Reconstrucción de Imagen a Nivel de Bits

# 1. Análisis del Problema y Consideraciones de Solución

Se recibió una imagen dañada a nivel de bits. Esta imagen fue transformada aplicando operaciones bit a bit: desplazamientos, rotaciones, o XOR, sumadas a una máscara, siguiendo un patrón desconocido. Nuestro objetivo fue reconstruir la imagen original, detectando automáticamente las transformaciones aplicadas.

Consideraciones técnicas:

- Las operaciones a nivel de bits deben respetar 8 bits.

- Valores de canales RGB pueden superar 255 temporalmente por la máscara.

- Memoria dinámica debe ser gestionada cuidadosamente.

- Rotaciones y desplazamientos deben ser implementados de manera segura.

# 2. Esquema de Tareas Definidas

- Cargar imagen dañada (I\_D.bmp).

- Cargar máscara (M.bmp).

- Cargar imagen complementaria para XOR (I\_M.bmp).

- Iterar sobre los archivos Mx.txt.

- Detectar transformación correcta para cada paso:

- Aplicar transformación a toda la imagen.

- Exportar la imagen final reconstruida (Imagen\_Arreglada.bmp).

# 3. Algoritmos Implementados

a) Desplazamiento de Bits

- Izquierda: (valor << bits) & 0xFF

- Derecha: (valor >> bits)

b) Operación XOR

- resultado = pixel1 ^ pixel2

c) Rotaciones de Bits

- Izquierda: ((valor << bits) | (valor >> (8 - bits))) & 0xFF

- Derecha: ((valor >> bits) | (valor << (8 - bits))) & 0xFF

d) Aplicación de transformaciones

- Sobre todo el arreglo de píxeles cargado en memoria.

e) Verificación de enmascaramiento a cada tipo de transformación

- Comparar resultado + máscara[i] contra referencia[i].

# 4. Problemas de Desarrollo Afrontados

- Límites de 8 bits: Necesidad de aplicar & 0xFF tras desplazamientos y rotaciones.

- Valores mayores a 255: Debido a la suma de la máscara, se permitieron variables de 32 bits (unsigned int).

- Memoria Dinámica: Se garantizó delete[] a toda memoria cargada con loadPixels() o loadSeedMasking().

- Operaciones bit a bit: Verificación exhaustiva de no perder información durante desplazamientos y rotaciones.

- Modularidad: Separación en archivos independientes para claridad (funciones\_imagen, funciones\_de\_transformacion, funciones\_de\_verificacion).

# 5. Evolución de la Solución

- El código original era monolítico y desordenado.

- Se modularizó progresivamente el programa.

- Se implementó un flujo robusto para manejar la reconstrucción.

- Se agregó seguridad en el manejo de memoria y enmascaramientos.

- Se agregó verificación automática paso a paso.

- Se exportó la imagen final corregida en formato BMP.

Resultado Final:

El programa permite ahora recuperar una imagen original dañada, detectando y aplicando automáticamente las transformaciones bit a bit requeridas.