Informe Desafío 1
Andrés Felipe Sepúlveda Rivillas
71.363.392

a) Análisis del problema y consideraciones para la alternativa de solución propuesta.

El problema propuesto para el Desafío 1 del curso de Informática II, se basa en una imagen BMP (Bitmap), la cual es un formato gráfico para almacenar imágenes digitales, la cual permite un acceso directo y rápido a cada pixel. En este caso, cada pixel está representado mediante valores de color en forma de tripletas RGB (red, green, blue) con un tamaño de 3 bytes.

Por otro lado, se tendrá una imagen distorsionada a través de transformaciones a nivel de bits, que pueden incluir desplazamientos, rotaciones y operaciones XOR con una imagen de distorsión aleatoria, todas estás transformaciones se aplican en un orden que se desconoce para el programador. Tras cada una de las etapas de transformación, se emplea una técnica de enmascaramiento que consiste en suma una porción de la imagen distorsionada con una imagen de color, esta porción de la imagen se determina a partir de una posición inicial, denominada semilla, la cual está contenida en los archivos de rastreo, mediante la suma:

$$S(k) = ID(k + s) + M(k) \text{ para } 0 \leq k < i \times j \times 3$$

Los archivos .TXT, contienen además de la semilla, información resultante del enmascaramiento realizado, determinado por conjuntos de valores enteros que representan la suma de los canales RGB, píxel a píxel.

El objetivo del desafío, es aplicar Ingeniería inversa para recuperar la imagen original que fue distorsionada mediante las transformaciones y las máscaras, usando los temas vistos en el curso, como son estructuras de control, operaciones a nivel de bits, punteros, arreglos dinámicos y funciones en C++.

Para abordar la solución al problema planteado, se tienen a disposición tres códigos elaborados en C++ con los cuales se puede acceder a los valores de los píxeles de una imagen BMP, abrir los archivos que contienen el enmascaramiento y exportar la información contenida en los arregles dinámicos como imágenes. Mediante estoy códigos y las funciones adicionales que se desarrollen, se podrá cumplir con el objetivo propuestos en el desafío, como los son:

- Función para revertir el enmascaramiento.
- Función para realizar operaciones a nivel de bit.
- Función para realizar la operación XOR con la imagen, en caso de ser necesario.

b) Esquema donde describa las tareas que usted definió en el desarrollo de los algoritmos.

Para el desarrollo del algoritmo se definen las siguientes, teniendo en cuenta los requerimientos establecidos y las herramientas proporcionadas:

1. Cargar las imágenes y los datos iniciales:

    i. Cargar la imagen transformada I_D.
    ii. Cargar la imagen que sirve de máscara M.
    iii. Cargar la imagen para las operaciones XOR I_M.
    iv. Cargar los archivos de rastreo.

2. Revertir el enmascaramiento y generar las permutaciones necesarias para determinar qué tipo de transformaciones se realizaron a la imagen:

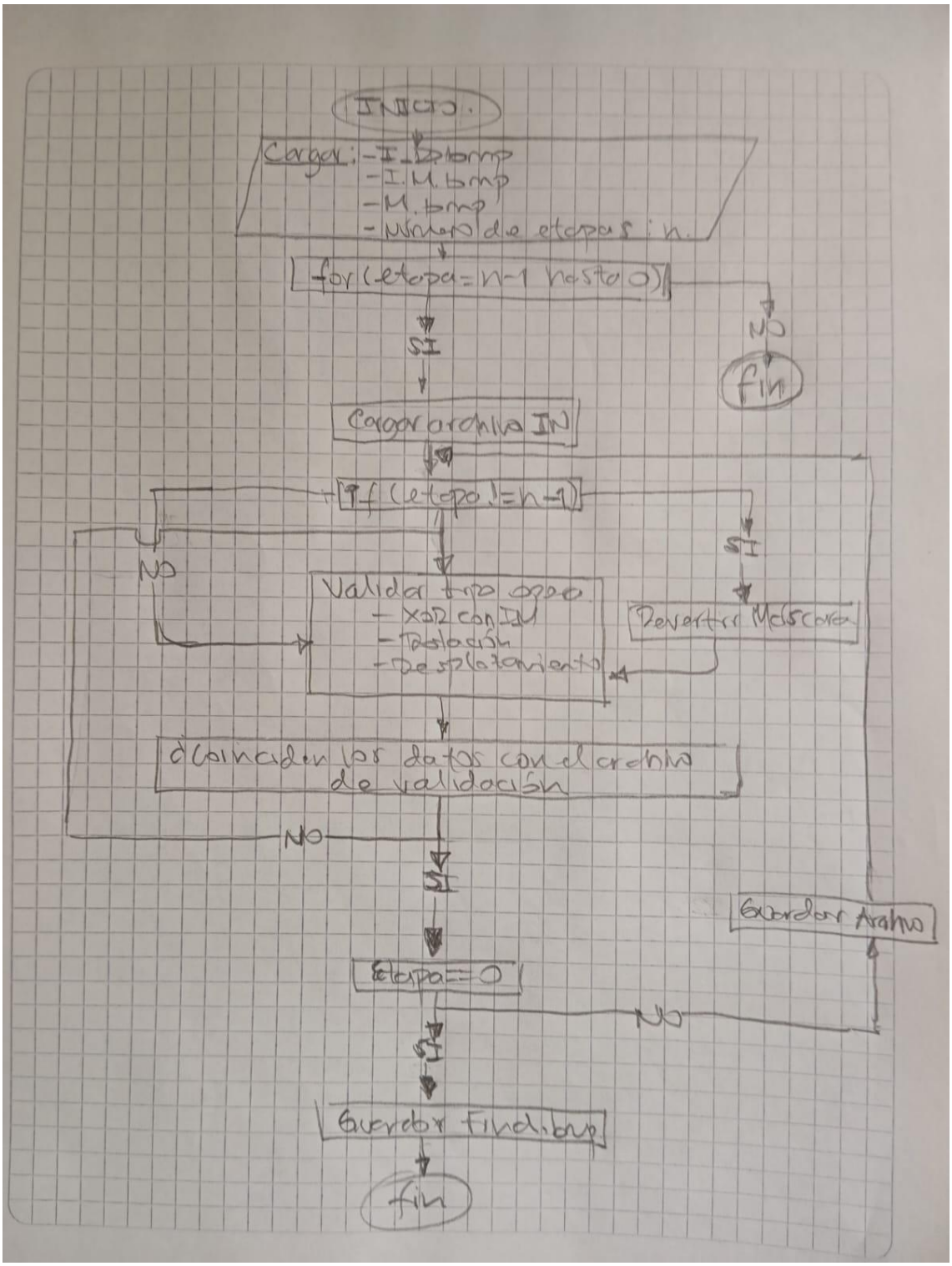
    i. Crear la función para revertir el enmascaramiento.
    ii. Generar la función que permita aplicar las permutaciones necesarias para determinar las transformaciones, teniendo en cuenta los tipos de operaciones a nivel de bits que se pueden usar.

3. Realizar las pruebas y visualizar las imágenes de salida:

    i. Crear una función para enmascarar que permita validar las operaciones realizadas.
    ii. Validar que las operaciones de permutación realizadas resulten correctas, mediante la comparación con los archivos .TXT de entrada.
    iii. Exportar la imagen generada luego de revertir el enmascaramiento y realizar las permutaciones, para validar visualmente el resultado.

c) Algoritmos implementados.

Según las consideraciones hechas en el ítem anterior, puedo plantear el siguiente esquema básico para implementar la solución del problema:



d) Problemas de desarrollo que afrontó.

1. Gestión de la memoria: debido al uso de los arreglos dinámicos y los punteros, se debe tener especial cuidado al momento de hacer la reserva de la memoria y especialmente en la correcta liberación de esta.

2. Se debe verificar el tamaño de las imágenes para garantizar que los tamaños de la imagen distorsionada y la imagen máscara para las operaciones XOR coincida, por otro lado, se debe garantizar que el tamaño de la imagen utilizada en el proceso de enmascaramiento sea igual o menor a la imagen distorsionada.

3. El tema de la validación al momento de realizar las operaciones de permutación se complica un poco, ya que de manera visual no se puede determinar si el proceso realizado es correcto o se encamina a la solución.

4. Se debe controlar de manera efectiva la aparición de valores negativos al momento de realizar las operaciones que revierten el enmascaramiento.

5. Para validar que las operaciones a nivel de bits realizadas fueran correctas, debí implementar una función para enmascarar de nuevo la imagen y poder validar con la imagen generada en la etapa anterior, este proceso me generó cierta dificultad, ya que no conocía la lógica para implementar la salida del archivo .txt, por lo tanto, use la función proporcionada que permitía entrar los datos del .txt y de esa manera obtuve la función que necesitaba y pude validar las operaciones en cada etapa del proceso.

6. Para estructurar todo el código de manera que pudiera tener una secuencia correcta y permitiera automatizar los procesos al máximo nivel posible, tuve que realizar muchos ensayos hasta llegar al funcionamiento deseado, solo en los ensayos estuve dos días para poder que todo quedara de la mejor manera cumpliendo el objetivo del desafío.

7. Dentro de la etapa de entrada quise implementar una carga de los archivos de manera totalmente automática, sin embargo, no pude lograrlo con el conocimiento que tengo hasta el momento, consulté algunas formas en la red, pero no las implementé ya que no podría explicar de manera coherente el código implementa lo que estaría en contravía con el objetivo de estudio.

e) Evolución de la solución y consideraciones para tener en cuenta en la implementación.

Desde el momento en que pude abordar de manera completa el enunciado del desafío, me pude dar cuenta de que debería realizar un análisis previo muy detallado antes de implementar cualquier función o porción del código, el enunciado en sí, planteo un gran reto ya que una lectura de corrido sin detenerse a analizar llevaría a un entendimiento inadecuado del problema que se plantea.

Partiendo de esa hipótesis, luego de realizar un análisis detallado de cada etapa del problema, establecí una ruta de acción basada en las tareas que propongo en el apartado b., no sin antes hacer un análisis detallado de las tres funciones que nos proporcionaron, la cuales son vitales dentro del funcionamiento completo que implementé.

Partiendo de las entradas que el código debería recibir, me enfoqué en analizar las imágenes que proporcionaron para el caso 1 y en los archivos de enmascaramiento. Luego de esto, implementé las funciones que me permitieran el manejo de las operaciones a nivel de bits y la operación XOR con la imagen para poder probar el set de prueba del caso1.

Para el caso 1, no tuve mayores inconvenientes, al devolver las transformaciones realizadas pude obtener de manera inmediata la imagen original. De manera que proseguí con el set de prueba para el caso 2. Allí debí analizar una vez más las condiciones y sugerencias que se establecían en el enunciado, ya que por un momento perdí el rumbo y me enfoqué más en tratar de llegar a la operación sin detenerme a pensar en cómo validar si esa operación que estaba realizando estaba correcta.

En este punto, pensé en implementar dos funciones que me permitieran elaborar el enmascaramiento y otra que me permitiera revertir el enmascaramiento con las cuales podría probar si las operaciones que iba realizando estaban correctas y poder limpiar la imagen de la máscara antes de elaborar cualquier operación. Pero una vez más me vi truncado debido a que al momento de hacer la validación entre el archivo .txt que me generaba la función de enmascaramiento con los datos de los archivos .txt proporcionados, obtenía diferencias significativas que no podía conciliar.

En ese momento, luego de realizar otras pruebas adicionales con las operaciones, me di cuenta que estaba omitiendo una parte muy importante, que al final fue la que me dio la luz para poder encontrar una solución, esa parte importante radicaba en que estaba realizando operaciones empezando por la XOR y si esa no daba seguía con las rotaciones a la izquierda y a la derecha empezando por 1 bit hasta los 8 bits y por último ensayaba el desplazamiento, sin embargo, no tenía en cuenta que debía devolver esas operaciones de manera que la imagen con la que estaba comparando no se corrompiera, al implementar ese bloque dentro del código comencé a tener resultados.

En la primera interacción con el código que tenía establecido, pude encontrar las operaciones que se habían realizado en cada etapa del caso 2, ingresando cada archivo .txt y cada imagen

de manera manual, de esta manera encontré la solución y pude obtener la imagen original, lo cual me dio la motivación suficiente para continuar buscando una solución más automática y que permitiera el manejo de otras imágenes distorsionadas.

Para solucionar el inconveniente, implementé mediante un ciclo for, una variable llamada etapa que me permitía controlar cuantas veces debía realizar el proceso de revertir la máscara, operar a nivel de bits y enmascarar de nuevo para verificar con el archivo .txt. Para el ingreso y salida de imágenes y el ingreso de los archivos .txt implemente un arreglo con el número determinado de etapas posibles según el set de prueba que me proporcionen. Como lo mencioné en el ítem anterior de los problemas afrontados, preferí no implementar una entrada más automática para no entrar en conflicto con los temas vistos hasta ahora y evitar algún inconveniente a la hora de sustentar el código.

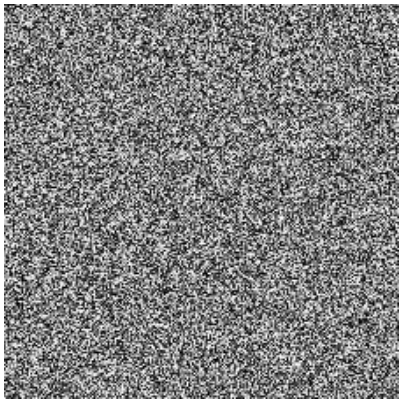
Teniendo en cuenta lo dicho, implementé el código de la manera más automática posible, que pudiera exportar la imagen original si me proporcionan una imagen distorsionada, una imagen para realizar operaciones XOR, una imagen máscara y los archivos .txt. Para el caso 2 específicamente, elaboré esta ruta de entrada, salida y resultados en cada etapa, para verificar la implementación:

VARIABLE ETAPA	ETAPA	IMAGEN ENTRADA	REVERTIR MÁSCARA	ARCHIVO DE SALIDA	ENMASCARAR	RESULTADO
6	7	I_D.BMP	N/A	ETAPA6.BMP	M6.TXT	XOR
5	6	ETAPA6.BMP	M6.TXT	ETAPA5.BMP	M5.TXT	RD2
4	5	ETAPA5.BMP	M5.TXT	ETAPA4.BMP	M4.TXT	XOR
3	4	ETAPA4.BMP	M4.TXT	ETAPA3.BMP	M3.TXT	RD5
2	3	ETAPA3.BMP	M3.TXT	ETAPA2.BMP	M2.TXT	XOR
1	2	ETAPA2.BMP	M2.TXT	ETAPA1.BMP	M1.TXT	RD4
0	1	ETAPA1.BMP	M1.TXT	FINAL.BMP	M0.TXT	XOR

Anexo: experimentos realizados con las operaciones a nivel de bits.

Para analizar el efecto de las transformaciones sobre la integridad de los datos y su utilidad en un escenario de encriptación básica de información, desarrollé diversas pruebas con las funciones que me permiten realizar operaciones a nivel de bits, los resultados fueron los siguientes:

- ✓ Operación XOR con la imagen:



- ✓ Operación rotación 4 bits a la derecha:

- ✓ Operación rotación 2 bits a la izquierda:

- ✓ Operación desplazamiento a la derecha de 1 bit:

De manera general, el desplazamiento de bits es básicamente una operación de corrimiento de determinados bits a la derecha o izquierda, lo que equivale a que el dígito seleccionado será desplazado la cantidad de bits y llenará con ceros los espacios que se dejen vacíos. Este tipo de operación no es muy recomendada en el ámbito de la encriptación, ya que puede generar pérdidas de información al momento de rellenar con ceros las posiciones que se mueven.

Por otro lado, la operación de rotación de bits es como la combinación de dos desplazamientos, de manera tal, que al mover determinado numero de bits en una dirección, los espacios vacíos no se llenan con ceros, sino que se llenan con el resto de los bits que saldrían por el otro lado, esta operación es reversible no genera pérdida de información, esto la hace muy confiable al momento de encriptar datos.

La operación XOR, es la más usada en la encriptación básica, ya que permite generar un valor opuesto al operar con otra grupo de datos, esto es muy deseado porque permite codificar una imagen, si se quisiera enviar codificada a un receptor, y su reversión se hace usando la misma operación, lo que la hace fácil de usar teniendo la imagen con la cual se generó la máscara.