sistema incluye algoritmos para realizar rotaciones circulares de bits, tanto a la izquierda como a la derecha. La rotación de bits mueve cada bit en un byte a una nueva posición según la cantidad especificada, y los bits que "salen" de un extremo "entran" por el otro. El proceso de recuperación utiliza la rotación a la izquierda para revertir una rotación a la derecha de 3 bits que se aplicó durante la distorsión.

Algoritmo de Desplazamiento de Bits

Aunque no se utilizan en el flujo principal de ejecución, el código implementa algoritmos de desplazamiento de bits tanto a la izquierda como a la derecha. A diferencia de la rotación, el desplazamiento introduce ceros en los bits vacíos tras el movimiento, lo que lo convierte en una operación no reversible directamente.

Algoritmo de Validación por Enmascaramiento

Se implementa un algoritmo de validación que utiliza una técnica de enmascaramiento para verificar la correcta aplicación de cada operación de reversión. Este algoritmo:

Toma una semilla que indica posición de inicio

Suma píxeles específicos de la imagen procesada con píxeles de una máscara

Compara estos resultados con valores predefinidos almacenados en archivos externos

Algoritmo de Comparación de Imágenes

El sistema incluye un algoritmo que compara dos imágenes píxel por píxel para determinar si son idénticas. Este algoritmo es crucial para la verificación final que confirma si la imagen recuperada coincide exactamente con la imagen original antes de la distorsión.

Algoritmos de Manipulación de Archivos de Imagen

Se implementan algoritmos específicos para la carga y exportación de imágenes en formato BMP, manipulando directamente los bytes que representan cada píxel RGB. Estos algoritmos gestionan la conversión entre estructuras de datos en memoria y archivos de imagen.

5. Problemas de desarrollo

1. Hallar el orden correcto en el que se aplicaron las operaciones a nivel de bits.

2. Problemas al usar la formula, ya que intentaba aplicar la formula como una resta en vez de una suma.

3. Recuperar la imagen de forma que coincidiera con la original.

4. Identificar que valores se debían usar para aplicar las operaciones a nivel de bits.

6. Evolución de la Solución

La solución implementada para la recuperación de imágenes distorsionadas ha evolucionado a través de un enfoque metódico:

1. Análisis del Proceso de Distorsión

El desarrollo comenzó con un análisis inverso del proceso de distorsión aplicado a la imagen original. Se identificó que la imagen había sido sometida a una secuencia específica de operaciones:

Un XOR inicial con una imagen de distorsión

Una rotación de bits a la derecha

Un XOR final con la misma imagen de distorsión

2. Diseño del Proceso de Recuperación

Una vez comprendido el proceso de distorsión, se diseñó un flujo inverso de operaciones para recuperar la imagen original:

Aplicar XOR para revertir la última modificación

Rotar bits a la izquierda para contrarrestar la rotación previa

Aplicar nuevamente XOR para recuperar la imagen original

3. Implementación de Mecanismos de Verificación

La solución evolucionó para incluir puntos de verificación después de cada operación, utilizando archivos de semilla (M0.txt, M1.txt, M2.txt) que permitieran validar la correcta aplicación de cada algoritmo en puntos específicos de la imagen.

4. Refinamiento del Sistema de Validación

Se desarrolló un esquema de verificación final que compara la imagen recuperada con la original, asegurando que el proceso completo funcione correctamente sin alteración de los datos visuales.

Consideraciones para la Implementación

Gestión de Memoria:

Implementación cuidadosa de la asignación y liberación de memoria dinámica para evitar fugas, especialmente al manejar grandes volúmenes de datos de imagen.

Verificación de la correcta carga de cada imagen antes de proceder con operaciones, liberando recursos en caso de error.

Precisión de las Operaciones a Nivel de Bits:

Las operaciones de rotación y XOR deben ser implementadas con exactitud para garantizar la reversibilidad perfecta.

Manejo adecuado del módulo 8 en las rotaciones para asegurar que el número de bits no exceda un byte.

Compatibilidad de Formatos:

Utilización del formato RGB888 para asegurar consistencia en la manipulación de datos.

Consideración del padding en archivos BMP para evitar distorsiones al procesar líneas de píxeles.

Modularidad del Código:

Separación clara de funcionalidades en métodos específicos para permitir pruebas aisladas y mantenimiento.

Diseño que facilita la adición de nuevas operaciones de manipulación de bits sin afectar el flujo principal.

Robustez ante Errores:

Implementación de verificaciones en cada paso del proceso para detectar desviaciones tempranamente.

Mecanismos de salida limpia cuando se detectan errores, liberando recursos adecuadamente.