Laboratorio Nro. 3: LinkedLists y ArrayLists

Agustín Nieto García
Universidad Eafit
Medellín, Colombia
anietog1@eafit.edu.co

David Immanuel Trefftz Restrepo
Universidad Eafit
Medellín, Colombia
ditrefftzr@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. Cálculo de complejidad de los ejercicios 1.1, 1.2, 1.3 y 1.4 para ArrayList y LinkedLists.
   *Desarrollo en el código.

	ArrayList	LinkedList
1.1 (multiply)	O(n)	O(n)
1.2 (smartInsert)	O(n)	O(n)
1.3 (wherePivot)	O(n)	O(n)
1.4 (ejercicio4)	O(m*m+m*n)	O(m*n)

2. Explicación de 2.1 y 2.2
   Explicación 2.1
   El problema es el siguiente: el teclado está dañado y se presionan solas las teclas 'inicio' y 'fin' representadas como '[' y ']' respectivamente, que generan que lo que se digite quede desordenado, así, al entregarnos un String de solo letras, '[', ']' y '_', nos solicitan retornar lo que realmente quedó escrito si sabemos que la tecla 'inicio' manda el texto posterior al inicio del String y la tecla 'fin' manda el texto posterior al final del String. La función brokenKeyboard() es la solución a este problema, que recibe un String y retorna una LinkedList de StringBuilders y esto se debe a que requiere de adiciones múltiples en diferentes partes de un String. De la siguiente manera:

   1. Inicia en la posición 0 del LinkedList.
   2. Va sumando 1 a 1 los caracteres del String a un StringBuilder, hasta encontrarse con un corchete o llegar al final del String. (Si llega al final del String, agrega el StringBuilder a la posición en que se hallaba.
   3. Al encontrarse con un corchete, agrega el StringBuilder en la posición en que se hallaba dentro de la LinkedList y determina a qué posición agregará los siguientes caracteres: si es '[',

entonces al inicio, si es ‘]’, entonces al final. Continúa con el carácter que iba después del corchete y vuelve al paso 2.

4. Al final, se retorna la LinkedList con todos los StringBuilders y para conocer el resultado solo hace falta un ciclo forEach() que imprima 1 a 1 los StringBuilders dentro de ella.

Explicación 2.2
*leer el problema en http://codeforces.com/problemset/problem/313/B

Primero se intentó que cada vez que se solicitara una operación tuviera que calcularse con un ciclo que revisara si dos caracteres adyacentes coincidían. Sin embargo, esta solución era muy lenta, con una complejidad de O(n), así que hubo que intentar resolver el problema de otro modo. Así, se decidió hacer otro arreglo que guardara la cantidad de veces que se cumplía $S_i = S_{i+1}$ desde la posición inicial (0) hasta el tamaño del string (n), arreglo al que llamamos acum. Lo cual hizo que las preparaciones fueran O(n), pero que la complejidad por cada query fuera O(1) y solo requiriera hacer acum[right]-acum[left].

3. Cálculo de complejidades 2.1 y 2.2
*En el código

2.1 O(n) -> El procesamiento de cada línea + la respuesta.
2.2 O(n) -> preparativos. O(1) -> consultas.

4. ¿Qué es m y qué es n en el numeral 3.3?

2.1 n corresponde a la cantidad de caracteres en el String dado.
2.2 (en los preparativos) n corresponde al tamaño del arreglo, pero después el O(1) se debe a que lo único necesario es realizar una resta.

4) Simulacro de Parcial

1. C
2. C
3.
   a. q.size() > 1
   b. <=
   c. q.remove()
   d. q.remove()

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co