udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA Y CIENCIAS

Universidad Diego Portales
Escuela de Informática y Telecomunicaciones
Segundo semestre 2021

Laboratorio 4

Esteban Alarcón Osorio

Índice

1. Introducción

En el presente informe se entregan los resultados y análisis realizados luego de la construcción de un algoritmo de Hash solicitado por el gremio de los dueños de los sitios de contraseñas expuestas. Además, dicho algoritmo se presentará de forma modular y granular.

2. Algoritmo propuesto

El algoritmo está desarrollado en el lenguaje de programación Python, específicamente se utilizó la versión 3.9. Dicho algoritmo está compuesto por dos archivos principales, en primera instancia está el archivo *hash.py*, el cual utiliza las funciones del archivo *functions.py*. Ambos archivos se explican a continuación.

2.1. Menú principal

```python
if __name__ == '__main__':
    comprobar = comprobacion(sys.argv)
    flag = True
    if(comprobar == 'Error'):
        print('Error!! Debe ingresar un texto o archivo de texto')
        flag = False

    #Ciclo principal para mantener en ejecución el menú
    while(flag):
        print('\nQuede desea hacer con el texto/archivo de entrada?\n')
        print('1) Realizar Hash\n')
        print('2) Calcular entropia\n')
        print('3) Salir\n')
        opcion = input('Ingrese opcion:\n')
        if(opcion == '1'):
            if(comprobar == False):
                #Realización del proceso de Hash para el texto ingresado
                print(hash(sys.argv[1], 'texto'))

            elif(comprobar == True):
                #Se imprime resultado por resultado en consola correspondientes a cada linea del archivo
                for i in hash(sys.argv[1], 'archivo'):
                    print(i)
            flag = False
        #Presentación de resultados de la forma Entropia | Texto entrada, por consola (STDOUT)
        elif(opcion == '2'):
            if(comprobar == False):
                print ("{:<10} {:<1} {:<10}".format('Entropia','|','Texto entrada'))
                print ("{:<10} {:<1} {:<10}".format(entropia(sys.argv[1]),'|', sys.argv[1]))

            elif(comprobar == True):
                #Se enlista cada linea para luego calcular la entropía de cada una de ellas.
                with open(sys.argv[1], "r", errors= 'ignore') as archivo:
                    lista = [linea.rstrip() for linea in archivo]
                print ("{:<10} {:<1} {:<10}".format('Entropia','|', 'Texto entrada'))
                for texto in lista:
                    print ("{:<10} {:<1} {:<10}".format(entropia(texto), '|', texto))
            flag = False
        elif(opcion == '3'):
            flag = False

        else:
            print('Debe ingresar una opcion valida\n')
```

Figura 1: Vista general - Menu principal

De forma genérica, este algoritmo realiza la función de menú principal del programa, previo a cualquier selección de opción a través de dicho menú, el input ingresado al ejecutar el programa (recordar que el

gremio solicita ingresar la entrada por STDIN) es llevado a una comprobación de tipo, la cual se explicará más adelante, esto se observa en la segunda línea que se puede apreciar en la figura 1. Luego, con la entrada ya validada, se pasa a realizar la tarea correspondiente a la opción ingresada. Todas las tareas se explican a continuación.

2.1.1. Opción 1 - Realizar hash

Para comenzar con el hash, primero se corrobora el tipo de dato ingresado, es decir, dada la comprobación antes mencionada se ve si la entrada es un archivo o un string, esto depende del valor de la variable *comprobar*, si es *False*, esto significa que se trata de un string, en cambio, con un valor *True* en la variable, estamos frente un archivo. Al tratarse de un string, el algoritmo se ejecutará una sola vez para realizar el hash correspondiente, en cambio, con una entrada del tipo archivo, el algoritmo de hash se ejecutará tantas veces como líneas de texto tenga dicho archivo, entregando un resultado por cada una de estas. Lo anterior, se puede apreciar en la siguiente figura.

```
if(opcion == '1'):
    if(comprobar == False):
        #Realización del proceso de Hash para el texto ingresado
        print(hash(sys.argv[1], 'texto'))

    elif(comprobar == True):
        #Se imprime resultado por resultado en consola correspondientes a cada linea del archivo
        for i in hash(sys.argv[1], 'archivo'):
            print(i)
    flag = False
```

Figura 2: Menú - Opción 1

2.1.2. Opción 2 - Calcular entropía

Para el cálculo de entropía se realiza la misma comprobación de entrada mencionada anteriormente. La lógica general se presenta a continuación.

```
elif(opcion == '2'):
    if(comprobar == False):
        print ("{:<10} {:<1} {:<10}".format('Entropia','|','Texto entrada'))
        print ("{:<10} {:<1} {:<10}".format(entropia(sys.argv[1]),'|', sys.argv[1]))

    elif(comprobar == True):
        #Se enlista cada linea para luego calcular la entropía de cada una de ellas.
        with open(sys.argv[1], "r", errors= 'ignore') as archivo:
            lista = [linea.rstrip() for linea in archivo]
        print ("{:<10} {:<1} {:<10}".format('Entropia','|', 'Texto entrada'))
        for texto in lista:
            print ("{:<10} {:<1} {:<10}".format(entropia(texto), '|', texto))
    flag = False
```

Figura 3: Menú - Opción 2

A rasgos generales, se utiliza la función *format()* (perteneciente a las librerias de Python) para entregar los resultados de entropía de forma ordenada y clara. Por otro lado, cuando se trata de una entrada de texto plano o string, se realiza el cálculo de entropía de forma única, a través de la función *entropia()*.

En cambio, al tratarse de un archivo (sección *elif* del código), se recorre el archivo en cuestión línea por línea para así almacenar cada una de las entradas en una estructura, en este caso una lista. Luego de tener la lista creada en su totalidad, se ingresan uno por uno los elementos a la función *entropia()* para así, presentar cada resultado por STDOUT.

2.1.3. Opción 3 - Salir

Como se aprecia en la figura 1, el menú principal se encuentra en un loop *while*, esto se hace con el fin de mantener en ejecución el algoritmo en todo momento, hasta que se seleccione una de las opciones. Al seleccionar la tercera opción, la variable encargada de mantener el loop ejecutándose se cambia a un valor *False*, para así terminar con la ejecución y por ende finalizar el programa.

```
elif(opcion == '3'):
    flag = False
```

Figura 4: Menú - Opción 3

2.2. Funciones

Como se ha mencionado anteriormente, desde el menú principal se realizan llamadas a distintas funciones del archivo *functions.py*. Para hacer un recorrido ordenado por dicho archivo, a continuación, se presentan las funciones utilizadas según orden de ejecución. En primera instancia se presenta la función correspondiente a la comprobación de tipo. Luego se muestran las funciones de realización del hash, para luego seguir con la función para el cálculo de entropía.

2.2.1. comprobacion(*entrada*)

En primera instancia, y como se menciona en la explicación general del menú principal, las entradas recibidas pasan por la función de comprobación. Dicha función tiene la siguiente estructura:

```
#Funcion para verificar el tipo de entrada recibida
#Luego de verificar se retorna el tipo de entrada (archivo o texto), en caso de ser vacia retorna el string 'Error'
def comprobacion(entrada):

    if(len(entrada) < 2 | len(entrada) > 2):
        return 'Error'

    else:
        try:
            with open(entrada[1], 'r') as f:
                return True
        except FileNotFoundError as e:
            return False
        except IOError as e:
            return False
```

Figura 5: Función de comprobación

En primera instancia se comprueba de que el largo de entrada tenga un valor de 2, donde los elementos correspondientes son *hash.py* y *"entrada"* debido al funcionamiento de la librería *sys* de Python. Al poseer

un largo correcto, es decir, un solo elemento de entrada, se intenta abrir dicho argumento como un archivo, en este punto se pueden dar dos posibles resultados, al lograr abrir la entrada como archivo se retorna el valor *True*, en caso contrario, al tratarse de una entrada de tipo string se retorna el valor *False*.

2.2.2. Hash - hash(*argumento, texto*)

Continuando con la ejecución y como se muestra en la figura 2, al seleccionar la primera opción el algoritmo ejecuta la función hash que se muestra a continuación.

```python
#Funcion inicial del Hash, en caso de ser un archivo, se hace la llamada a adaptacion linea por linea
#Se utiliza 'errors = ignore' para que en caso de recibir una linea de texto vacia en el archivo (largo 0) se continue la ejecución
#En los archivos, los resultados se agregan a una lista.
def hash(argumento, tipo):

    if(tipo == 'archivo'):
        passw = []
        with open(argumento, "r", errors= 'ignore') as archivo:
            lista = [linea.rstrip() for linea in archivo]
        for pw in lista:
            passw.append(adaptacion(pw))
        return passw
    elif(tipo == 'texto'):
        return adaptacion(argumento)
```

Figura 6: Función inicial hash

Esta función recibe como parámetros la entrada ingresada por el usuario (argumento) y el tipo de dato al cual este corresponde (tipo). Luego de ingresar a la función se verifica según el tipo, en caso de ser un archivo, este se recorre para almacenar cada línea en una lista y así enviar uno a uno de los elementos a la función *adaptación()*, los resultados obtenidos al pasar por la función se almacenan en una lista diferente para luego ser enviada como retorno. En el caso de que el tipo sea un texto, solamente se envía dicha entrada a la función antes mencionada.

2.2.3. Hash - adaptacion(*texto*)

En este punto del algoritmo, el parámetro ha recibido siempre es una cadena de texto, por lo que se procede a adaptar dicha cadena al largo correspondiente al hash. En este caso se utiliza un largo total de 25, pero si es que se desea aumentar es totalmente posible. Como es costumbre, la lógica de la función se presenta en la siguiente figura.

```python
#Función encargada de adaptar el texto de entrada según el largo correspondiente al hash
#En caso de ser menor al valor se agregan valores siguiendo la operacion:
#(valor ordinal(ASCII) del caracter actual en el ciclo + valor ordinal del primer valor de la lista)/2
#Al valor anterior se le extrae solamente el entero correspondiente.
#Por otro lado, en caso de ser mayor al largo del hash se elimina el primer valor y el ultimo de forma alternada.
#Luego de tener el largo necesario se llama a la funcion operaciones()
def adaptacion(texto):
    caracteres = []
    if  (len(texto) == 0):
        return 'Linea vacía'

    for letra in texto:
        caracteres.append(ord(letra))

    #Se realiza un cambio en el primer valor para no obtener como resultado un patrón para la primera letra
    caracteres[0] = caracteres[0] + caracteres[-1]
    if (len(caracteres) < 25):
        #aumentar cantidad de caracteres
        for i in range(0, 25-len(caracteres)):
            add = int((caracteres[i] + caracteres[0])/2)
            caracteres.append(add)
    elif(len(caracteres) > 25):
        #disminuir la cantidad de caracteres
        for i in range(0, len(caracteres)-25):
            if(i%2 == 0):
                caracteres.pop(0)
            else:
                caracteres.pop(-1)

    resultado = operaciones(caracteres)
    return resultado
```

Figura 7: Función adaptación

En primera instancia, al ingresar a la función, se verifica que no existan líneas en blanco como parámetro, si es el caso se envía un mensaje de aviso como retorno. Al estar seguros de que el parámetro tiene largo, se ingresa letra a letra a la lista ”caracteres”, pero en este caso cada una es ingresada según su valor numérico en ASCII. Para evitar patrones en la primera letra, como por ejemplo con las palabras test y texto, se realiza un cambio de este primer valor de la lista sumando los valores numéricos del elemento en la posición cero y en la última posición. Continuando, se hace la verificación de largo total, si es menor que 25, agregamos elementos a la lista siguiendo la operación de suma entre el elemento que se está recorriendo, con el primer elemento (el cual se cambió anteriormente), esta suma resultante se divide en 2 para generar una especie de promedio y siempre se obtiene solo la parte entera. Por otro lado, si el largo máximo del parámetro ingresado es mayor a 25, se comienzan a eliminar valores tanto del comienzo como del final dependiendo si el índice es par o impar. Para finalizar, se crea la variable ”resultado” la cual almacenará el retorno de la función. *operaciones()*.

2.2.4. Hash - operaciones(*lista*)

Esta función, la cual es el último paso del hash, recibe como parámetro la lista creada anteriormente de nombre ”caracteres”. Al ingresar a la función, en primer lugar, se crea un string vacío. Luego de eso, se comienza un ciclo *for* el cual tendrá la función de realizar las operaciones 3 veces, para así dar mayor cambio al resultado del hash. Ya dentro de este ciclo, se realiza un nuevo ciclo para recorrer la lista ingresada en parámetros, con esto cada elemento numérico (el cual corresponde a un carácter) pasará por las operaciones de multiplicación, modulo y suma con números random dentro de un rango determinado. El rango se definió de forma completamente aleatoria, y en el caso de la función random, esta usa una

semilla la cual cambia en cada hora. Al terminar ambos ciclos mencionados, se procede a realizar el cambio de valor numérico a char según ASCII, donde si el valor obtenido es par, se aumenta en uno y luego se cambia, y en caso contrario se disminuye uno. Todo esto se hace para ir agregando los chars obtenidos a la variable "resultado", la cual es el string vacío mencionado anteriormente. Para finalizar, dicha variable se entrega como retorno de la función. Todo lo mencionado se puede apreciar en la figura de a continuación.

```python
#Funcion encargada de realizar el cambio según operaciones matemáticas
#En este caso se realizan multiplicaciones, funcion modulo y suma valor por valor
#Luego de realizadas las operaciones, según la posición del caracter en una lista, se le agrega o quita 1
#Para terminar se retorna el resultado como String
def operaciones(lista):
    resultado = ''
    for i in range(0,2):
        for j in range(len(lista)):
            lista[j] = lista[j]*random.randint(20, 25)
            lista[j] = lista[j]%random.randint(60, 65)
            lista[j] = lista[j]+random.randint(40, 45)

    for elemento in lista:
        if(elemento%2 == 0):
            resultado += chr(elemento+1)
        else:
            resultado += chr(elemento-1)

    return resultado
```

Figura 8: Función operaciones

2.2.5. entropia(*texto*)

Por último, y correspondiente a la opción 2 del menú (figura 3), el cálculo de entropía se realiza siguiendo la siguiente formula:

$$H = L * \log_2(W) \tag{1}$$

Donde:

- H: Entropía.
- L: Largo palabra.
- W: Base utilizada.

Para efectos del algoritmo, toda palabra ingresada se calcula con base 128, como se aprecia en la siguiente figura.

```python
def entropia(texto):
    base = 128 #correspondiente a la cantidad de caracteres ASCII
    entropia = len(texto)*math.log(base, 2) #H = L*Log2(W)
    return entropia
```

Figura 9: Función entropia

3. Análisis de resultados

Para evitar repetición de datos se hará referencia al archivo *"Comparativa.pdf"* disponible en el repositorio de GitHub: `https://github.com/EstebanAlarcon0/Actividad-4---Hash`

3.1. Rendimiento

En la comparativa de rendimiento, se puede observar que los 3 algoritmos conocidos (SHA1, SHA256 Y MD5) conservan una cierta cercanía entre sus resultados de tiempo, en cambio, el algoritmo desarrollado muestra un aumento de tiempo de ejecución conforme aumentan la cantidad de entradas, esto es debido a que como se ha visto a lo largo del informe, para realizar las operaciones de hash y recorrido de entradas se utilizan varios ciclos *for*, lo que aumenta el tiempo de ejecución entre más "distancia.º "largo"debe recorrer. Además, siguiendo con el énfasis en los ciclos, se observa que en la función *operaciones()* se realiza un ciclo dentro de otro, lo que a nivel de recorrido puede llegar a ser bastante costoso.

3.2. Entropía

En el caso de la entropía, se aprecia que los algoritmos utilizan la misma base (números + letras minúsculas), por lo que, su diferencia de entropía se debe solo por el largo del hash obtenido. Lo anterior se puede apreciar en la tabla, ya que, MD5 y SHA1 al tener el mismo largo en su hash resultante su entropía es la misma, pero como SHA256 dobla el largo de su hash, según la formula, también doblará la entropía de los algoritmos mencionados anteriormente. En cambio, el algoritmo de hash realizado tiene una base totalmente distinta, ya que al tener posibilidad de retornar cualquier valor de la tabla ASCII la base aumenta de 16 a 128, comparando con los algoritmos ya conocidos.
Para finalizar, si bien el resultado de entropía del algoritmo generado es mayor a los resultados de SHA1 y MD5, sigue siendo menor al resultado de SHA256, y para solventar esto se puede realizar un aumento en la cantidad de caracteres resultantes del hash, es decir, aumentar el largo del resultado para aumentar la entropía.