Identificación:

Santiago Pardo - 202013024
Kevin Cohen - 202011864

Algoritmo de solución:

El problema C busca generar una super cadena que contenga todas las cadenas pasadas por parámetro y que la longitud de la cada fuera la mínima posible.

Como parámetros entra la cantidad de strings a analizar denominada n, el tamaño de los string denominado m, y los strings a analizar.

Para solucionar este algoritmo se hizo uso de una implementación de fuerza bruta, con la cual se hallaban todas las permutaciones de orden de las cadenas que entraban por parámetro, por lo que existían n! posibilidades de ordenar estas cadenas de caracteres. Para ello se utilizó el método recursivo AllRecursive, que su función es generar todas las permutaciones posibles de un arreglo, por lo que la ejecución termina cuando se generan todas las n! permutaciones, es importante resaltar que no guarda todas las permutaciones que genera, sino que cuando las genera sobre la misma estructura de datos crea la próxima permutación.

El método anteriormente mencionado, cada vez que genera una permutación invoca un método denominado mixString que recibe una lista de strings y los une de forma lineal, sobreponiendo los caracteres que tienen en común. Este método recorre toda la lista de strings hasta la posición n-2, ya que la posición n no es indexable y que el último string no tiene otro string después de el, en cada recorrido calcula cuantos caracteres se sobreponen con el string de la posición x del arreglo con el string de la posición x+1 del arreglo, con la función overlap, que recibe dos strings que hace un recorrido de m(el tamaño de los strings) a 0, para hallar cuantas i ultimas posiciones del primer string concuerdan con las i primeras posiciones del segundo string.

Seguidamente, con este resultado se concatena a la solución de mixString con el string generado por la concatenación del substring del primer string desde el carácter cero hasta el carácter m- el resultado de la función overlaps y el segundo string, este proceso se realiza hasta que se itere sobre toda la lista. Seguidamente, se compara si la longitud del string generado por la función mixstring es menor a la longitud del string que asumíamos que era el menor hasta el momento, si longitud es menor se guarda este string como respuesta hasta que encuentre un nuevo string de menor longitud o se acaben las permutaciones de las cadenas de caracteres.

Por otro lado, se podía realizar esta solución aplicando programación dinámica con la ecuación de recurrencia siguiente:

$$m(S,j)\begin{cases} |S| = 2 \rightarrow mixstrings(S_0, S_1) \\ |s| > 2 \rightarrow \min\left(\left\{S_{k\in S} \middle| m\left(\{S\} \setminus \{k\}\right\}, k\right)\right\} \end{cases}$$

Esta solución era más rápida ya que su complejidad es de Orden O($n^2 2^n$), pero tenía dos problemas, el primero es que la complejidad espacial es de orden O($n * 2^n$), por lo que ocupa demasiado espacio en memoria, y el otro es la implementación ya que hace uso de la estructura bitmap para representar todos los conjuntos posibles, por lo que la complejidad algorítmica era mucho mayor, por lo que nos decidimos por el de fuerza bruta, ya que este aunque sea más demorado en ejecución no ocupa tanta memoria, y la diferencia de velocidades no recompensaba el aumentar la complejidad algorítmica.

Análisis de Complejidades:

Con lo anteriormente mencionado podemos hacer un análisis de las complejidades espaciales y temporales. Para hallar la complejidad espacial revisamos las estructuras de datos usadas, en la solución presentada solo se utilizó un arreglo de n posiciones ya que sobre este mismo se generaban todas las permutaciones del arreglo de strings original.

Por otro lado, para encontrar la complejidad temporal se debe de remitir al método AllRecursive que lo que hace es generar todas las permutaciones de un arreglo, por lo que generar todas estas permutaciones, por la misma fórmula de permutación, si se tienen n elementos es de orden O(n!), pero por cada permutación se realiza el método mixString por lo que la complejidad de este se debe de multiplicar por n!.

La operación mixstring recorre sobre todo un arreglo de n strings y por cada recorrido se ejecuta la operación overlaps que hace un recorrido desde 0 hasta m(la longitud de cada cadena de caracteres), por lo que el método en si tiene una complejidad de orden O(nm).

Siguiendo lo anteriormente mencionado, por cada permutación generada se ejecuta un algoritmo de orden O(nm), por lo que al final la complejidad temporal del algoritmo final es de O(n!*n*m).

Comentarios finales

El algoritmo frente a la complejidad espacial, a comparación de las otras versiones, es mucho mejor, ya que solo ocupa un arreglo del tamaño n, sin embargo esta maximización de espacio trae la consecuencia de la complejidad espacial, ya que al ser un algoritmo factorial, cuando se trabajen con números demasiado grandes puede tomar meses o hasta años.