Laboratorio Nro. 03
Listas enlazadas y vectores dinámicos

Juan Pablo Yepes García
Universidad Eafit
Medellín, Colombia
jpyepesg@eafit.edu.co

Simón Cárdenas Villada
Universidad Eafit
Medellín, Colombia
scardenasv@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

3.1

Ejercicio	*Arraylist o vectores*	*LinkedList*
1.1 mapa	O(n)	n/a
1.4 nevera	n/a	Se hizo con queue y stack, Complejidad O(n*m)
2.1 teclado	n/a	O(n)

Aproximadamente podría tomar unos 108 Megabytes almacenar una matriz de ese tamaño, aunque dependiendo de la implementación y los tipos de datos almacenados podría variar significantemente. Este aproximado se hizo analizando las siguientes dos fuentes:

Wicklin, R. (2014, 28 abril). *How much RAM do i need to store that matrix?* Blog.sas.com. https://shorturl.at/afiMT

How to calculate the memory usage of a Java array. (s. f.). Javamex. Recuperado 27 de septiembre de 2020, de https://www.javamex.com/tutorials/memory/array_memory_usage.shtml

Ese problema de los identificadores puede ser solucionado con un mapeo de los datos, así que se implementó la misma estrategia sugerida en la documentación del laboratorio que es usar un HashMap y cambiar los identificadores por su posición en la lista, para evitar este problema.

3.2

El método del teclado roto tiene como parámetro un String s que es el que se va a analizar. Se empieza por crear un LinkedList de chars que almacenará las letras del String y luego con un ciclo se itera por cada char verificando si es un "[" o un "]". Con un índice se mantiene control sobre donde hay que insertar las letras de forma que se corrija el error del teclado.
Al final se crea una instancia de StringBuilder para concatenar por medio del método append() cada letra en el orden que es e imprimir en pantalla el resultado.

3.3
Complejidad: *O(n)*
3.4
Donde n es el número de letras (Characters) que componen el String que recibe el método.

4) Simulacro de Parcial

4.1 a) Una nueva lista donde L2 está al final de L1
b) O(n+m)
4.2 c) O(n)
4.4 - stack.push(token);
c) O(1)
4.8 O(n) y O(n)
4.10
d) O(n)
a) 6
b) O(n)
4.11
b) O(max(list)xn)
b) O(n)
4.12
- !s1.isEmpty()
- s2.push(s1.pop);
- s2.pop();
4.13
- O(n^2)
- O(n)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627 Tel: (+57) (4) 261 95 00
Ext. 9473