INFORME DE ANALISIS DE LECTURA COMPLETO

KEWI YILIAN MOYA CATAÑO 1033487777

MARDEY SOLANYI ARIAS GAVIRIA 1039689519

La empresa informá2 nos pide la creación de un algoritmo que revierta una imagen distorsionada a la original. Nos proporcionan que la imagen se les pudo haber hecho unas transformaciones que son a nivel de bits y una máscara que se le sumará a un valor de la imagen. Primero nos centraremos en hacer pruebas con transformaciones y la máscara para entender cómo funcionan y afecta los datos.

Análisis de las transformaciones

Tenemos que las trasformaciones son:

* Un xOr con una imagen de ruido
* Desplazamientos de bits a la derecha o izquierda
* Rotaciones a la derecha o la izquierda

Análisis de los desplazamientos

En el caso del desplazamiento de bits a la derecha o a la izquierda sabemos que el máximo rango de movimiento puede ser de 8 bits. Sabemos que esta transformación se les hace a todos los bytes RGB de la imagen. Realizando algunos experimentos descubrimos que los desplazamientos tienen perdida de la información.

Les mostrare un ejemplo con dos desplazamientos una de 8 bits a la derecha y otro a la de 4 bits a la izquierda:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

Bit original:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Primer desplazamiento:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Segundo desplazamiento:

Como pueden ver en este ejemplo los datos que se desplazan hacia un lado se pierden, y en caso de que se vuelva a desplazar, pero en su contraria, seguirá existiendo pérdida importante de información. En resumen, este operador daña la integridad de los datos y no sería útil para codificar ya que son operaciones irreversibles

Por otro lado, si esta información se pudiera almacenar en algún lugar, deberían ser específicos en un archivo aparte para posteriormente incorporarlos a la imagen original, sin embargo, este no es el caso.

Análisis de las rotaciones

En el caso de las rotaciones de bits a la derecha o a la izquierda sabemos que la máxima cantidad de movimientos es de 8 bits. Esta transformación se les aplica a todos los bytes RGB de la imagen.

Realizando algunos experimentos descubrimos que las rotaciones no tienen pérdida de la información.

Como la rotación de bit hace que el bit de un extremo pase a un extremo a otro no hay modificación de los datos ya que se hacen solo con la misma información, haciendo posible que vuelvan a su estado original.

Un ejemplo muy específico en este caso es que roten 4 bits a la izquierda y luego que se roten 4 bits a la derecha:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |

Bit original:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |

Primera rotación:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |

Segunda rotación:

Cómo podemos visualizar en este caso muy específico, observamos que con las rotaciones que fueron la misma cantidad de bit, pero a direcciones contrarias, vemos que volvieron a los datos originales; esto significa que las rotaciones pueden volver a su estado original y que no habrá pérdida de información. Como conclusión las rotaciones mantienen la integridad de los datos.

Análisis de xOr

En ese caso es un poco más complejo ya que la forma de hacer esa transformación tenemos una imagen que se va a utilizar para comparar cada a byte con la operación xOr de la imagen original. Con esto nos conlleva que toda la imagen va a ser transformada a partir de una imagen aleatoria del mismo tamaño.

Forma

El contenido generado por IA puede ser incorrecto.Primero vamos a ver cómo funciona xOr.

Teniendo en cuenta el caso en el que tomemos los primeros pixeles de cada imagen la originaria y las distorsionada. Y de eso tomamos el primer byte de cada uno y realizamos la operación y con el resultado realizamos la misma operación, pero se hace contra en byte de la imagen distorsionada.

M/O: byte de la imagen original

M/D: byte de la imagen distorsionada

C/R: resultado de xOr de M/O y M/D

C/Q: resultado de xOr de C/R y M/D

v C/Q

Tabla #2

Tabla #1

|  |  |  |
| --- | --- | --- |
| **M/O** | **M/D** | **C/R** |
| 0 | 1 | 1 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

|  |  |  |
| --- | --- | --- |
| **C/R** | **M/D** | **C/Q** |
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |

Como podemos ver comparando los resultados de la operación xOr hecho en las tablas nos damos cuenta a qué podemos volver a la información original,

Y esto se comprueba ya que M/O y C/Q, Sus valores de byte son los mismo, llegando a la conclusión que si guarda la integridad de los datos.

Conclusión del análisis de las trasformaciones.

Después de analizar cada tipo de transformación nos percatamos que verdaderamente las únicas 2 transformaciones que se pueden aplicar a la imagen sin perder información serían las rotaciones y las operaciones xOr. Ya que, según nuestros análisis y conclusiones de los diferentes tipos de transformaciones, consideramos que los desplazamientos pueden ser una trampa que nos aleja del camino correcto para hallar la imagen original.

Por tanto:

* Transformaciones que si se utilizan: xOr y rotaciones.
* Transformaciones que no se utilizan: desplazamientos.

Análisis de la mascara

la información que nos proporcionan de la máscara es que se le suma como valor entero a entero, y es más pequeña que la imagen original. La máscara utiliza un valor S aleatorio, que desplazará S pixeles desde el píxel cero de la imagen.

La máscara se añade a la imagen en cada paso de la transformación, esto nos lleva a pensar que la imagen tiene pocos pasos y vemos que la máscara solo distorsiona una pequeña parte de la imagen original. Pero si son muchos pasos la máscara se distorsionará mucho cuando intente volver a la original.

Un ejemplo grafico:

Imagen original:

Mascara:

Imagen A Imagen B

Como vemos en la imagen A se hicieron pocos pasos teniendo pocas máscaras eso nos da como resultado que si queremos pudiéramos buscar la forma de encontrar la imagen original a prueba de error solamente a transformación. pero siendo muy ineficientes por qué tendríamos que hacer una imagen por cada caso posible de transformación hasta encontrar una imagen muy parecida a la original, pero con fallas por las máscaras.

Pero en la imagen número B hay una cantidad de pasos mayor haciendo que la integridad de los datos de la imagen se vea afectada.

***Conclusión de la máscara***

Los valores de la máscara que están en los txt se deben eliminar (desenmascarar) haciendo una resta que es el proceso inverso a la suma; de esta manera ya casi aseguramos la integridad de los datos y tenemos la capacidad de volver a la imagen original sin corrupción.

Creación de la estructura del algoritmo

Para empezar, veamos cómo se hace el proceso de encriptación:

**TA:** trasformaciones

**M:** mascara

**TX:** archivo dejado en el proceso (pista)

**M\_O:** imagen original

**M\_D:** imagen distorsionada

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Paso 1*** | | | | | ***Paso 2*** | | | | | ***Paso 3*** | | | |
| **M\_O** | TA | M | TX.1 | M\_D1 | **M\_D1** | TA | M | TX.2 | M\_D2 | **M\_D2** | TA | M | M\_D3 |

Como vemos en el proceso de encriptación se divide en pasos y los pasos se dividen

En entrada de la imagen, transformaciones, máscara, archivo de texto, imagen encriptada. Este proceso se puede repetir varias veces. Pero nosotros no tendremos esas imágenes internadas ya que nuestras entradas son:

* Imagen distorsionada final. (en el caso de arriba seria M\_D3)
* Archivos de texto. (en el caso de arriba serian TX.1, TX.2)
* Máscaras
* Y una imagen de ruido gaussiano que se puede utilizar en una transformación.

Si quieres regresar a la imagen original tendríamos que hacer el proceso inverso. Y este proceso tenemos que encontrar las trasformaciones que se le hacen a la imagen original y de esta manera aplicarlo hasta llegar a la imagen original.

En la siguiente imagen lo veremos de manera más clara:Diagrama

El contenido generado por IA puede ser incorrecto.

(donde ver mejor: <https://miro.com/app/board/uXjVID8nT4Y=/?share_link_id=943962661199> )

Hay que tener en cuenta que las máscara solo se aplica entre pasos de encriptación, Ya después de ver la imagen el proceso de la decodificación está más claro, Primero es encontrar las últimas transformaciones que le hicieron a la imagen distorsionada final, después de esos tendríamos una imagen intermedia, le quitamos la máscara y de nuevo encontrar las trasformaciones y obtendríamos otra imagen intermedia, le quitamos la máscara y encontrar las transformaciones y de ese modo encontramos la imagen original. Esta es la lógica de la decodificación de la imagen.

Nota: de aquí en adelante las transformaciones se le van a recibir el nombre de filtros

***¿Cómo encontrar los filtros***

Ya sabemos que los únicos filtros que se van a hacer son el xOr y rotaciones, pero ahora cómo sabemos si se aplicaron o no y cómo se aplicaron. Estas se encuentran con las pistas que son los archivos txt, Ya que la información que proporcionan es de los pixeles desde la semilla (píxel de la imagen); hasta el final del pedazo de imagen que fue afectado por la suma de la máscara. Como la imagen nunca cambia sus dimensiones en todo el proceso entonces son puntos de referencias claros para poder comparar. A continuación, el siguiente paso es usar los valores que nos refieren el archivo txt y la imagen distorsionada. Y lo que hace es aplicar los filtros a los valores de la imagen distorsionada y compararlos hasta que sean iguales a los del archivo txt. Y así se encontrarían los filtros que se hicieron.

***Base de datos***

En el transcurso del programa tendremos tres bases de datos, todas serian de una dimensión los cuales son: un array y cada tripleta de espacios es un píxel de la imagen.

* Imagen distorsionada (cuando se confirme acciones para volver a la imagen original se harán aquí mismo)
* Imagen Ruido gaussianos (imagen que es necesaria para el xOr)
* Máscara

Representación gráfica:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Como podemos ver para movernos un píxel en la base de datos sería el # de pixeles por 3. (n = número de pixeles, n\*3).

***Tema de eficiencia en el algoritmo***

Relacionado al tema de eficiencia del algoritmo tenemos claro que no podemos utilizar completamente todas las bases de datos para probar que filtros son los correctos ya que las bases de datos ocupan demasiado espacio, Un ejemplo sería una imagen de 100x200 pixeles y este le añadimos que cada píxel RGB el base se guarda en tres espacios. (100x200) x3 y este nos da que ocupa un espacio de memoria de 60.000, y otros 60.000 por la imagen de ruido y sabemos que la máscara va a hacer menor. Ya que trabajar con esa cantidad de datos y que una constantemente se crearán réplicas para poder volver a la imagen original. Pero también solo podemos con los valores que nos dan la máscara y esto sería más eficiente ya que no trabajaríamos con una gran cantidad.

Teniendo en cuenta lo anterior, decidimos trabajar sobre las pruebas comparando los valores de un píxel a otro píxel (píxel de la imagen sin la máscara y con la transformación con el valor del píxel del txt) , después de saber que ya los tenemos la información correcta de que filtros se aplicó, el siguiente paso es replicar el mismo procedimiento hasta llegar a la imagen original.

***Funciones necesarias.***

Los profesores nos proporcionaron algunas funciones que nos ayudan en el proceso de codificación.

Loadpixeles: carga una imagen y la convierte en formato de 24 bits, extrae los datos de esta en un puntero que contiene un arreglo lineal y retorna este puntero. (con esta función podemos sacar la información de la imagen distorsionada)

ExportImage: crea imagen de QT vacía, y copia los valores de RGB del puntero anterior (se pasa como argumento de la función) y guarda esta imagen.

LoadSeedMasking: Lee los archivos de texto dados para la práctica, toma la primera línea como semilla y devuelve un arreglo con datos RGB y cantidad de pixeles.

Loadpixeles2: carga una imagen y la convierte en formato de 24 bits, extrae los datos de esta en un puntero que contiene un arreglo lineal y retorna este puntero. (en resumen, esta función podrá ser utilizada para sacar la información de las imágenes de la entrada) y no devuelve algunos valores a comparación de la original.

Verificación\_imagenes: comprueba que si se subió correctamente las imágenes. Y que la distorsionada y la gaussiana sean de igual tamaño.

Verificaciontxt: verifica que los datos del archivo se subieran correctamente.

prueba\_xor: comprueba que si se hizo un xOr a la distorsionada.

imagen\_xor: hace xOr a toda la imagen distorsionada.

Rotar\_r: rota un valor una cantidad n de bits a la derecha.

Rotar\_l: rota un valor una cantidad n de bits a la izquierda.

Prueba\_r: comprueba si se hizo una rotación a la derecha.

Prueba\_l: comprueba si se hizo una rotación a la izquierda.

imagen\_rotar\_l: rota toda la imagen una cantidad de bits a la izquierda.

imagen\_rotar\_r: rota toda la imagen una cantidad de bits a la derecha.

Teniendo en cuenta las funciones proporcionadas y la definición de las funciones, se lleva a cabo la codificación de la siguiente manera:

1. Utilizamos la función de loadpixeles para que nos brinde la información de la imagen distorsionada.
2. Según nuestro análisis anterior, queremos desenmascarar, por tanto, debemos utilizar la función loadpixeles nuevamente, pero esta vez con la imagen de la máscara para extraer la información a un arreglo y junto con la función loadseedmasking para obtener la información de los archivos de texto.

Después de tener la imagen distorsionada y la imagen de la máscara en arreglos, se crea una función que se encarga de restar la imagen de la máscara con el archivo txt, permitiéndonos tener la porción de la imagen.

1. A continuación, con esa información, debemos crear funciones que se encarguen de realizar las diferentes transformaciones que queremos aplicar.

Inicialmente, se aplica la transformación xor y se compara la base de datos con el archivo txt, en caso de que los pixeles no sean los mismos, entonces se aplican los desplazamientos a la derecha o a la izquierda, y se repite el proceso de comparación.

1. Cuando finalmente se encuentran los datos que coinciden, entonces se muestra por pantalla cuál fue la transformación aplicada.
2. Este procedimiento se repite hasta que se encuentre la imagen original.

***Navegación en la base de datos***

A lo largo del programa necesitamos en ocasiones algunos valores específicos de las 4 bases de datos, esto significa que se implementaron diversas funciones para tareas en particular en el programa. Estas funciones se enfocan en:

* escoger valores específicos en la base de datos
* cómo navegar en la base de datos para modificar a sólo ciertos valores
* cómo modificar toda la base de datos

todo esto teniendo en cuenta de no crear replicas durante el programa, sino trabajando directamente sobre las bases de datos (punteros).

El funcionamiento del programa es el siguiente.

* Inicio
* Subir los archivos e imágenes
* Hay que confirmar que se subieron correctamente
* Sacar el resultado esperado
* Prueba xOr. Determina si se hizo un xOr
* Prueba rotación. Determina si se hizo alguna rotación
* Termina el programa.

Esta es la parte central de los procesos y decisiones. Esta es la línea principal que toma el programa. Claramente en las decisiones pueden determinar un cambio de acciones.

(donde ver el diagrama: <https://miro.com/app/board/uXjVIBFGFiQ=/?share_link_id=630170873550> )

***Problemas de desarrollo afrontado***

Durante el programa nos percatamos que los archivos no estaban siendo leídos correctamente por las funciones que cargaban la información, por tanto, en las entradas principales del programa (txt, i\_m, i\_d…), tuvimos que poner la ruta absoluta de donde se encontraban los archivos para que el programa pudiera identificarlas y encontrarlas.

Dentro de la solución del problema y en el proceso de codificación, tuvimos que ser muy rigurosos con el tipo de variable que iba a tomar una función y el valor que iba a retornar (bases de datos) para evitar que los tipos generaran errores.

Inicialmente se descartó la posibilidad de trabajar con toda la información de la imagen completa, por tanto, inicialmente decidimos trabajar comparando toda la información que se encontraba en el txt y en la máscara. Sin embargo, se llegó a la conclusión de que cada transformación iba a generar un nuevo comportamiento en la imagen y por esta razón, por cuestiones de eficiencia y análisis, consideramos que no era necesario comparar todo el txt y toda la máscara, sino solamente la primera posición del primer píxel, con la primera posición y el primer pixel del txt

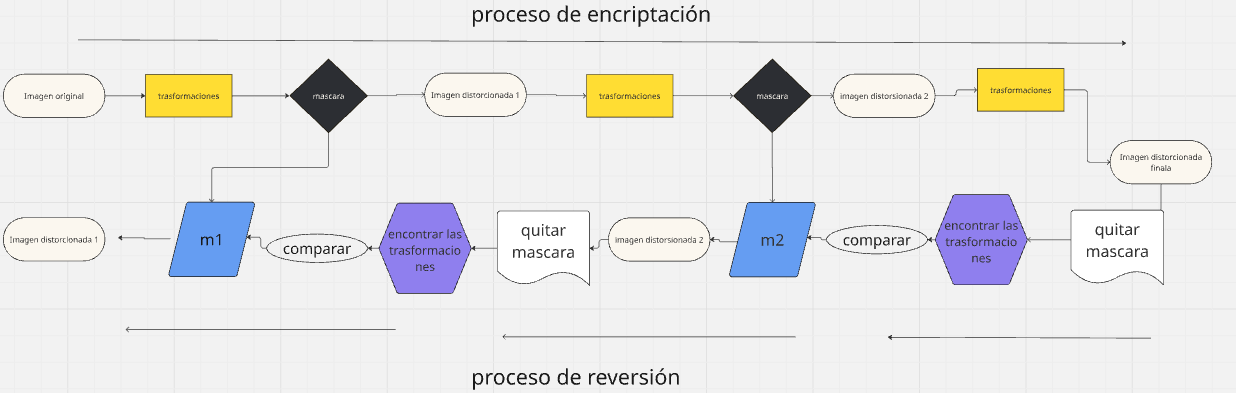
***Evolución de la solución y consideraciones en la implementación***

Inicialmente la propuesta de la solución estaba basada en el análisis del siguiente diagrama de flujo.

Diagrama

El contenido generado por IA puede ser incorrecto.

Sin embargo, a medida que el proceso de análisis se iba haciendo más riguroso, nos percatamos que las imágenes resultantes (imágenes de ruido de cada paso), realmente no estaban siendo sumadas desde una posición semilla con una máscara como se creía, sino que la única base de datos que estaba siendo modificada con la máscara, era el pedazo de imagen que representaba el txt. De esta manera el proceso se simplifica a tal punto que, solamente se debe restar la máscara a los txt y posteriormente encontrar las transformaciones, si la imagen en la posición de la máscara genera el mismo resultado que las posiciones de los txt, entonces eso quiere decir que la transformación fue hallada y ahora simplemente se debe aplicar a toda la imagen. A continuación, repetir el proceso hasta que se halle la imagen original. Por tanto, el diagrama de flujo que representa la implementación final es:



Por otro lado, para hacer el código más eficiente, se decidió que no era necesario comparar toda la base de datos de la máscara con todo el txt, ya que, con aplicar una transformación específica, cada resultado de la transformación espera un valor diferente, por tanto, solamente se comparó la primera posición del primer píxel de la máscara junto con la primera posición del píxel en la imagen (en la posición de la máscara). Si ambos pixeles R representaban el mismo número, esto quería decir que la transformación es la correcta y por tanto se debe aplicar a toda la imagen. Con esta optimización se hace más eficiente el programa, evitando comparar toda la base de datos.