PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 - Programación Avanzada
1^{er} semestre 2017

Actividad 12

Manejo de strings, bytes y bytearrays

Introducción

¡Se filtró un archivo con información de conversaciones de los ayudantes! Sin embargo, este archivo está corrupto con un mecanismo para evitar el acceso a ellas. Desesperado por saber si estaría la solución de la siguiente tarea, decides aplicar lo aprendido en clase y arreglar el archivo.

Instrucciones

Junto con el enunciado, podrás encontrar un archivo con el nombre `chatayudantes.iic2233`. Este contiene la información de dos archivos: un `.gif` y un `.wav`. Para obtener los archivos originales, un misterioso hombre le deja una nota con instrucciones.

Paso 1: Arreglando los bytes

- Cada byte del archivo original resulta de la suma de 4 bytes consecutivos en el archivo corrupto. Para recuperar esta información, debes realizar esta operación comenzando con el primer `byte`, como se explica a continuación:

  Supongamos que el primer byte del archivo corrupto es 250, el segundo es 123, el tercero es 12 y el cuarto es 0. Entonces, el número que representa al primer byte original será 250+123+12+0 = 385. Luego, el número que representa al segundo byte original es la suma de los siguientes 4 bytes; y así sucesivamente, hasta completar el proceso.

  Nota: No te preocupes si el valor de cada número en este paso es mayor a 255, ya que en los siguientes pasos debes obtener los valores de cada byte como corresponden.

- Una vez realizado el paso anterior, deberás tomar cada número obtenido y reemplazar cada dígito de ese número por otro, como se indica a continuación.

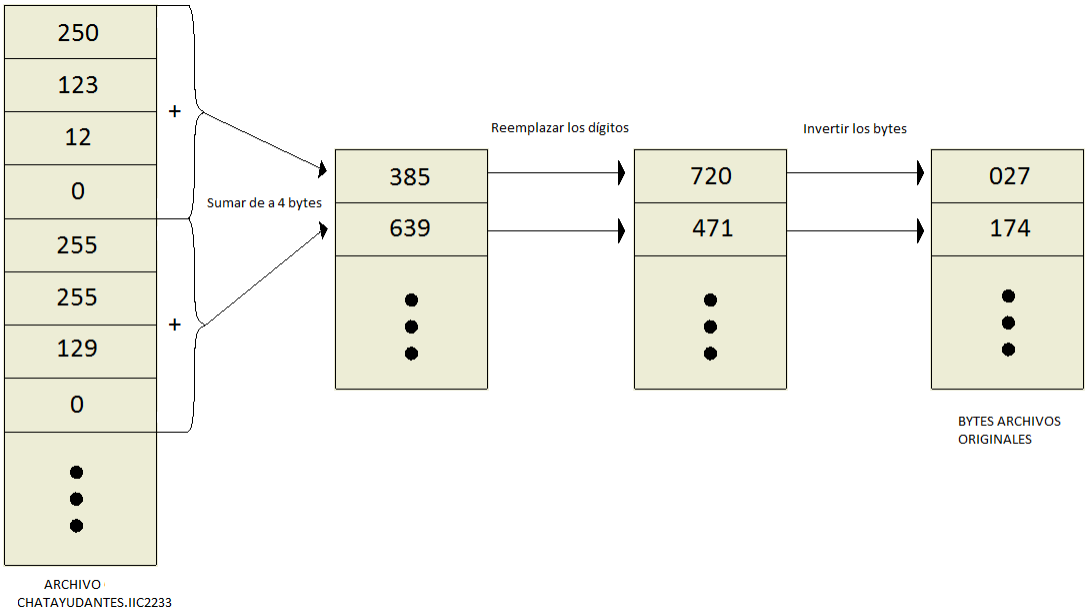
  $$1 \leftrightarrows 9 \ ; \ 2 \leftrightarrows 8 \ ; \ 3 \leftrightarrows 7 \ ; \ 4 \leftrightarrows 6 \ ; \ 5 \leftrightarrows 0$$

  OJO: Todos los números deben ser analizados como números de tres dígitos. Pueden asumir que no habran numeros de cuatro digitos. Por ejemplo, si el resultado de la suma del paso anterior fuera 8, debemos procesar este número como 008, y realizando los cambios de número, este quedaría como 552 (al cambiar 0 por 5 y 8 por 2).

- Luego, cada número obtenido en el paso anterior debe ser invertido.

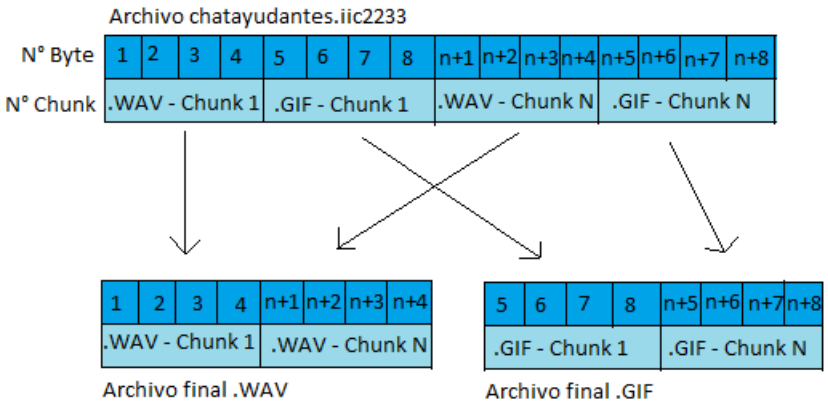
  Por ejemplo, el número 552 se convierte en el número 255, y el número 1 se convierte en el número 100 (pues 1 debe ser considerado como 001).

El siguiente esquema explica gráficamente el proceso del paso 1:



Paso 2: Separando los bytes

- Los bytes obtenidos del paso anterior se encuentran estructurados de la siguiente manera:



  El tamaño de cada chunk depende del archivo final al que pertenece. Estos se detallan a continuación.

- Chunks del archivo de audio:

  - El tamaño de cada chunk se corresponde con su número primo. Un número primo es un número natural mayor que 1 que tiene únicamente dos divisores distintos: él mismo y el 1. Por ejemplo el tamaño del primer chunk será 2; el segundo 3; el tercero 5; y así sucesivamente.

- El tamaño final del archivo `.wav` es de 9783 bytes.

  OJO: El tamaño del último chunk del archivo de audio no necesariamente concuerda con el número primo que le corresponde. Es importante que el tamaño de este archivo sea exactamente igual a 9783. Si no es así, los archivos podrían contener errores.

- Chunks del archivo de imagen:
  - El tamaño de cada chunk se corresponde con su número malvado. Un número malvado es todo número natural cuya expresión en base binaria contiene una cantidad par de unos (12 y 15 son números malvados ya que 12 = b1100 y 15 = b1111). Así, el tamaño del primer chunk será 3 (3 = b11); el segundo 5 (5 = b101); el tercero 6 (6 = b110); y así sucesivamente.
  - Como este archivo es más grande que el de audio, una vez que los bytes del archivo de audio llegan al tamaño final, todo el resto de bytes que quedan corresponden al chunk final del archivo de imagen.

Paso 3: Escribiendo la información

Una vez obtenidos los bytes/bytearrays que componen el archivo de imagen y el archivo de audio corregido, es necesario que sean escritos mediante **buffering**. Esto quiere decir que la escritura se realice por medio de transportes de información de tamaño fijo, evitando escribir la información por completo de una sola vez. Este método de lectura/escritura es bastante útil cuando se trata de información de gran tamaño. Su aplicación de Python deberá simular esta técnica escribiendo en los archivos mediante bytes/bytearrays de tamaño:

- **512** en el caso de archivo de audio
- **1024** en el caso de archivo de imagen

Es necesario que en cada iteración de escritura se haga saber al usuario el estado del proceso. Para esto se le pide que imprima una tabla de seguimiento similar a la siguiente:

- `TOTAL` corresponde a la cantidad de bytes que deben ser escritos.
- `PROCESADO` corresponde a la cantidad de bytes que se han conseguido escribir hasta ahora.
- `SIN PROCESAR` corresponde a la cantidad de bytes que aun no han sido escritos.
- `DELTATIME` corresponde a la diferencia de tiempo entre que comenzó la escritura y el tiempo actual.

Es necesario hacer dos tablas distintas, una para el `.gif` y otro para el `.wav`.

Requerimientos

- (2.00 pts) Arreglar bytes.
- (2.00 pts) Ordenar bytes.
- (1.00 pts) Escritura mediante buffering.
- (1.00 pts) Conseguir los archivos finales.

Notas

- Las librerías `date` y `datetime` pueden ser utilizadas para calcular el tiempo que tarda en realizarse el tiempo de escritura.
- Procure ser cuidadoso siguiendo las estructuras señaladas y en su desencriptación: cualquier error, por mas pequeño que sea, no permitirá recuperar los archivos originales.
- El nombre de los archivos finales queda a su criterio, pero debe respetar el tipo de archivo (`.gif` y `.wav`).

Entrega

- **Lugar:** GIT - Carpeta: Actividades/AC12
- **Hora:** 16:55