README
Arroyo Rivera Juan José 416053223

1. *Definición del problema:*
   Crear un programa que procese un archivo de imagen png y un archivo de texto txt con un mensaje, y oculte el mensaje dentro de la imagen a través de la técnica de esteganografía LSB

2. *Análisis del problema*:
   Se requiere indicar la opción a elegir: ocultar o revelar. Posteriormente las instrucciones en consola nos guían para ingresar los nombres de archivos y su extensión dependiendo de la opción seleccionada entre las siguientes opciones:

   Caso 1. Ocultar: Indicar los nombres de los archivos de imagen y texto, en formato png y txt respectivamente. Los cuales deben estar dentro de la carpeta /data. Y ahí mismo especificar el nombre del archivo de imagen de salida, el cuál se guardará en la carpeta /out

   Caso 2. Revelar: Indicar el nombres de los archivos de imagen con el texto oculto, la cual debe estar en la carpeta /out y especificar el nombre de archivo txt donde se guardará el mensaje revelado

   Estas instrucciones en consola como vimos, depende del método usado (ocultar o revelar) y por lo tanto las puse dentro del mismo método con el fin de usar ahí mismo las variables locales que guardan los archivos de entrada. Puse valores por defecto para los nombres de archivos, lo cual simplifica la ejecución en caso de que se quieran usar los archivos por defecto que adjunte.

   Si se quiere usar archivos diferentes, se pueden usar sus nombres o se pueden renombrar utilizando los nombres por defecto:

   original.png Para la imagen original donde se ocultara el texto
   input.txt Para el texto donde viene el mensaje a ocultar

   Se utilizó la clase Image de la biblioteca PIL para manejar archivos de imagen y poder manipularlas a nivel de píxeles, así como crear nuevos archivos de imagen a partir de modificar uno original.

   De forma general en los métodos de ocultar y revelar se hizo lo siguiente :

   Ocultar:

   Convertimos el texto original a su representación en binario a 8 bits, una cadena sin espacios de 0’s y 1’s, y le ponemos al final un delimitador 1111111111111110 el cual

nos será útil cuando estamos decodificando, pues nos indica cuando hemos terminado de procesar el texto original codificado.

Luego iteramos sobre cada bit de nuestro texto codificado y simultáneamente sobre cada píxel de la imagen. Obtenemos sus valores RGBA y sobre cada componente, este valor entero (R,G,B o A) lo convertimos a su representación en binario y el bit más a la derecha lo reemplazamos por el bit del texto en la iteración en curso.

Este número binario modificado en su bit más a la derecha, lo volvemos a convertir a entero y sobreescribimos el valor original entero del componente.

Cuando terminamos de iterar sobre los 4 componentes del píxel actual, este píxel lo sobreescribimos en la imagen original con estos nuevos valores de cada componente actualizados.

Esta escritura sobre los píxeles termina cuando hemos terminado de iterar sobre el texto original convertido a su representación binaria, finalmente esta imagen modificada la guardamos en el archivo de imagen de salida especificado.

Revelar:

Abrimos la imagen y vamos iterando sobre cada pixel, obtenemos su valor cada componente en entero, lo transformamos a su representación binaria y su extraemos su último bit, el cuál guardamos en una cadena que contiene los LSB's leídos.

Tenemos que hacer la extracción de los bits más a la derecha en la misma dirección de lectura de pixeles que cuando ocultamos y también siguiendo el orden RGBA, pues es de la forma en la que se hizo la codificación de lo contrario el mensaje, aunque sea un mensaje con números de 8 bits puede ser diferente al original.

Cuando hayamos extraído al menos 16 bits, tras cada procesamiento de cada píxel verificamos si los últimos 16 bits extraídos corresponden al delimitador 1111111111111110, si es así terminamos en ese punto la lectura de pixeles.

Finalmente sobre la cadena de bits extraída, cuya longitud es múltiplo de 8, vamos procesando por bloques de 8 caracteres y transformando estas cadenas de 8 bits en su entero correspondiente, este entero lo transformamos a el carácter correspondiente en ASCII y lo concatenamos a una cadena, que corresponde al texto revelado, el cual escribimos en el archivo de texto de salida especificado.

3. *Selección de la mejor alternativa:*

Se eligió la biblioteca PIL porque posee un fácil manejo de archivos de imagen, pudiendo iterar sobre cada pixel como si fuera una matriz y además tiene métodos para obtener sus componentes de forma sencilla y la sobreescritura se hace como si se cambiara un valor en cierta posición (i,j) de una matriz.

La manipulación de las representaciones binarias las hice con cadenas, y así al

guardar el último bit solo creo una nueva cadena con el último carácter modificado, ya que en Python no existe la modificación de un objeto String.

4. *Mantenimiento futuro*:

   Sería conveniente manejar excepciones en las partes donde se hace escritura y lectura de archivos, pues en esta versión al seguir los pasos correctamente y por ejemplo, escribir un nombre de archivo inexistente el programa interrumpe su ejecución.

   También añadiría tests más exhaustivos para cada método en los que se hacen conversión de variables, en caso de que se quieran usar para extender la funcionalidad de programa, y en caso de que alguien quiera hacer debugging al implementar una nueva funcionalidad recurra a los resultados de las pruebas de cada método. Como yo utilicé estos métodos en lugares donde tenía claro la entrada y la salida, además de que es un script muy pequeño, no me representaba algún beneficio añadir tests, pero en caso de extender el código y tener que reusarlo si es necesario.