Autores e identificación

William Mendez – 202012662

Juliana Galeano – 202012128

Daniel Aguilera – 202010592

Boris N. Reyes R. – 202014743

PROYECTO, PARTE 3

Introducción

El propósito de este informe es mostrar la documentación del proyecto de curso en su segunda parte y brindar respuesta a los escenarios planteados en su enunciado. Se presentan tres componentes respectivos al algoritmo solución, análisis de complejidad tanto temporal como espacial, y la respuesta ante los escenarios de comprensión de problemas algorítmicos.

1. Algoritmo de solución

El algoritmo diseñado por el grupo se basa en recorrer la cadena de entrada de derecha a izquierda para revisar el orden en el que aparecen letras nuevas y la cantidad de veces que aparece cada letra, al tener este orden de letras nuevas en este sentido podemos conocer el orden en el que fueron eliminadas de la cadena original.

Luego recorremos las llaves del diccionario donde almacenamos las letras y la cantidad de veces que se repiten y calculamos la cantidad de veces que una letra debe estar en la palabra original siguiendo la regla de que si una letra se eliminó en la iteración n del cifrado y el total de veces que aparece en la cadena cifrada es m entonces esta debe estar m/n veces en la cadena original. A su vez en este ciclo almacenamos el tamaño total que debe tener la palabra original.

Finalmente, con el tamaño total calculado para la palabra original creamos un slice de la entrada y lo recorremos letra a letra para revisar que en este se encuentran las letras encontradas y que estas solo se repiten la cantidad de veces calculada en el ciclo anterior, y en caso de que no sea así marcamos la salida como “NO EXISTE”, de lo contrario se imprime el resultado de la palabra encontrada y las letras en el orden que se eliminaron.

E/S	Nombre	Tipo
Entrada	entrada	String
Salida	salida	String

	Descripción	Explicación
Precondición	Ninguna	No se pide una condición particular acerca de la entrada, esta puede ser cualquier string ya que no se piden explicitamente cadenas que tengan todos sus caracteres y estén propiamente encriptadas
Postcondición	Para que **salida** sea la entrada desencriptada y su orden de encriptación es necesario que esta cuente con la cadena inicial al inicio y concatenada a esta contenga sus proceso de encriptación letra por letra. Adicionalmente, para que **salida** sea "No existe" no se debe cumplir lo anterior	Cadena desencriptada y orden de encriptación en caso de que se pueda hacer este proceso o "No existe" en caso de que no sea posible desencriptar

2. Análisis de complejidades espacial y temporal

Tiempos:

- Asignación (c_1)
- Comparaciones (c_2)
- +,*,-,/ (c_3)

Método procesar( )

```python
def procesar(entrada):
    letras_repeticiones = {}

    for n in range(1, len(entrada)+1):
        actual = entrada[-n]
        if actual in letras_repeticiones:
            letras_repeticiones[actual] += 1
        else:
            letras_repeticiones[actual] = 1

    letras_cantidad = {}
    orden_eliminacion = ""
    cantidad_palabra = 0

    tamanio = len(letras_repeticiones)
    for letra in letras_repeticiones:
        veces = letras_repeticiones[letra]//(tamanio)
        tamanio -= 1
        letras_cantidad[letra] = veces
        cantidad_palabra += veces
        orden_eliminacion = letra + orden_eliminacion

    palabra_original = entrada[0:cantidad_palabra]
    funciona = True

    for i in range(cantidad_palabra):
        letra = palabra_original[i]
        if letras_cantidad[letra] > 0:
            letras_cantidad[letra] -= 1
        else:
            print("NO EXISTE")
            funciona = False
            break

    cifrada = cifrarCadena(palabra_original, list(orden_eliminacion))

    if funciona and cifrada == entrada:
        salida = palabra_original + " " + orden_eliminacion
        print(salida)
```

Se realiza el cálculo de la complejidad teniendo en cuenta los parámetros dados anteriormente, esta se calcula línea por línea para obtener el resultado esperado.

$$c1 + n(3c1 + c_{accessL} + c_{inD} + c2 + 2c_{accessD} + c3) + 4c1 + c_{len} + m(5c1 + 2c_{accessD} + 4c3) + 2c1 + s + i(3c1 + c_{accessS} + 2c_{accessD} + c2 + c3) + c2 + c1 + 2c3 + (m * i)$$

Dada la complejidad se procede a simplificar para encontrar la cota designada. Adicionalmente, se van a definir las constantes que fueron añadidas para su mejor entendimiento:

$$c_{accessL} = Acceso\ a\ una\ lista$$

$$c_{accessD} = Acceso\ a\ un\ diccionario$$

$$c_{inD} = Operación\ in\ sobre\ un\ diccionario$$

$$c_{len} = Operación\ len$$

$$c_{accessS} = Acceso\ a\ un\ string$$

$$s = Tamaño\ del\ slicing\ a\ realizar\ sobre\ el\ string$$

$$n = Tamaño\ de\ la\ entrada$$

$$i = Tamaño\ palabra\ original$$

$$m = Numero\ de\ letras\ presentes\ en\ la\ cadena$$

$$m * i = Complejidad\ del\ algoritmo\ de\ cifrado$$

Se resuelven y simplifican las constantes:

$$8c1 + c2 + 2c3 + n(3c1 + c_{accessL} + c_{inD} + c2 + 2c_{accessD} + c3) + c_{len} + m(5c1 + 2c_{accessD} + 4c3) + s + i(3c1 + c_{accessS} + 2c_{accessD} + c2 + c3) + (m * i)$$

Para la simplificación de nuestra ecuación hacemos uso de la sustitución de expresiones por sub k's para así resolver de forma más sencilla.

$$k_1 = 3c1 + c_{accessL} + c_{inD} + c2 + 2c_{accessD} + c3$$

$$k_2 = 5c1 + 2c_{accessD} + 4c3$$

$$k_3 = 3c1 + c_{accessS} + 2c_{accessD} + c2 + c3$$

$$k_4 = 8c1 + c2 + 2c3 + c_{len}$$

Dadas las nuevas k's, se simplifica a:

$$nk_1 + mk_2 + ik_3 + k_4 + s + (m * i)$$

Complejidad Temporal Ponderada:

Teniendo en cuenta que la complejidad depende de distintas variables y contamos con 3 ciclos diferentes, podemos asumir de forma segura que la complejidad es:

$$\boldsymbol{O(n + m + s + i(1 + m))}$$

Complejidad espacial:

A lo largo de la solución se usaron las siguientes estructuras: Diccionarios y Listas.
Teniendo esto en cuenta es posible calcular la complejidad espacial, ya que sabemos la complejidad espacial de cada estructura. En el orden dado tenemos, $O(n)$, $O(n)$. En conclusión, es posible afirmar que la complejidad espacial es $O(n)$ debido a las razones dadas.

3. Respuestas a los escenarios de comprensión de problemas algorítmicos.

3.1. Se le pide generar cadenas *t* para las cuales es imposible obtener *s* y la secuencia de remoción de caracteres

- Dado el escenario existen tres casos en los cuales es imposible desencriptar la cadena entregada como parámetro, estos principalmente se basan en errores causados a propósito en la forma en la que se encripta para generar cadenas t imposibles de desencriptar.

1. **Caso 1: Cadena desordenada luego de cadena original:**
   Este caso es posible analizarlo de con ayuda del ejemplo dado en el enunciado, este puede ser visto de así:

q	w	e	q	e	e	w	e	w

Como es visto, la cadena cifrada correctamente debería ser:

q	w	e	q	e	w	e	e	w

En el primer ejemplo la cadena se encuentra desordenada a la hora de su cifrado, dado esto, es imposible obtener la cadena correctamente dado que daría está en desorden.

2. **Caso 2: Caracteres no presentes en la cadena en el cifrado:**
   En este caso, dado a que se presentan caracteres no presentes en la cadena original, es imposible obtener la cadena original dado a los caracteres adicionales.

q	w	e	q	e	w	e	x	w

En este caso la x no está presente en la cadena original, por ende, es imposible descifrarla.

3. **Caso 3: Caracteres eliminados anteriormente:**
   En este caso, teniendo en cuenta la cadena usada anteriormente, se sigue el mismo procedimiento, teniendo en cuenta la construcción de este caso.

q	w	e	q	e	w	e	e	q	w

El carácter había sido eliminado en la primera encriptación y en la segunda se vuelve a agregar, por esto es imposible desencriptar dado a que existe un carácter eliminado anteriormente.

4. **Caso 1: Cadena aleatoria:**
   Este caso es posible analizarlo de con ayuda del ejemplo dado en el enunciado, este puede ser visto de así:

q	x	b	n	f	d	v	m	p

   Dada esta cadena, no existe una cadena inicial valida, por esto es imposible obtener algo, ya que no existe una cadena a descifrar en primer lugar.

3.2. Se añade un nuevo paso a las operaciones de encriptación así:
   a. Concatena a la derecha de la cadena *t* la cadena *s*.
   b. Selecciona un carácter arbitrario presente en *s* y se remueve todas sus ocurrencias. El carácter seleccionado debe estar presente en *s* al momento de realizar esta operación.
   c. Selecciona aleatoriamente un carácter no presente en *s* (en su versión inicial) y concatenarlo a la derecha de la cadena *t*.

   El nuevo reto es definir el orden de eliminación correcto e identificar los caracteres que no hacen parte de la palabra original ya que nosotros recorremos la cadena cifrada al revés para conocer el orden de eliminación de caracteres, sin embargo, los caracteres aleatorios generarán errores porque nuestro algoritmo los tomaría como caracteres eliminados y presentes en la cadena original, lo cual causaría problemas en el resto del algoritmo y llevaría obtendríamos resultados erróneos.

   En este caso habría que repensar y rehacer todo el algoritmo porque actualmente nuestro algoritmo no podría identificar las letras que no hacen parte de la palabra original.

4. Referencias

- https://stackoverflow.com/questions/37350450/why-is-a-list-access-o1-in-python
- https://stackoverflow.com/questions/1115313/cost-of-len-function
- https://stackoverflow.com/questions/13203601/big-o-of-list-slicing
- https://stackoverflow.com/questions/69640743/what-is-the-time-complexity-for-a-break-statement