**Informe análisis desafio 1**

**Luis Miguel Alzate Rios**

**Jerónimo Herrera Espinosa**

**1. Introducción**

El propósito de este reto es reconstruir una imagen original a partir de una imagen modificada y una serie de enmascaramientos aplicados sobre ella. La imagen final se obtiene tras efectuar diversas transformaciones en la imagen original, y la tarea consiste en identificar esas transformaciones en el orden adecuado y revertirlas paso a paso.

Este informe detalla el análisis del problema, el enfoque secuencial inverso adoptado para resolverlo, y la forma en que cada módulo contribuye al proceso de reconstrucción.

**2. Descripción del problema**

En este reto, se nos proporciona una imagen transformada final (ID), una máscara (M), y una serie de archivos . txt (M1. txt, M2. txt, . . . , Mn. txt). Cada archivo . txt contiene los resultados de un enmascaramiento de una imagen transformada aplicada en cada etapa del proceso. Nuestra tarea es:

* Identificar qué transformaciones se aplicaron a la imagen original, en qué orden.
* Deshacer estas transformaciones de manera secuencial para reconstruir la imagen original.

Para hacerlo, debemos explorar las transformaciones posibles que fueron implementadas en la imagen original, verificar cuál se utilizó en cada etapa, y aplicar sus transformaciones inversas en el orden adecuado.

**3. Análisis del enfoque adoptado**

**Enfoque secuencial inverso**

En lugar de probar todas las combinaciones posibles de transformaciones y realizar un retroceso (lo cual sería ineficaz), hemos decidido adoptar un enfoque secuencial inverso. Esto significa que:

Comenzamos con la imagen final (ID).

Para cada archivo . txt (desde Mn. txt hasta M1. txt):

* Probamos las transformaciones posibles sobre la imagen actual.
* Verificamos si alguna transformación aplicada produce el mismo resultado que el contenido del archivo . txt.
* Cuando encontramos una transformación válida, sabemos que fue la última que se aplicó (en el proceso de distorsión), y por lo tanto es la primera que debemos deshacer.
* Aplicamos la inversa de esa transformación y pasamos al siguiente archivo . txt.
* Repetimos el proceso hasta que lleguemos a M1. txt, con lo cual habremos deshecho todas las transformaciones en orden inverso y recuperado la imagen original.

Este enfoque es más eficaz y directo, ya que no necesitamos explorar todas las combinaciones posibles de transformaciones, .

**4. Descripción de los módulos**

**Módulo 1: Entrada y salida**

**Objetivo:** Cargar la imagen final transformada (ID), la imagen aleatoria (IM), y la máscara (M) desde archivos BMP, y cargar los archivos . txt con los resultados de enmascaramiento.

**Funciones:**

* Unsigned char\* loadPixels(. . . ):

Esta función se recibe y carga una imagen BMP y la convierte en un arreglo dinámico con los valores de los canales RGB de cada pixel.

Entradas:

>QString input = variable en la que se guarda la ruta de la imagen BMP como un string.

>int &width = anchura de la imagen BMP.

>int &height = altura de la imagen BMP.

Salida:

>Unsigned char\* pixelData = arreglo dinámico con los valores de los canales RGB.

* Bool exportImage(. . . ):

Esta función recibe un arreglo dinámico (con la misma estructura con los canales RGB) y la convierte y exporta como una imagen BMP

Entradas:

>Unsigned char\* pixelData = arreglo dinámico con los valores de los canales RGB de los pixeles de la imagen.

>int width = anchura de la imagen BMP.

>int height = altura de la imagen BMP.

>QString archivoSalida = string con el nombre y la dirección con la que exportaremos el arreglo dinámico ya convertido a imagen BMP.

Salidas:

>Esta función retorna un booleano que indicara si la operación tuvo éxito (true) o no (false).

* Unsigned int\* loadSeedMasking(. . . ):

Esta función carga el archivo . txt que contiene la semilla, el número de píxeles enmascarados, y los resultados del enmascaramiento.

Entradas:

>Const char\* nombreArchivo = tal y como lo indica el nombre, esta variable contiene el nombre del archivo que deseeamos cargar.

>int &seed = semilla desde donde se empieza el enmascaramiento.

>int &n\_pixels = numero de pixeles de la imagen BMP.

Salidas:

>unsigned int\* RGB = puntero hacia el arreglo con los datos RGB.

**Módulo 2: Transformaciones.**

**Objetivo:** Probar todas las transformaciones posibles sobre la imagen en cada etapa para identificar cuál de ellas produce el resultado de enmascaramiento esperado.

**Funciones:**

* Unsigned char\* aplicarXOR(. . . ):

Esta función aplica una transformación XOR entre la imagen y la imagen aleatoria.

Entradas:

>unsigned char\* imagen = arreglo dinámico de los datos RGB de la imagen.

>unsigned char\* imagenMascara = arreglo dinámico de la imagen aleatoria con la que aplicaremos el XOR a la imagen.

>int tamaño = tamaño de los arreglos dinámicos imagen e imagen aleatoria (ambos tienen el mismo tamaño).

Salidas:

>unsigned char\* imagenXOR = arreglo dinámico con los datos RGB resultantes de la aplicación del XOR entre la imagen y la imagen mascara (imagen ^ imagenMascara).

* Unsigned char rotación(. . . ):

Esta función aplica el operador de rotación a los bits de un elemento, esta función ayuda a la función de rotarBits.

Entradas:

>unsigned char elemento = elemento (dato R,G o B) al que aplicaremos el rotamiento.

>unsigned int numRotaciones = números de bits a rotar (el máximo es 8)

>char dirección = carácter el cual será ‘D’ o ‘I’ para representar el sentido en el cual se aplicara la rotación.

Salidas:

>unsigned char elementoRotado = elemento después de aplicar la rotación en la dirección y cantidad de bits indicada.

* Unsigned char\* rotarBits(. . . ):

Esta función rota cierta cantidad de bits a cada canal RGB de los pixeles de la imagen en una dirección específica.

Entradas:

>unsigned char\* imagen = imagen a la cual le aplicaremos la rotación de bits.

>int tamaño = tamaño del arreglo “imagen”.

>int NumRotaciones = cantidad de bits a rotar.

>char dirección = carácter que indicara la dirección de la rotación (‘I’ o ‘D’).

Salidas:

>unsigned char\* imagenRotada = arreglo con los valores de la imagen original ya rotada.

* Unsigned char\* desplazarBits(. . . ):

Esta función desplaza cierta cantidad de bits los canales RGB de los pixeles de la imagen en una dirección determinada.

Entradas:

>unsigned char\* imagen = imagen original que deseamos aplicar el desplazamiento.

>int tamaño = tamaño del arreglo “imagen”.

>int NumDesplazamientos = cantidad de bits a reemplazar.

>char dirección =carácter que indicara la dirección de la rotación (‘I’ o ‘D’).

Salidas:

>unsigned char\* imagenDesplazada = arreglo con los valores de la imagen original ya desplazada.

**Módulo 3: Validación de Transformaciones**

**Objetivo:** Verificar qué transformación aplicada sobre la imagen produce el resultado del enmascaramiento conforme a los archivos . txt.

**Funciones:**

* Bool validarEnmascaramiento(. . . ):

Esta función verifica si al validar el enmascaramiento aplicado sobre la imagen candidata esta coincide con su respectivo archivo .txt.

Entradas:

>const unsigned char \* imagen = arreglo de la imagen original la cual queremos verificar.

>const unsigned char\* mascara = arreglo con la mascara que aplicaremos a la imagen original.

>const unsigned char\* resultado = resultado esperado después de aplicar la mascara al archivo original.

>int seed = semilla que representa la ubicación inicial para el enmascaramiento de la imagen.

>int n\_pixels = cantidad de pixeles a enmascarar

>int total\_bytes = numero total de bytes.

Salidas:

Esta función retorna un booleano que denota si la validación sale correcta (true) o no (false).

* Bool detectarTransformacion(. . . ):

Esta función evalúa las transformaciones posibles sobre la imagen y confirma cuál se alinea con el archivo . txt en ese momento. Registra la transformación identificada.

Entradas:

>unsigned char\* imagenActual = imagen a la cual se le aplicaran y detectara la operación usada.

>unsigned char\* IM = imagen random con la cual se verifica la operación XOR.

>unsigned char\* mascara = mascara que fue aplicada después de cada operación.

>unsigned int\* resultado = resultado esperado con el cual se verifica que operación fue usada.

>int seed = semilla desde donde empieza el proceso de enmascaramiento.

>int n\_pixels = numero de pixeles

>int total\_bytes = numero total de bytes.

>char\* transformacionUsada = variable en donde se guardara la transformación usada.

Salidas:

La función retorna un bool de True en el momento que detecte la transformación usada (la transformación usada queda guardada en la variable transformacionUsada)

**Módulo 4: Reconstrucción**

**Objetivo:** Implementar las transformaciones inversas en el orden adecuado para restaurar la imagen original.

**Funciones:**

* Void aplicarTransformacionesInversas(. . . ):

esta función ejecuta las transformaciones inversas en la secuencia correcta, comenzando desde el último archivo . txt hasta el primero.

Entradas:

>unsigned char\* entrada = imagen a la cual le aplicaremos la transformación inversa.

>unsigned char\* salida = arreglo donde se guardara el resultado de la aplicación de la transformación inversa sobre la imagen entrada.

>unsigned char\* imagenRandom = imagen random (útil si la transformación aplicada fue XOR).

>const char\* nombreTransformacion = como su nombre lo indica, variable donde se guarda el nombre de la transformación que vamos a invertir.

>int totalBytes = cantidad total de bytes.

* Bool reconstruirSecuencial(. . . ):

Esta función organiza todo el proceso de reconstrucción, desde la identificación de las transformaciones hasta la implementación de las inversas.

Entradas:

>unsigned char\* arrayImagenFinal = arreglo dinámico de la imagen final.

>unsigned char\* arrayImagenRandom = arreglo dinámico de la imagen random (útil para la transformación XOR).

>unsigned char\* arrayMascara = arreglo de la mascara.

>int numArchivos = cantidad de archivos/transformaciones usadas.

>int ancho = ancho de la imagen.

>int alto = alto de la imagen.

>const char\* archivoSalida = archivo de salida

Salidas:

Esta función retorna un booleano que nos indicara si la operación fue exitosa o no.

**5. Conclusión**

En síntesis, el enfoque secuencial inverso que hemos adoptado resulta más eficaz que intentar todas las combinaciones posibles de transformaciones. Al utilizar los archivos . txt como guía, podemos identificar qué transformación se aplicó en cada etapa y revertirlas de manera progresiva para recuperar la imagen original.

Cada módulo desempeña una función concreta en el proceso de carga de datos, aplicación de transformaciones, validación y verificación de transformaciones, y finalmente la reconstrucción de la imagen.