**PROCESO DE ANÁLISIS Y DISEÑO DE LA SOLUCIÓN PLANTEADA PARA EL DESAFÍO 1**

*Adriana Camila Erazo Mora*

*María Angélica Osorio Rincón*

1. **Contexto general del desafío:**

El problema consiste en recuperar una imagen original IO de dimensiones m filas por n columnas, que ha sido sometida a una serie de transformaciones binarias y enmascaramientos parciales, a partir de una imagen máscara IM de dimensiones m filas y n columnas y una máscara de dimensiones i ≤ m filas y j ≤ n columnas, generando al final de cada transformación una imagen distorsionada ID y un archivo de texto correspondiente al enmascaramiento.

* Cada transformación se compone de una operación binaria entre una imagen (IO o alguna imagen intermedia ID) y la imagen máscara IM.
* El enmascaramiento parcial consiste en sumar una máscara M a una porción de la imagen transformada ID, desplazada mediante una **semilla** s. Esta suma se registra en un archivo .txt.

El objetivo final es **revertir todos los pasos** para recuperar la imagen original IO, usando únicamente la información proporcionada: los archivos de texto plano, la imagen máscara IM, la máscara M y la imagen transformada (imagen que resulta de todas las transformaciones y enmascaramientos aplicados).

1. **Comprensión y separación de procesos:**

Es fundamental distinguir entre los dos procesos que componen cada transformación:

1. *Transformación binaria:* es una operación a nivel de bits aplicada entre imágenes completas (dimensión de m filas y n columnas), que afecta a cada byte (componente R, G o B) de la imagen. Las transformaciones que se pueden aplicar son: operación XOR, OR, AND, rotaciones de bits y NOT. Esta operación modifica la imagen.
2. *Enmascaramiento:* no modifica la imagen, sino que permite ocultar información en una porción específica de ella. Además, se define por una máscara de dimensiones i ≤ m filas y j ≤ n columnas y una semilla s que indica el punto de inicio en la imagen transformada. Esta operación genera un archivo de texto que contiene la suma entre la máscara M y la imagen transformada ID.
3. **Interpretación del archivo .txt:**

Como se mencionó anteriormente, cada archivo de texto tiene dos elementos:

* Primera línea: la semilla s, usada para determinar desde qué posición se aplica el enmascaramiento.
* Líneas siguientes: valores enteros correspondientes a S(k) = ID (k + s) + M(k), donde ID representa la imagen transformada final o intermedia, M la máscara aplicada y k el índice que permite moverse entre cada byte de la máscara.

El archivo sirve para validar si las transformaciones que se realicen permiten obtener la imagen original IO.

1. **Modelo de reconstrucción inversa (hipótesis):**

Dado que las transformaciones fueron aplicadas de forma secuencial, la reconstrucción debe realizarse en *orden inverso:*

* Tomar la imagen final ID.
* Aplicarle la transformación (por medio de operaciones binarias) con la imagen máscara IM.
* Aplicar enmascaramiento a la imagen resultante, sumando la máscara M desde la posición s validando en el archivo .txt N-1.
* Validar si después de los pasos anteriores se obtiene exactamente el archivo .txt N-1. Si no es así, se debe volver a realizar la transformación de la imagen final ID con la imagen máscara IM, usando una operación binaria diferente, hasta que se obtenga el mismo archivo de texto.
* Si se obtiene el mismo archivo, entonces se repite el proceso: se hace la transformación de la imagen resultante con la imagen máscara IM, luego se enmascara, sumando la máscara M desde la posición s con base en el archivo .txt N-2 y se valida si el archivo generado es exactamente el archivo .txt N-2. El proceso continúa hasta llegar a la imagen original IO.

1. **Retos del problema:**

* Desconocimiento de la operación binaria específica usada en cada transformación: se debe probar con distintas operaciones hasta encontrar la correcta en cada etapa.
* Control de índices: como cada píxel tiene 3 bytes (R, G, B), los desplazamientos y posiciones deben estar cuidadosamente alineados con esta estructura.
* Validación exhaustiva: se requiere verificar que cada transformación y enmascaramiento coincidan perfectamente con los archivos .txt, lo cual implica reconstrucciones intermedias precisas.

1. **Estrategia de solución gráfica:**
   1. **Representación visual de las transformaciones aplicadas a la imagen original IO:**

La siguiente imagen representa gráficamente las transformaciones que se realizan dependiendo el caso:

(Si hay más de dos procesos de transformación y enmascaramiento, se irían anexando en el mismo orden, hasta llegar a la imagen distorsionada final)

Imagen que contiene colorido, azul, grande, calle

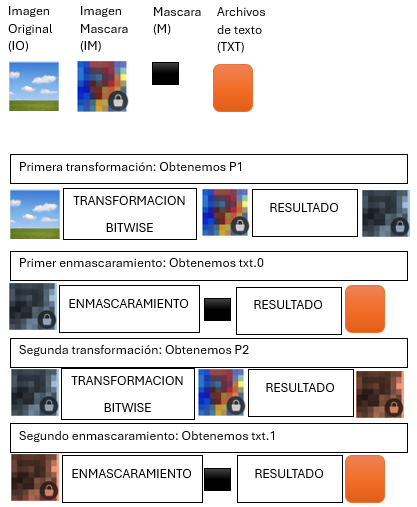
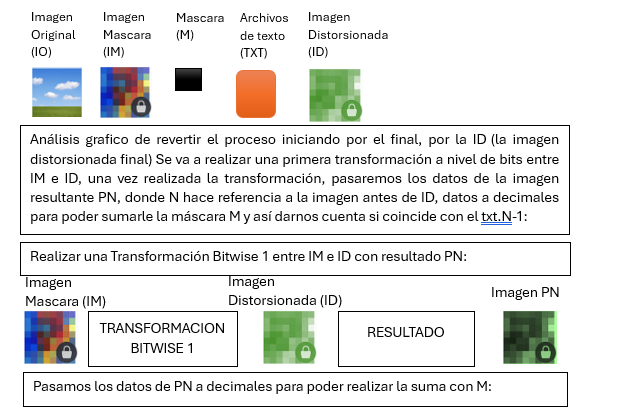
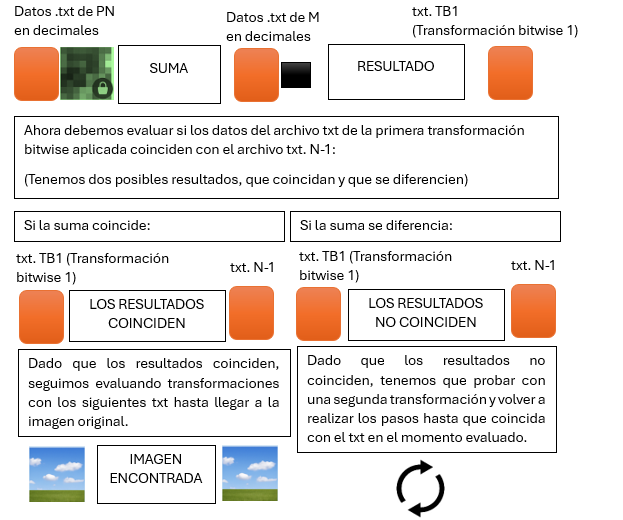
El contenido generado por IA puede ser incorrecto.

Imagen Distorsionada

(ID)

* 1. **Representación visual de la solución planteada para revertir el proceso:**



****

1. **Justificación de la solución planteada:**

La solución planteada es viable, ya que no se requiere ID para reconstruir a IO, pero se parte de ID porque es el último estado conocido. Además, la única manera de verificar si se ha deshecho correctamente una transformación es comparando el archivo .txt.

1. **Resultado esperado:**

Al finalizar el proceso, se habrá revertido toda la cadena de transformaciones y enmascaramientos, obteniendo la imagen original IO en formato BMP, la cual puede ser visualizada y verificada.

1. **Docstrings de funciones para cada transformación (operaciones a nivel de bits):**
   1. **Función AND (Operación &&):**

@brief Aplica la operación AND bit a bit entre los bytes de dos imágenes RGB. Esta función realiza una operación binaria AND (‘&’) entre cada byte correspondiente de dos imágenes de igual tamaño: la imagen distorsionada (ID) y la imagen máscara (IM). Cada byte representa un canal de color (R, G o B) y está almacenado en un arreglo lineal de tipo ‘unsigned char\*’ en formato RGB secuencial (R, G, B, R, G, B, ...). La operación AND se realiza byte a byte, es decir, por cada canal de cada píxel: resultado[i] = ID[i] | IM[i]. El resultado de esta transformación se almacena en un nuevo arreglo dinámico con el mismo formato RGB lineal.

@param ID Arreglo de bytes de la imagen distorsionada.

@param IM Arreglo de bytes de la imagen máscara.

@param dataSize Cantidad total de bytes (ancho × alto × 3 canales).

@return Puntero a un arreglo dinámico de tipo ‘unsigned char\*’ que contiene los resultados de la operación AND por byte.

*Ejemplo de la función AND:*

ID = 11001100 (204 en entero)

IM = 10101010 (170 en entero)

Resultado = 10001000 (136 en entero)

* 1. **Función OR (Operación ||):**

@brief Aplica la operación OR bit a bit entre los bytes de dos imágenes RGB. Esta función realiza una operación binaria OR (‘|’) entre cada byte correspondiente de dos imágenes de igual tamaño: la imagen distorsionada (ID) y la imagen máscara (IM). Cada byte representa un canal de color (R, G o B) y está almacenado en un arreglo lineal de tipo ‘unsigned char\*’ en formato RGB secuencial (R, G, B, R, G, B, ...). La operación OR se realiza byte a byte, es decir, por cada canal de cada píxel: resultado[i] = ID[i] | IM[i]. El resultado de esta transformación se almacena en un nuevo arreglo dinámico con el mismo formato RGB lineal.

@param ID Arreglo de bytes de la imagen distorsionada.

@param IM Arreglo de bytes de la imagen máscara.

@param dataSize Cantidad total de bytes (ancho × alto × 3 canales).

@return Puntero a un arreglo dinámico de tipo ‘unsigned char\*’ que contiene los resultados de la operación OR por byte.

*Ejemplo de la función OR:*

ID = 11001100 (204 en entero)

IM = 10101010 (170 en entero)

Resultado = 10001000 (238 en entero)

* 1. **Función NOT (Operación ~):**

@brief Aplica la operación NOT bit a bit a los bytes de una imagen RGB.

Esta función realiza una operación binaria NOT (‘~’) sobre cada byte de una imagen: la imagen distorsionada (ID).

Cada byte representa un canal de color (R, G o B) y está almacenado en un arreglo lineal de tipo ‘unsigned char\*’ en formato RGB secuencial (R, G, B, R, G, B, ...). La operación NOT invierte todos los bits de cada byte:

resultado[i] = ~ID[i].

El resultado de esta transformación se almacena en un nuevo arreglo dinámico con el mismo formato RGB lineal.

@param ID Arreglo de bytes de la imagen distorsionada.

@param dataSize Cantidad total de bytes (ancho × alto × 3 canales).

@return Puntero a un arreglo dinámico de tipo ‘unsigned char\*’ que contiene los resultados de la operación NOT por byte.

*Ejemplo de la función NOT:*

ID = 11001100 (204 en entero)

Resultado = 00110011 (51 en entero)

* 1. **Función XOR (Operación ^):**

@brief Aplica la operación XOR bit a bit entre los bytes de dos imágenes RGB.

Esta función realiza una operación binaria XOR (‘^’) entre cada byte correspondiente de dos imágenes de igual tamaño: la imagen distorsionada (ID) y la imagen máscara (IM).

Cada byte representa un canal de color (R, G o B) y está almacenado en un arreglo lineal de tipo ‘unsigned char\*’ en formato RGB secuencial (R, G, B, R, G, B, ...). La operación XOR se realiza byte a byte, es decir, por cada canal de cada píxel:

resultado[i] = ID[i] ^ IM[i].

El resultado de esta transformación se almacena en un nuevo arreglo dinámico con el mismo formato RGB lineal.

@param ID Arreglo de bytes de la imagen distorsionada.

@param IM Arreglo de bytes de la imagen máscara.

@param dataSize Cantidad total de bytes (ancho × alto × 3 canales).

@return Puntero a un arreglo dinámico de tipo ‘unsigned char\*’ que contiene los resultados de la operación XOR por byte.

*Ejemplo de la función XOR:*

ID = 11001100 (204 en entero)

Resultado = 01100110 (102 en entero)

* 1. **Función rotación izquierda:**

@brief Aplica una rotación a la izquierda de n (1 ≥ n ≤ 8) bits sobre cada byte de una imagen RGB. Esta función realiza una operación binaria de rotación a la izquierda sobre n bits de cada byte de la imagen distorsionada (ID). Cada byte representa un canal de color (R, G o B) y está almacenado en un arreglo lineal de tipo ‘unsigned char\*’ en formato RGB secuencial (R, G, B, R, G, B, ...).

La operación se realiza byte a byte, es decir, por cada canal de cada píxel:

resultado[i] = (ID[i] << n) | (ID[i] >> (8 - n))

El resultado de esta transformación se almacena en un nuevo arreglo dinámico con el mismo formato RGB lineal.

@param ID Arreglo de bytes de la imagen distorsionada.

@param dataSize Cantidad total de bytes (ancho × alto × 3 canales).

@param n Número de bits a rotar a la izquierda.

@return Puntero a un arreglo dinámico de tipo ‘unsigned char\*’ que contiene los resultados de la rotación.

*Ejemplo de la función rotación izquierda (3 bits):*

ID = 11001100 (204 en entero)

n = 3

*Resultado*

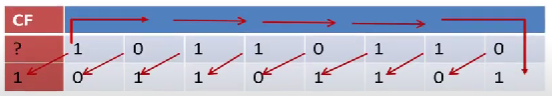
*Inicio*: 11001100

*Rotación 1:* 10011001

*Rotación 2:* 00110011

*Rotación 3:* 01100110

01100110 (102 en entero)

*Aplicación visual de cada rotación:*

*Bajo la fórmula:*

resultado[i] = (ID[i] << n) | (ID[i] >> (8 - n))

ID = 11001100 (204 en entero)

n = 3

*Resultado*

ID[i] << 3 = 01100000

ID [i] >> (8-3) = ID[i] >> 5 = 00000110

Aplicamos OR:

01100000

00000110

OR 01100110 (102 en entero)

* 1. **Función rotación derecha:**

@brief Aplica una rotación a la derecha de n (1 ≤ n ≤ 8) bits sobre cada byte de una imagen RGB. Esta función realiza una operación binaria de rotación a la derecha sobre n bits de cada byte de la imagen distorsionada (ID). Cada byte representa un canal de color (R, G o B) y está almacenado en un arreglo lineal de tipo ‘unsigned char\*’ en formato RGB secuencial (R, G, B, R, G, B, ...).

La operación se realiza byte a byte, es decir, por cada canal de cada píxel:

resultado[i] = (ID[i] >> n) | (ID[i] << (8 - n))

Esta operación asegura una rotación circular, es decir, los bits que "salen" por la derecha son reinsertados por la izquierda. El resultado de esta transformación se almacena en un nuevo arreglo dinámico con el mismo formato RGB lineal.

@param ID Arreglo de bytes de la imagen distorsionada.

@param dataSize Cantidad total de bytes (ancho × alto × 3 canales).

@param n Número de bits a rotar a la derecha.

@return Puntero a un arreglo dinámico de tipo ‘unsigned char\*’ que contiene los resultados de la rotación.

*Ejemplo de la función rotación derecha (3 bits):*

ID = 11001100 (204 en entero)

n = 3

*Resultado*

*Inicio:* 11001100

*Rotación 1:* 01100110

*Rotación 2:* 00110011

*Rotación 3:* 10011001

10011001 (153 en entero)

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.*Aplicación visual de cada rotación:*

*Bajo la fórmula:*

resultado[i] = (ID[i] >> n) | (ID[i] << (8 - n))

ID = 11001100 (204 en entero)

n = 3

*Resultado*

ID[i] >> 3 = 00011001

ID[i] << (8 - 3) = ID[i] << 5 = 10000000

*Aplicamos OR:*

00011001

10000000

OR10011001 (153 en entero)

**NOTA:** En la solución planteada no se tendrá en cuenta la operación binaria desplazamiento hacia la izquierda (<<) y desplazamiento hacia la derecha (>>), dado que son operaciones parcialmente irreversibles (al aplicarlas se puede “perder” información sobre la imagen BMP).

**Acerca de funciones de NOT:** Estamos analizando si realmente necesitamos esta función para la solución planteada, ya que esta función no requiere tomar datos de IM y una imagen PN (resultante de cada transformación) o ID, porque se realizan bajo una sola imagen, bajo unos datos específicos, ya sean de PN o de ID, pero no compuesta.

1. **Docstrings de funciones para procesar la información:**
   1. **Función de conversión unsigned char\* 🡪 int\***

@brief Convierte un arreglo dinámico de bytes (unsigned char\*) a enteros (int\*) manteniendo el orden RGB. Esta función toma un puntero a un arreglo de tipo ‘unsigned char\*’, que contiene los valores RGB de una imagen en formato lineal (R, G, B, R, G, B, ...), y lo convierte en un nuevo arreglo dinámico de tipo ‘int\*’. Esta conversión permite realizar operaciones posteriores (como enmascaramiento. Cada byte del arreglo original se convierte explícitamente a un entero utilizando ‘static\_cast<int>’, manteniendo la misma secuencia de datos RGB.

Ejemplo: salida[i] = static\_cast<int>(entrada[i]);

@param entrada Puntero a un arreglo dinámico de tipo ‘unsigned char\*’ que representa los datos RGB de una imagen.

@param dataSize Tamaño total del arreglo (en bytes), equivalente a ancho × alto × 3 canales.

@return Puntero a un nuevo arreglo dinámico de tipo ‘int\*’ con los mismos valores convertidos a enteros, ordenados en formato RGB secuencial (R, G, B, R, G, B, ...).

1. **Consideraciones importantes para la eficiencia:**

* Cuando se esté realizando el proceso de enmascaramiento entre la imagen transformada y la máscara M, basta con revisar si al menos un valor S(k) es diferente con respecto al archivo de texto N-1 para determinar si la transformación aplicada fue correcta o no.

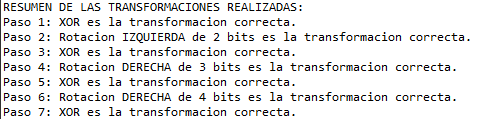
1. **Consideraciones importantes para liberar memoria dinámica:**

* Cuando se obtenga el arreglo de tipo ‘unsigned char’ que se crea en las funciones de transformaciones, se puede liberar la memoria dinámica que ha sido reservada para el arreglo de las imágenes ID e IM, ya que el espacio ocupado por estos puede ser usado para almacenar otros datos, como el arreglo que contiene los datos de la máscara o la suma resultante del enmascaramiento.

1. **Conclusiones**

* Dada la posible solución lineal y obteniendo el resultado final de las transformaciones inversas a ID  y lograr llegar a la IO, nos hemos dado cuenta de que las funciones NOT, AND y OR, son transformaciones que no se usan:

En la siguiente imagen podemos observar la conclusión anterior:

****