Diseño y Análisis de Algoritmos: Proyecto - Parte 3

Camilo Morillo Cervantes (202015224)
Juan Sebastián Alegría (202011282)

Departamento de Ingeniería de Sistemas y Computación, Universidad de los Andes

3 de junio de 2022

I. Algoritmo de solución.

En primer lugar, se llama a la función decoder con cada una de las entradas para que se indique si no existe la solución o nos retorne la cadena original junto al orden de los caracteres que se eliminan. Para esto, se tiene el arreglo appear que almacenará cada caracter nuevo encontrado de un recorrido inverso que se realiza por la entrada y que luego se invierte para dar el orden con el que fueron eliminados durante el proceso de encriptación. Además, se guarda en la variable endIndex el índice del último nuevo caracter encontrado en el recorrido inverso para comenzar la prueba de cadenas de la función coder desde el siguiente índice al endIndex.

Luego de inicializar estas dos variables se llamará a la función coder para hallar la cadena original desde la encriptada que nos dan en la entrada. Esta función va llevando en cuenta una cadena original y su cadena encriptada que en un inicio estarán vacías. Luego, se comenzará un ciclo que se ejecutará mientras que la solución no concuerde con la entrada, para así ir probando con cadenas más largas cada vez. Si endIndex ha alcanzado el largo de la entrada se retornará "NO EXISTE".

En cada ciclo se inicializan las tres variables originalString, solution, y removeString con el valor de la subcadena de la entrada desde el caracter inicial hasta el último endIndex, para luego utilizar el proceso de encriptación descrito por Criptón. Así se añade a solution la cadena de removeString que elimina los caracteres del arreglo orden/appear sucesivamente hasta terminar todo el proceso. Finalmente, se compara si solution es igual a la entrada, y de ser así se retornará el valor que tiene originalString.

De este modo, luego de terminar la ejecución de coder, el programa retornará originalString junto a el orden de eliminación de caracteres descrito en appear. En caso de que no exista una solución, se indicará que su inexistencia desde la función coder.

II. Análisis de complejidades.

La función solución tiene la siguiente complejidad, siendo c_1 asignaciones, c_2 comparaciones, y c_3 operaciones:

```
def decoder(data):
    appear, endIndex = [], 0 # 2c1
    for i in range(len(data)-1, -1, -1): # n(2c1 + c2 + c3)
        c = data[i] # c1
        if c not in appear: # c2
            appear.append(c) # c3
            endIndex = i # c1
    appear.reverse() # n
    originalString = coder(data, appear, endIndex) # 2nc2 + 4nc1 + nc3 + n^2 + 2n^3 + 2n^2c1 + n^2c3
    if originalString == "NO_EXISTE": # c2
        return "NO_EXISTE"
    return originalString + "_" + "".join(appear) # n
```

Dada la complejidad de la función coder:

$2nc_2 + 4nc_1 + nc_3 + n^2 + 2n^3 + 2n^2c_1 + n^2c_3$

```
def coder(data, orden, endIndex):
    solution, originalString = "", "" # 2c1
    while solution != data: # nc2 + n(c2 + 4c1 + c3 + n + n(2n + 2c1 + c3))
        if endIndex == len(data): # c2
            return "NO_EXISTE"

        endIndex += 1 # c1 + c3
        originalString = solution = removeString = data[0:endIndex] # n + 3c1
        for i in orden: # n(2n + 2c1 + c3)
            removeString = removeString.replace(i, '') # 2n + c1
            solution += removeString # c1 + c3
    return originalString
```

Sumatoria de complejidades:

$T(n) = 2n^3 + n^2 + c_2 + 2nc_3 + n^2c_3 + 3nc_2 + 6nc_1 + 2n^2c_1 + 2n + 2c_1$

Por lo tanto, este algoritmo tiene una complejidad O(n^3) donde n es el largo de la cadena de entrada

Como está función de solución se ejecuta el número de casos de entrada, entonces la complejidad total del programa es $O(n_{casos} * n_{len(entrada)})$

III. Otros escenarios.

ESCENARIO 1: Se le pide generar cadenas t para las cuales es imposible obtener s y la secuencia de remoción de caracteres.

Para este escenario se pensaron varias alternativas como añadir caracteres en posiciones aleatorias a cadenas base o utilizar rangos aleatorios para actualizar cadenas y así encontrar las t objetivo, pero solían tener algunos casos margen de error y no ser soluciones totalmente correctas. Por esto, se pensó la solución descrita a continuación que aunque es bastante ineficiente es correcta.

Se utilizará exactamente las mismas funciones coder y decoder de la solución original, pero al llamarla en el método principal, no se utiliza como parámetro data la entrada que nos dan actualmente sino cadenas generadas en un ciclo. Esto, dado que, la función decoder estará dentro de un ciclo infinito que terminará cuando se encuentre la cantidad deseada de cadenas imposibles de obtener s y su secuencia de remoción.

Este ciclo estará generando todas las cadenas posibles de caracteres por fuerza bruta y llamará en cada iteración a la función solución decoder para así verificar si la cadena generada hace parte de las deseadas (decoder retorna NO EXISTE en estos casos). En caso de que sea una cadena deseada se añadirá a un arreglo y cuando el arreglo contenga el número de cadenas requeridas por el programa, lo retornará. Este retorno será un arreglo con todas las cadenas que cumplen la condición del escenario.

ESCENARIO 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.

Para añadir este paso adicional a la encriptación, se tiene el reto de que cuando se esté en la función coder se debe eliminar uno de los caracteres de la entrada antes de comenzar cada iteración. Si en alguna de las iteraciones se eliminó un caracter que no fue insertado originalmente en alguno de los pasos C, se deberá reiniciar toda la función coder actual eliminando caracteres en algún orden distinto. Por lo que se ejecutaria como máximo k! veces la función coder, siendo k el número de caracteres distintos en la entrada.

Si se aplica a la solución actual, se deberá llamar a una función que creará todos los ordenes de eliminación de caracteres posibles a partir de los caracteres de la entrada, esta función los colocará en un arreglo que contiene otros subarreglos con cada una de las permutaciones con repetición de la entrada.

Ahora se deberá agregar un nuevo parámetro a la función coder que será el orden de eliminación de caracteres aleatorios del paso C y, además, esta función se tiene que llamar ahora dentro de un ciclo que se ejecutará hasta que se hayan probado todas las permutaciones generadas por la nueva función explicada anteriormente o hasta que se encuentre una originalString que genere la entrada (esto se indicará a través del retorno de coder que ahora es una tupla que contiene originalString y solution).

Finalmente, se deberá remover los caracteres aleatorios encontrados de appear, para así retornar por separado la cadena original, el orden de eliminación de caracteres arbitrarios presentes del paso B, y el orden de eliminación de caracteres aleatorios no presentes del paso C.