Jorge Daniel Cruz Case – A01634536

Reflexión Actividad integradora 2.3

Durante la realización de la actividad integradora se nos dio la tarea de utilizar varios tipos de estructuras de datos que hemos visto durante el segundo periodo de la clase de programación. Para esta fue importante conocer las ventajas y desventajas de cada una de ellas.

En el programa que fue realizado hay varias cosas que se deben tener en consideración para el código y ejecución, uso de memoria y tiempo de ejecución. Es importante que no ocurran fallas con el uso de memoria, también teniendo en cuenta la cantidad de datos a manejar (16807) la complejidad temporal debe de ser lo más baja posible para evitar un programa que tarde minutos de doble digito en correr.

Teniendo en cuenta lo del uso de la memoria podemos analizar las diversas estructuras lineales utilizadas y/o vistas en clase, *Doubly Linked List, Linked List, Queue & Stack.*

Un *Stack,* es un tipo de estructura de datos que contiene una serie de Nodos en una secuencia lineal, teniendo una estructura LIFO o “Last In, First Out” mimicando a un Stack de platos, al añadir un dato es equivalente a añadir un plato a una pila de estos. Este cuenta con las funciones normales de una estructura de datos como “is empty”, “pop”, “push” & “top”, con su único dato importante es que elemento está en el “top” del Stack.

Ahora que ya sabemos el funcionamiento de *Stack,* podemos analizar *Doubly Linked List* & *Linked List* para posteriormente compararlas.

Según Geeks for Geeks, la *Linked List,* es una estructura de datos en la cual los elementos de la llista se almacenan mediante apuntadores y conectores entre sí. Puede decirse que una Linked List tiene las siguientes partes, Nodos y conectores entre los mismos, un Nodo contiene el dato dentro del mismo, y un apuntador hacia el siguiente. Linked List es una de las estructuras de datos más utilizadas en el mundo de la programación, por lo cual lentamente se ha llegado a un consenso de las funciones primarias que toda Linked List deberá tener. Estas pueden ser vistas en la siguiente tabla:

Método	Complejidad	Funcionamiento / Uso
Inserción (addFirst)	$O(1)$	Recibe un valor como parámetro, el cual hace que su *next* apunte hacia la cabeza de la lista, posteriormente mueve la cabeza al nodo parámetro. La complejidad es O(1) ya que esto solo se realiza al principio de la lista y no depende de n.
Inserción (addLast)	$O(n)$	Recibe un valor como parámetro para insertarlo al final de la lista. Si la lista esta vacía, el parámetro se hace la cabeza, de lo contrario recorrerá la lista mediante el clásico "p = p->next" hasta que p->next sea igual a nulo. Cuando esto pase el "p->next" apuntará al nodo con el valor y el *next* del valor será NULL.
Borrado (deleteData)	$O(n)$	Recibe un valor que quiere borrar dentro de la lista. Primero checa si el valor esta en la cabeza, si lo esta la cabeza se convierte en el *next*, y el nodo se borra. De lo contrario avanzará la lista hasta que encuentre el nodo con el valor, checa que el mismo no se nulo, si no lo es borra el nodo y desplaza los nodos. Si se pudo borrar regresa TRUE, de lo contrario un FALSE. Su complejidad es algo similar a O(n) debido a que se debe recorrer el arreglo para encontrar el dato, si el dato esta al inicio su complejidad es O(1).
Borrado (deleteAt)	$O(n)$	Recibe un índice a borrar en el arreglo. Primero comprueba que el índice no este fuera de rango. Si no lo está prosigue con la búsqueda. Si el índice es la cabeza la cabeza se convierte en el siguiente y se borra el nodo. La lista se avanzará hasta que encuentre el índice ingresado, una vez que lo encuentra lo borra y desplaza

		la lista. Su complejidad es O(n) ya que se debe recorrer la lista para encontrar el dato.
Search (findData)	$O(n)$	Recibirá un dato que quiere buscar en la lista, si la lista esta vacío regresará -1. De lo contrario recorrerá la lista hasta que el dato en el nodo se igual al ingresado, posteriormente regresará el índice en el que esta ese dato. El método también se puede realizar para que busque un dato en una posición, el cambio es que la lista se avanza hasta que el índice sea igual al ingresado. Su complejidad es O(n) ya que se debe recorrer el arreglo para encontrarlo.

Ahora pasamos a *Doubly Linked List*, esta es una estructura de datos la cual a diferencia de su contraparte la "Linked List", esta consiste en Nodos que almacenan datos, y en lugar de apuntar hacia el Nodo de enfrente, estos también apuntan a los Nodos anteriores. Esta cuenta con 2 datos importantes, Head & Tail, la cabeza y la cola de la Lista. Similar a la Linked List, la Double Linked List es una de las estructuras de datos más utilizadas en el mundo de la programación, por la cual similar a su contraparte, se ha llegado a un consenso de que funciones o métodos esta estructura debe contener, estás son las siguientes:

Método	Complejidad	Funcionamiento / Uso
Inserción (addFirst)	$O(1)$	Recibe un valor como parámetro, primero checa si la lista está vacía, si lo está el valor se hace la cabeza y la cola al mismo tiempo, si no hace que su *next* apunte hacia la cabeza de la lista, posteriormente mueve la cabeza al nodo parámetro. La complejidad es O(1) ya que esto solo se realiza al principio de la lista y no depende de n.

Inserción (addLast)	$O(1)$	Recibe un valor como parámetro para insertarlo al final de la lista. Si la lista está vacía, el parámetro se hace la cabeza, de lo contrario se moverá a la cola, el *next* de la cola apuntará al valor y la cola se convertirá en este. Su complejidad es O(1) porque solo se hace 1 vez al final de la lista y no depende de su tamaño.
Borrado (deleteData)	$O(n)$	Recibe un valor que quiere borrar dentro de la lista. Si el dato esta en la cabeza, la borra y recorre la lista. De lo contrario deberá recorrer la lista mientras el dato sea diferente al nodo. Si el dato está en la cola, la borra y recorre la lista, de lo contrario solamente borra el nodo y recorre la lista. Su complejidad es O(n) ya que se debe recorrer la lista hasta encontrar el dato.
Search (findData)	$O(n)$	Este search funciona exactamente igual que aquel de la single Linked List, ya que este search no depende de la cola ni de la dobla referencia.

Teniendo esto en cuenta ¿Por qué fue utilizada Doubly Linked List y no Single Linked List? Para esto tenemos que ver las ventajas y desventajas de ambos.

Según “Geeks for Geeks” y “Tutorials Point”, Single Linked List es la estructura a elegir cuando el propósito de la estructura es salvar memoria debido a que esta solamente utiliza un solo nodo en la memoria, también se está manejando con una pequeña cantidad de datos, es conocido que sus complejidades llegan a variar en O(n). También es transversal en una dirección.

Por otro lado, Doubly Linked List presenta más consumo de memoria ya que esta utiliza 2 nodos en la memoria, pero a diferencia de la single Linked List, la Doubly tiene una complejidad promedio de O(1), la cual la hace preferente al manejar grandes cantidades de datos, como los de la tarea. Esta también está enfocada en mantener memoria y una rápida ejecución, mientras permite realizar búsquedas e iteraciones dentro de la lista.

En conclusión, Single Linked List es favorable cuando la memoria es limitada y la cantidad de datos a trabajar es poca, y Doubly Linked List es favorable cuando la memoria no es tanto problema, la cantidad de datos es alta y se requieren hacer iteraciones y búsquedas en la lista.

Las ventajas de Doubly Linked List son prácticamente los requisitos o funciones que se realizan en el programa realizado, las búsquedas e iteraciones dentro de las listas son comunes dentro del programa desarrollado, por lo cual es obvio porque se decidió utilizar esta estructura lineal para la actividad.

Referencias

Isaac Computer Science. (s.f.). *Stack.* Recuperado de: https://isaaccomputerscience.org/concepts/dsa_datastruct_stack?examBoard=all&stage=all

Geeks for Geeks. (2021). *Doubly Linked List | Set 1 (Introduction and Insertion)*. Recuperado de: https://www.geeksforgeeks.org/doubly-linked-list/

Geeks for Geeks. (2021). *Linked List | Set 1 (Introduction).* Recuperado de: https://www.geeksforgeeks.org/linked-list-set-1-introduction/

Sharma N. (2019). *Difference between Singly linked list and Doubly linked list in Java.* Tutorials Point. Recuperado de: https://www.tutorialspoint.com/difference-between-singly-linked-list-and-doubly-linked-list-in-java

Tutorials Point. (s.f.). *Data Structure and Algorithms - Linked List.* Recuperado de: https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm