Tecnológico de Monterrey

Act 2.3 - Actividad Integral Estructura de Datos Lineales (Evidencia Competencia)

Nikole Morales - A01782557

TECNOLÓGICO DE MONTERREY

Campus Santa Fe

Grupo: 570

Fecha de entrega: 14 de julio de 2023

Profesor: Dr. Eduardo A. Rodríguez Tello

Programación de estructuras de datos y algoritmos fundamentales

Estructura de Datos Lineales

Reflexión

Las estructuras de datos lineales es una manera muy eficiente de almacenar y organizar datos e información dentro de la programación, las estructuras de datos lineales más comunes son los arras, linked list, stacks, queues, etc.

Para esta actividad se utilizó una double linked list, la cuál está compuesta por nodos. A diferencia de una linked list simple la doble tiene dos punteros lo cual hace que sea más eficiente recorrer dicha lista debido a que te puedes mover con más facilidad a los nodos de la izquierda o de la derecha.

Por otro lado las estructuras “stacks” son mayormente utilizados para almacenar datos de manera secuencial ya que utilizan un principio de "último en entrar, primero en salir", con esto nos podemos dar cuenta que para acceder a un elemento en una posición media se tendría que vaciar la lista, lo cual resultaría poco eficiente. Al parecido sucede con “queues” la manera para almacenar datos o para acceder de estos es secuencial, el principio que esta estructura utiliza es "primero en entrar, primero en salir", donde se insertan los nodos al final de la serie, y eliminan por el frente, por eso al igual que “stacks” acceder a un node medio sería muy complicado.

(UAM, 2015)

Dentro de esta situación problema se usó una “double linked list” debido ya que permite la inserción y eliminación de datos (nodos) con mayor facilidad. Aunque también se podrían usar arreglos, para los arreglos hay que declarar su tamaño desde un inicio, si bien es posible añadir más elementos este proceso tomaría más tiempo, lo que significa que su ejecución sería más lenta. En la actividad anterior se realizó de esa forma y se observó que el tiempo de ejecución fue más.

Cabe mencionar que dependiendo del contexto de la situación la estructura más eficiente puede variar, en este caso en específico fue más eficiente la “double linked list” ya que buscar los datos de la fecha y hora fue más sencillo.

(GeeksForGeeks, 2018)

Analizando las complejidades computacionales de las operaciones más importantes de la estructura que se utilizó para este programa son las siguientes:

Operación	Algoritmo	Complejidad computacional	Descripción

Inserción	**addFirst**	O(1)	Se agrega un nodo al inicio de la lista creando un nuevo *head.* La complejidad es constante debido a que solo se actualizan los punteros para que apunten al nuevo *head*, independientemente de la cantidad de elementos en la lista el tiempo para agregar el nuevo nodo al inicio es el mismo.
	addLast	O(1)	Se agrega un nodo al final de la lista creando un nuevo *tail.* La complejidad es constante debido a que solo se actualizan los punteros para que apunten al nuevo *tail*, independientemente de la cantidad de elementos en la lista el tiempo para agregar el nuevo nodo al final es el mismo.

Tabla 1.1: Complejidad computacional de las operaciones de inserción.

Operación	Algoritmo	Complejidad computacional	Descripción
Borrado	**~DLinkedList()**	O(n)	Es un destructor que se encarga de recorrer todos los nodos de la double linked list para eliminarlos a cada uno. La complejidad es lineal ya que recorre cada nodo de la lista (uno por uno) lo que significa que la complejidad también es proporcional al número de nodos. (Whitney, 2023)

Tabla 1.2: Complejidad computacional de las operaciones de borrado.

Operación	Algoritmo	Complejidad computacional	Descripción
			Este algoritmo de búsqueda binaria se utilizó para encontrar un elemento específico en la double linked list.

Búsqueda	**binarySearch**	O(log n)	Debido a que la lista está ordenada, la búsqueda binaria puede dividir el espacio de búsqueda a la mitad en cada iteración, lo que resulta en una complejidad logarítmica. El tiempo de ejecución de la búsqueda aumenta de forma logarítmica. La búsqueda se hace a partir de las fechas solicitadas por el usuario. (Aguilar, 2006)

Tabla 1.3: Complejidad computacional de las operaciones de búsqueda.

En conclusión es más conveniente usar listas doblemente enlazadas para esta situacion problema ya que es más flexible a la hora de buscar la información dentro del archivo “bitácora.txt”, ya que al ser una lista con demasiados datos se puede acceder a ella más fácilmente ya sea para buscar información, como en el caso de que se tiene que buscar por rango de fecha, así cómo también el ordenamiento de dichos datos es menos tardado.

Referencias

Aguilar, L. J., & García, L. S. (2006). *Programación en C++: un enfoque práctico*. McGraw-Hill Interamericana de España. Recuperado el 14 de Julio de 2023 en: http://repositoriokoha.uner.edu.ar/fing/pdf/5230.pdf

Whitney, T. (2023). *Destructores (C++)*. Microsoft. Recuperado el 14 de julio de 2023, en: https://learn.microsoft.com/es-es/cpp/cpp/destructors-cpp?view=msvc-170

S.A. *QuickSort – Data Structure and Algorithm Tutorials (2014, enero 7).* GeeksforGeeks. Recuperado el 14 de julio de 2023 en: https://www.geeksforgeeks.org/quick-sort/

S.A. *Introduction to doubly linked list – data structure and algorithm tutorials.* (2018, mayo 25). GeeksforGeeks. Rcuperado el 14 de julio de 2023 en: https://www.geeksforgeeks.org/data-structures/linked-list/doubly-linked-list/

Moltó Martínez, G. (2011). Estructuras de Datos Lineales: Pila, Cola y Lista con Punto de Interés. Recuperado el 14 de julio de 2023.

UAM. (2015). *Estructura de Datos: Estructura de Datos Lineales.* México. Recuperado el 14 de julio de 2023 de: http://ri.uaemex.mx/oca/view/20.500.11799/34615/1/secme-19001.pdf