udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA Y CIENCIAS

Actividad 4 - Hash

Nombre: Nicolás Román
Profesor: Nicolás Boettcher
Ayudante: Víctor Manríquez

1. Funcionamiento programa

El programa fue desarrollado en el lenguaje de Python 3.9.8 y para ser ejecutado es necesario ejecutar el archivo Menu.py, en el cual se encuentran las opciones disponibles para realizar.

1.1. Menú

Al ejecutar este, se imprimirán una lista de opciones disponible.

1. Hash archivo: Si se desea hashear más de una linea de texto es posible hacerlo mediante esta opción, para esto es necesario ingresar el carácter 1 que corresponde a esta opción, posterior a esto se debe escribir el nombre del archivo, un ejemplo de esto es el archivo 50.txt utilizado para realizar la comparación.

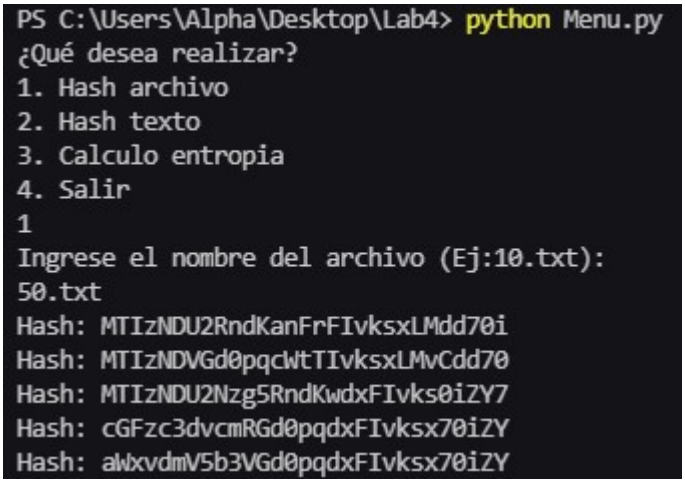

Figura 1: Ejemplo de la primera opción.

2. Hash texto: Para aplicar el algoritmo a solo una linea de texto es necesario ingresar el carácter 2 que corresponde a esta opción, posterior a esto se debe escribir el texto que se desea hashear.

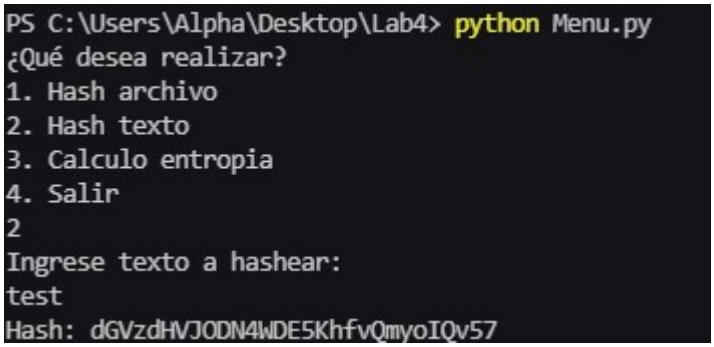

Figura 2: Ejemplo de la segunda opción.

3. Calculo entropía: Para calcular la entropía de un texto es necesario ingresar la tercera opción ingresando el carácter 3, posterior a esto es necesario ingresar el texto y se retornara un mensaje con el mismo texto y la entropía del mismo. Rara realizar este calculo se asume que se utilizara una base ASCII, por lo cual el valor esta definido en 128 en la función.

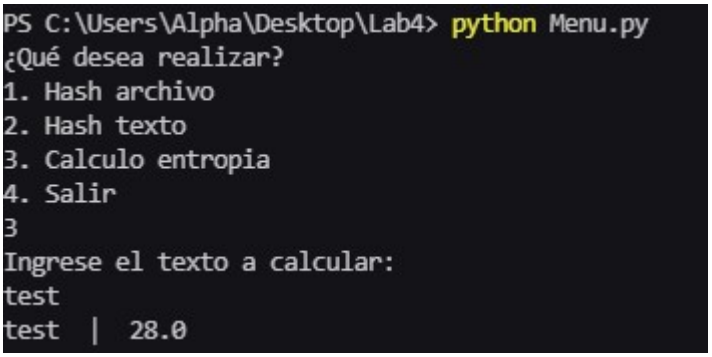

Figura 3: Ejemplo de la tercera opción.

4. Salir: Termina la ejecución del programa al ingresar el carácter 4 desde el menú.

```python
#Menu de navegación
if __name__ == "__main__":
    while True:
        print("¿Qué desea realizar?")
        print("1. Hash archivo")
        print("2. Hash texto")
        print("3. Calculo entropia")
        print("4. Salir")
        opcion = input()
        if opcion=="1":
            archivo=input("Ingrese el nombre del archivo (Ej:10.txt): \n")
            try:
                for line in fileinput.input(archivo):
                    transformacion(line.rstrip())
            except:
                print("Archivo invalido")
        elif opcion=="2":
            texto=input("Ingrese texto a hashear: \n")
            transformacion(texto)
        elif opcion=="3":
            # Formula:  H= Largo* log2 (Base), Para esto se definio la base correspondiente a todos los caracteres ASCII
            texto=input("Ingrese el texto a calcular: \n")
            entropia= len(texto)*math.log(128,2)
            print("Texto:",texto," |  Entropia:",entropia)
        elif opcion=="4":
            break
        else:
            print("Entrada invalida")
```

Figura 4: Código del menú implementado.

Luego de ejecutar cualquier opción se volverá al mismo menú hasta que se ingrese la opción de salir.

1.2. Adaptación del largo del texto ingresado

Antes de realizar el algoritmo de Hash implementado se adapto el texto según el largo, buscando llegar a los 25 caracteres en total para cualquier texto recibido, para esto se definieron 3 casos:

- Mayor a 25 de largo: En el caso de que el texto sea mayor a 25 caracteres, se intercalara entre un carácter del texto ingresado y un carácter aleatorio (minúscula, mayúscula o número) hasta llegar a los 25 caracteres. Para terminar las iteraciones correctamente, fue necesario realizar una condición donde cuando se lleguen a los 24 caracteres se agregue un ultimo carácter y se salga del for implementado. Para agregar los caracteres aleatorios se genero un texto con 13 caracteres con una semilla donde se utiliza el minuto de ejecución del programa y por cada iteración se desplaza sobre este texto generado para cambiar el carácter ingresado.

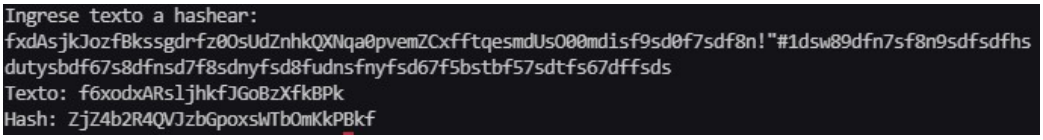

Figura 5: Ejemplo de un texto con más de 25 caracteres.

- Menor a 25 de largo: En el caso en que el texto sea menor a 25 caracteres de largo, se añadirán caracteres aleatorios (minúsculas, mayúsculas o números) hasta llegar a los 25 caracteres, para poder replicar estos caracteres previo a esto se definió una semilla con el minuto de ejecución del programa.

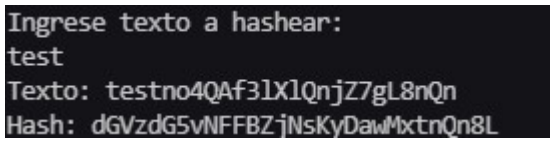

Figura 6: Ejemplo de un texto con menos de 25 caracteres.

- 25 de largo: Si el texto ingresado cuenta con 25 caracteres este sera hasheado directamente, sin ningún tipo de modificación.

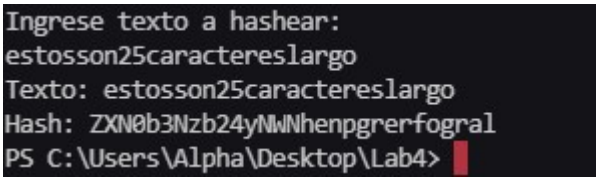

Figura 7: Ejemplo de un texto con 25 caracteres.

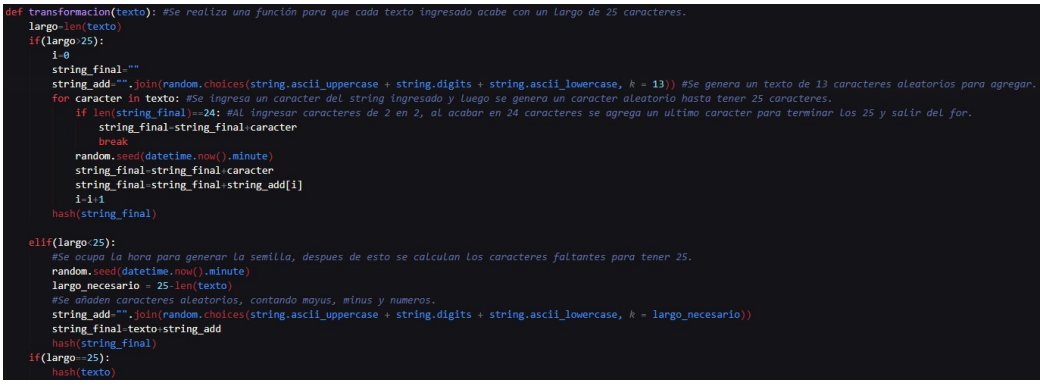

Figura 8: Código de las adaptaciones de largo implementadas.

Una vez adaptado el largo al solicitado todas las opciones pasan a hashear el nuevo texto generado.

1.3. Hash

Para realizar el hash al texto recibido de 25 caracteres de largo se aplicaron 3 algoritmos de criptografia vistos en clase.

- Base64: Los primeros 12 caracteres son cifrados utilizando base64, para esto se utilizo la librería de Python llamada base64. Mientras se itera sobre cada carácter en el string entregado estos se almacenan en otro string denominado texto_1, cuando la iteración llega al carácter 12 se utilizara este string almacenado con los primeros 12 caracteres y se ocupara la librería mencionada para realizar el cifrado respectivo, almacenando el valor obtenido en un string llamado hash_final, este cifrado siempre retornara un total de 16 caracteres.

- Caesar: Para los caracteres desde la posición 12 a la posición 20 se realizo un desplazamiento Caesar, para obtener el valor del desplazamiento se ocupo una semilla con el minuto de ejecución del programa y se genero un numero entre 1 y 24. Luego de esto se utiliza la implementación obtenida de `https://www.tutorialspoint.com/cryptography_with_python/cryptography_with_python_caesar_cipher.htm`, donde se obtiene el valor en Unicode del carácter ingresado, para luego realizar el desplazamiento respectivo y almacenando directamente a hash_final el carácter obtenido desde el valor del Unicode generado.

- Reverse: Los últimos 5 caracteres son cifrados implementando Reverse, Para realizar la implementación se almacenaron los caracteres a invertir en texto_3 de la misma manera en la que se realizo en texto_1, al llegar al carácter 25 se itera sobre los ultimos 5 caracteres en un while, partiendo desde la ultima posición del texto de 5 caracteres hacia atrás, almacenado estos en un string llamado Reverse y agregándolo una vez finalizado a hash_final.

Finalmente, es enviado el string de hash_final y al texto al que se le implemento este hash.

```python
def hash(texto):
    contador = 0
    hash_final = ""
    texto_1=""
    texto_3 = ""
    Reverse=""
    for caracter in texto: #Se itera sobre la lista y se separa en tres grupos diferentes.
        contador=contador+1
        if contador<=12: #Se ocupa la libreria de base64 para realizar un cifrado en este lenguaje.
            texto_1=texto_1+caracter
            if contador==12: #Se ocupa esta condicional para ocupar solo los 12 primeros caracteres.
                Base64_Encoded=base64.b64encode(texto_1.encode())
                hash_final=hash_final+Base64_Encoded.decode()
        elif contador<=20: #Desde el caracter 12 al 20 se realizara un cifrado caesar con desplazamiento según el largo.
            #Se define la semilla, esta generara un numero dependiendo del largo del texto a hashear.
            #Este valor puede ser replicado solo conociendo el largo y usando la misma función.
            random.seed(len(texto))
            #Para ser utilizado en caesar se definira un numero entre 1 y 24
            desplazamiento=random.randint(1,24)
            if (caracter.isupper()):  #Implementación obtenida de https://www.tutorialspoint.com/cryptography_with_python/cryptography_with_python_caesar_cipher.htm
                hash_final=hash_final+(chr((ord(caracter) + desplazamiento-65) % 26 + 65))
            else:
                hash_final=hash_final+(chr((ord(caracter) + desplazamiento-97) % 26 + 97))
        elif contador<=25: #Reverse
            texto_3=texto_3+caracter #Se agrupan los ultimos caracteres para aplicar el reverse
            if contador==25:
                i = 4 #Se define el largo de los caracteres que se necesitan invertir (del caracter 20 al 25)
                while i>=0: #Se realiza un while que va disminuyendo, ingresando los ultimos caracteres primero.
                    Reverse = Reverse + texto_3[i]
                    i = i - 1
                hash_final=hash_final+Reverse
    print("Hash:",hash_final)
```

Figura 9: Código de la implementación del hash realizado.

2. Análisis comparación

Al visualizar los valores entregados, se pudo visualizar la poca viabilidad que tiene crear un algoritmo de Hashing propio. Esto se ve reflejado en los valores obtenidos al momento de realizar la comparación; donde se puede ver una gran diferencia tanto en temas de entropía como de tiempo.
En el caso de la entropía se puede visualizar que la mayoría de los otros algoritmos tienen una mayor entropía a la obtenida al crear un algoritmo, a pesar de que este permite la utilización de cualquier carácter ASCII debido al Reverse implementado al final del mismo, el largo del hash generado no se compara a los demás, lo cual demuestra que los demás algoritmos son mucho

más seguros que el algoritmo creado.
Por otra parte, los tiempos de ejecución son aproximadamente 64 más rápidos que el algoritmo creado, algo que es esencial debido a que al momento de procesar una enorme cantidad de información se visualizara una notable diferencia en los tiempos de ejecución. Esto es debido a lo bien optimizados que están los algoritmos de hash más conocidos y que la constante utilización de ciclos en la implementación realizada aumento en forma exponencial los tiempos de ejecución.

Algoritmo	Base	Largo	Entropia	1 Credencial	10 Credenciales	20 Credenciales	50 Credenciales
Creado	128	29	203	0.000266	0.002096	0.004684	0.009623
SHA256	36	64	330.9	1.39e-05	0.000193	0.000120	0.000150
SHA1	36	40	260.8	4.09e-06	0.000111	0.000107	0.000143
MD5	36	32	165.4	8.39e-06	0.000148	0.000118	0.000145

Figura 10: Valores obtenidos al realizar la comparación.