Sufijo Balanceado

Claudia Alvarez Martínez
Roger Moreno Gutiérrez
Jan Carlos Pérez González

Cuarto año, Ciencias de la Computación.
Facultad de Matemática y Computación, Universidad de La Habana, Cuba

1 Problema:

Te dan una cadena S de longitud N y un número entero K.

Sea C el conjunto de todos los caracteres en S. La cadena S se llama *buena* si, para cada sufijo de S:

La diferencia entre las frecuencias de cualquier par de caracteres en C no excede K.

En particular, si el conjunto C tiene un solo elemento, la cadena S es buena.

Encuentra si existe una reordenación de S que sea buena. Si existen múltiples reordenaciones de este tipo, imprime la reordenación lexicográficamente más pequeña. Si no existe tal reordenación, imprime -1 en su lugar.

Nota que un sufijo de una cadena se obtiene eliminando algunos (posiblemente cero) caracteres desde el principio de la cadena. Por ejemplo, los sufijos de $S = abca$ son $\{a, ca, bca, abca\}$.

2 Nuestra redefinición del Problema:

Sea S una cadena de longitud n con un conjunto de caracteres C, donde cada caracter $c \in C$ aparece con una frecuencia $f_c(S)$.
Una cadena S' es un reordenamiento *bueno* de S si para todo sufijo T de S', y para todo par de caracteres $c_1, c_2 \in C$, se cumple la siguiente invariante:

$$|f_{c_1}(T) - f_{c_2}(T)| \leq k$$

donde $k \in \mathbb{Z}$ y $f_{c_i}(T)$ representa la frecuencia del caracter c_i en el sufijo T, con $i = 1, 2, \ldots, |C|$.

Se asume que las cadenas de entrada están formadas utilizando el alfabeto inglés Σ, que cumple: $|\Sigma| = 26$.

3 Soluciones implementadas:

Para la solución del problema veremos dos enfoques:

1. Fuerza bruta.
2. Estrategia greedy basado en la propiedad de subestructura óptima y el principio greedy, que garantiza que una solución globalmente óptima puede ser construida mediante una serie de elecciones localmente óptimas.

3.1 Un primer acercamiento: Fuerza bruta

En este enfoque, generamos todas las permutaciones de la cadena. Para cada permutación, verificamos que se cumpla la invariante antes mencionada en cada uno de los sufijos, y así determinar si la cadena califica como *buena*. Luego entre todas las posibles *cadenas buenas* devolvemos la cadena más pequeña lexicográficamente.

Análisis de Complejidad Temporal:

Dado que se generan todas las permutaciones de una cadena de longitud n se tiene una complejidad de $O(n!)$ ya que hay $n!$ permutaciones posibles para una cadena de longitud n.
Generar todas las combinaciones de 2 caracteres del conjunto de caracteres únicos de la cadena tiene una complejidad $O(c^2)$.
Luego, recorrer cada permutación para verificar si se cumple la invariante, implica que, para cada sufijo, en el peor caso, se realiza una verificación sobre todas las combinaciones de 2 caracteres, lo cual tiene una complejidad $O(n * c^2)$.

Teniendo en cuenta que estamos trabajando con el alfabeto inglés Σ, podemos afirmar que el valor de c es constante dado que $c \leq |\Sigma|$.Por tanto, la complejidad total del algoritmo es $O(n! * n * c^2) = O(n!)$.

Análisis de Complejidad Espacial:

Almacenar todas las permutaciones de la cadena y todas las combinaciones de 2 caracteres únicos tiene un costo espacial de $O(n!)$ y $O(c^2)$ respectivamente.
Además, mantener los diccionarios de frecuencias, tanto para la cadena de entrada como para las verificaciones de las frecuencias en los sufijos, requiere $O(c)$ en cada caso.

Al igual que en la complejidad temporal el valor de c es constante y por tanto, la complejidad espacial total es $O(n! + c^2 + 2c) = O(n!)$.

3.2 Estrategia Greedy para la construcción de la solución:

La estrategia de solución se basa en construir el reordenamiento lexicográfico más pequeño respetando la invariante de frecuencias bajo los principios fundamentales del enfoque greedy: **greedy-choice property** y **optimal substructure**.

Para ello veremos dos posibles formas de construcción para la cadena resultante.

3.2.1 De derecha a izquierda, colocando siempre el mayor caracter lexicográficamente posible y comprobando la invariante para todo par de caracteres:

- **Inicialización de frecuencias:** Dado el conjunto de caracteres C de la cadena S y sus respectivas frecuencias $f_c(S)$, nuestro objetivo es iterar sobre C para formar una nueva cadena S'.
- **Elección greedy:** En cada paso de la construcción de la cadena S', seleccionamos el caracter lexicográficamente más grande disponible, que no haya excedido su frecuencia original en S y que, al añadirse al sufijo en construcción, mantenga la invariante de frecuencias.
  Este es el enfoque greedy, ya que se toma la mejor decisión en cada paso sin preocuparse por los subproblemas futuros.
- **Subestructura óptima:** La solución parcial construida en cada paso sigue manteniendo la estructura óptima de los subproblemas, ya que en cada subproblema se resuelve la instancia más pequeña del problema utilizando una combinación del caracter seleccionado y el sufijo restante.
- **Iteración por caracteres:** El algoritmo procede de derecha a izquierda, asegurándose de que la elección localmente óptima (el caracter más grande disponible que respete la invariante) produzca un reordenamiento globalmente óptimo. Tras seleccionar el caracter, se reduce el problema a la construcción del sufijo restante de la cadena que cumpla la invariante.
- **Verificación de la invariante:** Antes de agregar el caracter seleccionado a S', se comprueba que su inclusión no viola la invariante de frecuencias. Si cumple con la condición, lo añadimos. Si no es posible, se pasa al siguiente caracter disponible. Este proceso garantiza que, al completar cada paso, las elecciones hechas hasta el momento respeten la condición impuesta por el problema.
- **Reinicio del proceso:** Tras agregar un caracter, repetimos el proceso volviendo al caracter lexicográficamente más grande disponible.
- **Condición de fallo:** Si en algún momento no es posible agregar ningún caracter sin violar la invariante, declaramos que no es posible construir un reordenamiento *bueno* para la cadena S.

Análisis de Correctitud:

- **Proposición:** La cadena S', construida bajo el algoritmo greedy propuesto, es el reordenamiento lexicográfico más pequeño de S que cumple la condición de ser *buena*.

  Demostración de minimalidad:
  Supongamos que existe otro reordenamiento S'' de S que es lexicográficamente menor que S'. Esto implica que en alguna posición i de S'', el caracter c_i en esa posición es menor que el caracter correspondiente en S', y los caracteres hasta la posición c_{i-1} en ambas cadenas son idénticos. Sin embargo, esto contradice el principio greedy debido a que el algoritmo siempre selecciona el caracter lexicográficamente más grande que no viole la invariante de frecuencias en el sufijo en construcción. Por lo tanto, la existencia de un caracter más pequeño lexicográficamente en S violaría dicha invariante en esa posición, lo que contradice la hipótesis de que S es un reordenamiento válido. Por tanto no puede existir un S'' menor que S'.

- **Proposición:** El reordenamiento S' cumple la condición de ser una *buena* cadena, es decir, para cada sufijo T de S', la diferencia entre las frecuencias de cualquier par de caracteres c_1 y c_2 es, como máximo, k.

  Invariante: Durante cada iteración del algoritmo, al agregar un caracter c a la cadena parcial S', se verifica que la diferencia de frecuencias con el resto de caracteres no exceda la diferencia máxima k. Esto garantiza que cada sufijo de la cadena generada hasta ese punto también cumple con la invariante.

  Demostración por inducción que, en todo momento, los sufijos de la cadena construida cumplen la condición de la diferencia de frecuencias:

  - **Caso base** (subestructura inicial)**:** Antes de agregar ningún caracter a la cadena S', la cadena vacía cumple trivialmente la condición de ser una solución óptima.
  - **Hipótesis de inducción:** Supongamos que después de añadir i caracteres, todos los sufijos de la cadena parcial S'_i cumplen la invariante.
  - **Paso inductivo** (propiedad de subestructura óptima)**:** Al añadir el caracter c en la (i+1)-ésima posición, el algoritmo hace la elección greedy del caracter más grande disponible que mantenga la invariante de frecuencias. Esta elección garantiza que la cadena resultante S'_{i+1} sea la solución lexicográficamente más pequeña para el subproblema que incluye este nuevo caracter. Dado que el nuevo sufijo generado después de añadir c también cumple la invariante, la subestructura resultante es óptima.

  Por tanto, por el principio de subestructura óptima, hemos demostrado que cada decisión greedy en cada paso asegura que la cadena parcial resultante es una solución óptima para el subproblema. Al finalizar la construcción de la cadena S', podemos garantizar que todos sus sufijos cumplirán la condición de diferencia máxima k.

- **Condición de terminación:** Si el algoritmo llega a un punto en el que no se puede agregar ningún caracter sin violar la invariante, esto significa que no existe

ningún reordenamiento *bueno* para la cadena S. En consecuencia, el algoritmo finaliza y declara que no es posible construir tal reordenamiento. Esto garantiza que el algoritmo siempre termina.

Análisis de Complejidad Temporal:

El algoritmo propuesto utiliza una estructura de datos que permite almacenar resultados previamente calculados, donde la inserción, eliminación y consulta se realizan en tiempo $O(1)$, pero para ello debemos realizar un análisis de la cadena de entrada S, lo cual tiene un costo asociado.
Recorrer la cadena S para almacenar los caracteres $c \in S$ con sus frecuencias de ocurrencia tiene costo $O(n)$, donde n esta dado por el tamaño de la cadena.
Tanto la ordenación de los caracteres lexicográficos como la combinación de todos los pares de caracteres tiene un costo de $O(clogc)$ y $O(c^2)$ respectivamente, donde c es la cantidad de caracteres distintos de S. Además, como necesitamos un estructura para llevar las frecuencias actuales de los caracteres en la cadena en construcción S', con su inicialización, estaríamos consumiendo un costo $O(c)$.
Dado que intentamos construir una cadena S' que sea un reordenamiento de S, el algoritmo realiza hasta un total de n iteraciones haciendo, por cada una, comparaciones con el resto de caracteres, teniendo esto una complejidad temporal $O((c-1)*n)$.
Por último, para devolver la cadena lexicográficamente más pequeña, dada la forma de construcción de S', necesitamos buscar su reverso, teniendo esto costo $O(n)$.

Sumando todas las complejidades, la complejidad temporal total del algoritmo es $O(n + c^2 + clogc + c + (c-1)*n + n)$ y dado que estamos trabajando con el alfabeto inglés Σ, podemos afirmar que el valor de c es constante dado que $c \leq \Sigma$. Por tanto la complejidad total del algoritmo es $O(n)$.

Análisis de Complejidad Espacial:

El espacio requerido para almacenar la frecuencia de cada caracter en la cadena de entrada es $O(c)$ y dado que para cada caracter se almacenan todas las combinaciones posibles con otros caracteres, se requiere, además, un espacio proporcional a $O(c^2)$.
Por otro lado, mantener un diccionario que almacena las frecuencias de los caracteres ordenados de manera descendente y otro para llevar el seguimiento de la frecuencia de los caracteres en la cadena en construcción, requiere $O(c)$ para cada uno.
Como la cadena resultante puede crecer hasta el tamaño de la cadena de entrada, esta cadena requiere un espacio $O(n)$.

Sumando todos estos componentes, la complejidad espacial total del algoritmo es $O(3c + c^2 + n)$. Sin embargo, dado que c es un valor constante y limitado ($c \leq |\Sigma|$), la complejidad espacial se puede simplificar a $O(n)$.

3.2.2 De izquierda a derecha, colocando siempre el menor caracter lexicográficamente posible de forma tal que la diferencia entre el caracter de mayor frecuencia y el de menor frecuencia no exceda k:

- **Condición inicial:** Para construir un reordenamiento *bueno*, es necesario que la cadena S satisfaga la condición $|f_{max} - f_{min}| \leq k$.
- **Proceso de construcción:** Para cada posición i del tamaño de la cadena, intentamos colocar el caracter lexicográficamente más pequeño que esté disponible, es decir, el de menor orden alfabético entre los que no hemos agotado su frecuencia. Al intentar colocar un caracter c, disminuimos temporalmente su frecuencia en 1 y verificamos si las frecuencias de los caracteres restantes siguen cumpliendo la condición inicial, o sea, que la diferencia entre la frecuencia máxima y mínima de los caracteres no exceda k. En caso de que la diferencia exceda k, probamos con el siguiente caracter lexicográficamente más pequeño y repetimos el proceso.
- **Balanceo del sufijo**: Este procedimiento garantiza que en cada paso el sufijo restante de la cadena sea balanceado, ya que estamos colocando el caracter lexicográficamente más pequeño que preserva la condición $|f_{max} - f_{min}| \leq k$ en el conjunto de caracteres restantes.

Análisis de Correctitud:

En cada paso i, el algoritmo selecciona el caracter c más pequeño lexicográficamente de los disponibles. Esta elección es localmente óptima porque, si es posible colocar el caracter más pequeño manteniendo la propiedad de balanceo de frecuencias, entonces hacerlo minimizará el lexicográfico del resultado en esa posición.

La propiedad greedy del problema se basa en la observación de que, en cada paso, la mejor elección (en términos de minimizar el orden lexicográfico) no depende de las decisiones futuras. Una vez que seleccionamos el caracter lexicográficamente más pequeño que mantiene la condición $|f_{max} - f_{min}| \leq k$, el resto de la cadena puede construirse de manera independiente de esta decisión. Es decir, seleccionar el caracter c en la posición i no afecta negativamente a la posibilidad de cumplir la condición anterior en las posiciones subsiguientes, siempre y cuando esa condición se cumpla para los caracteres restantes. Esta es una característica crucial de los algoritmos greedy: la subestructura óptima, donde la solución óptima del problema puede obtenerse combinando soluciones óptimas de subproblemas más pequeños (en este caso, los sufijos restantes).

Invariante: En cada paso del algoritmo, el sufijo restante de la cadena debe cumplir la propiedad de que la diferencia entre la frecuencia máxima y mínima de los caracteres restantes no exceda k.
Esta invariante se mantiene porque, al colocar un caracter c en la posición i,

verificamos que la propiedad se cumpla antes de hacer la inserción. Si la invariante se cumple después de colocar c, entonces el sufijo resultante sigue siendo válido y podemos proceder al siguiente paso sin arriesgar la corrección global de la solución.

Este algoritmo es óptimo porque cada decisión que toma —colocar el caracter más pequeño posible en cada posición— es la mejor opción local disponible y no compromete las decisiones futuras. Si alguna decisión local no fuera óptima (es decir, si colocáramos un caracter más grande en lugar del más pequeño posible), entonces el reordenamiento completo no sería el lexicográficamente más pequeño posible. Esto asegura que el algoritmo no solo genera una solución correcta, sino que es óptima en términos del criterio lexicográfico.

Análisis de Complejidad Temporal:

Recorrer la cadena S para almacenar los caracteres $c \in S$ con sus frecuencias de ocurrencia tiene costo $O(n)$, donde n esta dado por el tamaño de la cadena. La ordenación de los caracteres lexicográficos de manera ascendente tiene un costo de $O(clogc)$. Luego, para cada posición i, intentamos colocar hasta 26 caracteres (uno para cada letra del alfabeto). Verificamos si el sufijo restante es balanceado en $O(2c)$ ya que necesitamos obtener la frecuencia máxima y mínima, donde para el caso peor será $O(|\Sigma|)$ debido al tamaño del alfabeto, que en este caso es 26. Así, la complejidad total es $O(n + clogc + n * c^2) = O(n)$.

3.2.3 Análisis de Complejidad Espacial:

Con esta implementación logramos reducir en constante el tiempo espacial ya que solo requerimos $O(c)$ para almacenar las frecuencias de los caracteres. Luego, el diccionario que contiene las frecuencias de los caracteres ordenadas lexicográficamente, también ocupa $O(c)$, ya que solo contiene los caracteres distintos de la cadena de entrada S. La cadena resultante crece con cada iteración hasta alcanzar el tamaño de S, lo que requiere $O(n)$ espacio, donde n es el tamaño de S. En cada paso, el algoritmo necesita calcular la frecuencia máxima y mínima de los caracteres restantes. Este proceso requiere espacio adicional para almacenar las frecuencias y aunque el cálculo en sí toma tiempo $O(c)$, no afecta directamente la complejidad espacial.

La complejidad espacial es $O(n)$, ya que c es una constante limitada por el tamaño del alfabeto y nos permite simplificar en términos asintóticos.