Universidad de los Andes 28/05/2018

David Felipe Niño Romero 201412734

María del Rosario León 201423755

Proyecto de Diseño y análisis de algoritmos problema C

1. Algoritmo de solución:
   1. Descripción de algoritmo:

Para solucionar el problema fue necesario empezar por encontrar la palabra más larga del texto ingresado como parámetro y empezar a variar los anchos de línea a partir de allí.

Debido a la naturaleza del problema, fue necesario simular los diferentes casos para los cuales los anchos de línea variaban. Con cada variación de los anchos de línea, el texto sufría un cambio considerable ya que las nuevas líneas, estaban conformadas por palabras que con la anterior configuración se veían cortadas.

Para cada ancho de línea se corta el texto, guardando las cadenas de texto cortadas en una posición de una lista, verificando que la línea sea cortada en un espacio blanco y esta sea de longitud igual o menor al ancho de línea. Luego de tener la lista con las cadenas de texto que corresponden al texto cortado, se hace la búsqueda por el río más largo del texto para este ancho de línea. Se itera sobre la lista de cadenas de texto cortadas. Se toman cadenas de la lista por parejas. Es decir, para la primera iteración, se toman los elementos 0 y 1 de la lista mencionada. Para cada una de las cadenas tomadas, se genera una lista en la que se guardan los índices de los espacios blancos. Con estas listas, y debido a que por iteración se toman 2 líneas, se tienen dos listas de índices de caracteres blancos. Estas son comparadas, recorriendo la lista de índices de caracteres blancos que corresponde a la línea de más abajo en el párrafo (lista de cadenas de caracteres) y comprobando si existe un índice que pueda conformar un río. Con cada iteración se suma una unidad si el río sigue aumentando o se empieza a encontrar un nuevo río si el anterior se corta. Esta iteración se hace con cada ancho de línea viable. Para volver un poco más eficiente el algoritmo, por cada vez que se corta el texto de entrada, el método guarda cuál fue la mínima longitud en la que una palabra fue cortada. Es decir, si, por ejemplo, la línea “naranja mecánica” fue cortada en el décimo carácter para una iteración, se guarda el número 7, que es el número mínimo que se necesita sumar al ancho de línea anterior para que en la siguiente iteración la palabra “mecánica” quepa en la misma línea con la palabra “naranja”.

El algoritmo inicial (el que varía el ancho de línea) termina cuando la longitud del río encontrado en el texto se vuelve mayor al número de líneas en el texto (para el caso del algoritmo esto se traduce en el tamaño de la lista que contiene las cadenas de caracteres cortadas). Esto se debe a que, si la cantidad de líneas es menor al tamaño del río encontrado hasta ahora, no hay manera en que el río sea más largo a partir de allí.

Explicación y argumentación del algoritmo implementado

* + 1. Método palabraMasLarga() :
       1. Contexto: párrafo: La cadena de caracteres que ingresa como parámetro al algoritmo.
       2. Precondición: El párrafo no está vacío.
       3. Postcondición: La respuesta será la longitud de la palabra de mayor longitud de la cadena de texto ingresada.
    2. Método rioMasLargo(): encuentra el río más largo para todas los anchos de línea posibles.
       1. Contexto: -anchoMin: número retornado por el método palabraMasLarga(). -Párrafo: la cadena de texto de entrada.
       2. Precondición:
       3. Postcondición: Se encuentra el río de longitud más larga en el texto y se encuentra el mínimo ancho de línea que forma el mencionado río.
    3. Método partir(**int** ancho): Parte el texto pasado como parámetro inicialmente al ancho de línea pasado como parámetro. Retorna las líneas del texto partidas al ancho de línea en un arreglo de strings.
       1. Contexto: -párrafo: la cadena de texto inicial. -ancho: El ancho de línea con el cual se desea partir la cadena de texto.
       2. Precondición: ancho > 0
       3. Postcondición: Se retorna una lista de strings con las líneas del texto original partidas al ancho de línea (o menos, en caso de que se parta una palabra) pasado como parámetro.
    4. Método encontrarRio(): Encuentra el río más largo en las líneas pasadas como parámetro.
       1. Contexto: lineas: es el arreglo de líneas de texto partidas que retorna el método partir(**int** ancho).
       2. Precondición: el tamaño de la lista líneas es mayor a cero
       3. Postcondición: Se encuentra la longitud del río más largo en el texto dado el ancho de línea pasado por parámetro.
    5. Método indicesDeEspacios(String linea): retorna una lista con los índices de los caracteres blancos que contiene la línea pasada como parámetro.
       1. Contexto: linea: cadena de texto a la cual se le quieren hallar los índices de sus caracteres vacíos.
       2. Precondición: la línea pasada como parámetro no está vacía ni es nula.
       3. Postcondición: Se retorna una lista con los índices de los caracteres blancos en la línea pasada como parámetro.
    6. Método compararEspacios(List<Integer> lista, List<Integer> lista2): retorna una lista con los números que dentro de las dos listas no difieren en más de 1.
       1. Contexto: -lista: lista con los índices de los caracteres blancos de una cadena de texto. -lista2: lista con los índices de los caracteres blancos de una cadena diferente de texto.
       2. Precondición: ninguna lista es nula ni está vacía.
       3. Postcondición: Se retorna una lista con los números que se comparten entre las listas o que la diferencia entre ellos es menor o igual a 1.

1. Análisis de complejidad
   1. Complejidad temporal:

Para el análisis de complejidad temporal del algoritmo en su totalidad, se asumirá como número m de elementos como la cantidad de palabras que se encuentran en la cadena de entrada (equivalente a n/(#espacios-1), donde n es el tamaño del problema y #espacios es el número de espacios que se encuentran en la cadena). EL método palabraMasLarga() itera m veces. El método rioMasLargo() itera O(n) veces, aproximadamente, en el peor de los casos, ya que en este caso, se tiene que iterar para todos los anchos hasta llegar a un ancho que iguala al texto sin ser partido. Sin embargo, en el método anterior, se hace un llamado a partir(**int** ancho) que ejecuta un ciclo O(n) veces, aproximadamente (si se tienen en cuenta las constantes, se debe multiplicar por un factor de 1/ancho, donde ancho es el ancho actual sobre el que se quiere partir la línea), en el peor de los casos. Luego de esto, se hace un llamado al método encontrarRio() que itera O(m) veces en el peor de los casos. Por lo tanto, juntando la complejidad de los métodos mencionados anteriormente, se tiene una complejidad parcial de en el peor de los casos de

* 1. Complejidad espacial:

Se guarda la cadena de texto de entrada, con lo cual para esto se tiene n. Adicionalmente, se tiene un arreglo de tamaño n/k para guardar las líneas del texto luego de partirlo. Sin embargo, este arreglo se reutiliza por cada entrada en la iteración. Por último, se usa un arreglo adicional (índices actuales) de tamaño n/2 en el peor de los casos para guardar los índices de espacios blancos por línea. Sin embargo, al igual que en el caso pasado, se reutiliza este arreglo por cada línea.

Esto da como resultado una complejidad de:

1. Comentarios finales:

Se probó el algoritmo diseñado para el peor caso propuesto caracteres. Con esta carga, el algoritmo se demoró 75 segundos en encontrar la respuesta. Se observa un comportamiento muy similar al calculado en la complejidad temporal. En la siguiente tabla, se muestra la relación de tiempo aproximado para diferente carga.

|  |  |
| --- | --- |
| n (carga en caracteres) | T (tiempo en segundos) |
| 200000 | 75 |
| 100000 | 20 |
| 50000 | 10 |
| 25000 | 3 |

Es importante resaltar que, en cada oportunidad para mejorar la eficiencia del algoritmo, se hizo lo posible por mejorarla. Esto se ve cuando se están buscando los índices de espacios blancos en la línea (no se recorre el arreglo entero), al momento de comparar las listas de índices (solo se recorre una lista) y al momento de detener el algoritmo (no espera hasta que el ancho de línea conduzca a una sola línea).