CAMPUS QUERÉTARO

TC1031. PROGRAMACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS FUNDAMENTALES (GRUPO 826)

"Actividad Integradora 2. Estructuras de Datos Lineales"

Evidencia presentada por la estudiante:

A01706095 - Naomi Estefanía Nieto Vega

Profesor:

Dr. Eduardo Arturo Rodríguez Tello

Fecha de entrega:

Lunes 12 de abril de 2021

Actividad Integradora 2. Estructuras de Datos Lineales

A01706095 - Naomi Estefanía Nieto Vega

***Resumen*—En esta reflexión se hablará acerca de la importancia y eficiencia del uso de las diferentes estructuras de datos lineales en la solución de la situación problema planteada, además se hablará acerca de las listas doblemente ligadas y por qué son relevantes para la solución propuesta además del análisis de la complejidad de sus principales funciones como lo son la inserción, el borrado y la búsqueda y su impacto en la solución propuesta.**

***Index Terms*—linear data structures, data structures, stack, quicksort, algorithm**

I. INTRODUCCIÓN

Hoy en día los algoritmos de ordenamiento y las estructuras de datos son muy importantes en el área computacional y en muchas más áreas en los que son utilizados ya que de acuerdo con la literatura un algoritmo es una serie o secuencia ordenada de pasos que dan solución a un determinado problema. No obstante, pueden existir distintos algoritmos que den solución a un mismo problema y es aquí donde entra la importancia de implementar el más eficiente en cuanto a tiempo y recursos ya que estos deben ser utilizados correctamente y de esta forma la computadora podrá darnos resultados muy buenos.

Asimismo, al conocer distintos tipos de estructuras de datos y su implementación podemos hacer uso de ellos para mejorar el manejo que tenemos sobre los algoritmos de acuerdo con las necesidades que tenga el problema, y así proveer una solución adecuada, eficiente y rápida.

Para el desarrollo de esta situación problema, principalmente nos enfrentamos al problema del manejo de una bitácora con gran volumen de datos, esto es un factor muy importante a considerar para plantear una solución adecuada y eficiente, mediante el uso de listas doblemente ligadas y el análisis de la complejidad de sus funciones que se explicará a continuación. En la figura 1, podemos observar la complejidad de las funciones de algunas estructuras de datos lineales.

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Figura 1. Complejidad de algunas funciones de las estructuras de datos lineales. [l] Imagen obtenida de: www.bigocheatsheet.com

II. DESARROLLO

Para los algoritmos de búsqueda tenemos dos opciones, la búsqueda secuencial que en su mejor caso cuenta con una complejidad de C(n) = 1 si el elemento buscado es el primero de la lista, y en su peor caso cuenta con una complejidad de C(n) = n si el elemento buscado no existe o se encuentra en la última posición. La segunda opción es la búsqueda binaria, cabe aclarar que ambos métodos realizan lo mismo pero con un algoritmo distinto, en este caso utilicé la búsqueda binaria en un arreglo ya ordenado que cuenta con una complejidad de O(log n) para su peor caso y un O(1) en su mejor caso.

Sin embargo, para el desarrollo de esta actividad integradora utilicé el algoritmo de ordenamiento «QuickSort» y a continuación explicaré los motivos por los cuales consideré a este algoritmo como la mejor opción de los algoritmos de ordenamiento vistos:

1. Es un algoritmo basado en la técnica de divide y vencerás por lo que funciona eligiendo un pivote para particionar el arreglo con base en esto y así ir ordenando los elementos.
2. No necesita espacio auxiliar de almacenamiento ya que es un algoritmo de ordenamiento in situ, esto quiere decir que no necesita espacio de memoria auxiliar para hacer el ordenamiento, a diferencia de «MergeSort» por ejemplo, en el que tenemos que crear dos arreglos temporales para fusionar las mitades.
3. Es un algoritmo que es estable, esto quiere decir que si tenemos dos objetos con la misma clave aparecerán en el mismo orden en la salida arrojada.
4. Aunque sabemos que en el caso de las listas doblemente ligadas un mejor algoritmo para esto sería el «MergeSort», podemos adaptar el algoritmo de «QuickSort» para que funcione de forma iterativa, que si bien sabemos esto no altera su complejidad ya que la esencia del algoritmo es la misma, podemos notar que mejora en cuestión de eficacia de uso de memoria de acuerdo con la complejidad espacial, ya que en este caso nosotros manipulamos el stack.

Las estructuras de datos son formas de organización y representación de datos o información, en el caso de la programación, son formas de organización que nos permiten utilizar estos datos de una forma efectiva. Existen dos tipos de estructuras de datos:

- Estructuras de datos lineales
- Estructuras de datos no lineales

Su principal diferencia es que la estructura de datos lineal tiene a sus elementos de datos dispuestos de manera secuencial y

cada uno está conectado con su elemento anterior y siguiente. Por otra parte, las estructuras de datos no lineales no tienen una secuencia establecida para conectar a sus elementos, por lo que un elemento puede tomar varias rutas para conectarse con otros elementos. Este tipo de estructuras son un poco más complejas de implementar pero son más eficientes en cuanto a uso de memoria en la computadora. [2] En la figura 2 podemos observar de manera gráfica la clasificación de los tipos de estructuras de datos.

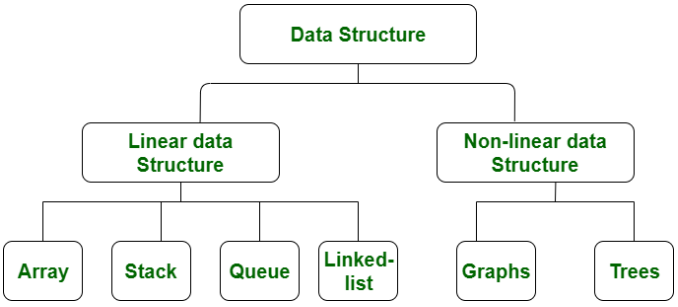

Figura 2. Clasificación de las estructuras de datos. Imagen obtenida de: geeksforgeeks.org/difference-between-linear-and-non-linear-data-structures/

Ahora bien, para la solución de esta segunda actividad integradora fue muy importante el uso de estructuras de datos lineales que como sabemos son estructuras que tienen elementos conectados de manera secuencial de forma que el elemento está conectado a su elemento anterior y al próximo, además son estructuras relativamente sencillas de implementar pues la naturaleza de la memoria de las computadoras también es secuencial. [2] Algunas estructuras de datos de esta naturaleza son las listas ligadas (Linked List), colas (Queue), pilas (Stack) y arreglos (Array). La importancia de las estructuras de datos lineales radica en que gracias a ellas podemos hacer más eficiente la organización de datos y la forma en que posteriormente utilizamos o analizamos estos datos. Además nos permiten integrarlas en conjunto con algoritmos de ordenamiento de forma que podemos mejorarlo en términos de eficiencia.

Para este caso en particular utilicé nuevamente el algoritmo de Quicksort en su versión iterativa, por lo que su complejidad algorítmica y espacial permanece siendo la misma (ver figura 3), ya que la esencia del algoritmo es la misma y trabaja de la misma forma que la versión recursiva, la única diferencia es que en la implementación iterativa manejamos explícitamente el stack a diferencia de la implementación recursiva en la que dependemos del stack que viene implícito.

Mejor	Promedio	Peor	Estable	Complejidad espacial
$O(n \log n)$	$O(n \log n)$	$O(n^2)$	Si	$O(\log n)$

Figura 3. Complejidad algorítmica del algorítmo «QuickSort».

Otra modificación que fue realizada al algoritmo es en los parámetros de entrada, ya que el anterior recibe un arreglo y ahora recibe una lista doblemente ligada (Doubly Linked List).

Las listas ligadas son un tipo de estructura de datos lineal en la que los elementos no son ordenados o almacenados en localidades temporales de memoria contiguas, sino que están ligados mediante el uso de apuntadores (ver figura 4) [4]. Sabemos que existen tres principales tipos de listas ligadas (Linked List):

- Lista ligada simple
- Lista doblemente ligada
- Listas circulares

La principal diferencia entre la lista ligada simple y la doblemente ligada es que en la simple únicamente tenemos un apuntador o referencia al siguiente nodo. En cambio, en la doblemente ligada además del apuntador hacia el siguiente nodo tenemos otro extra que apunta hacia el nodo anterior y esto nos da la posibilidad de volver más eficientes nuestras funciones al momento de implementarlas.

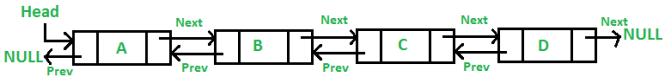

Figura 4. Funcionamiento de las listas doblemente ligadas. Imagen obtenida de: geeksforgeeks.org/doubly-linked-list/

Por lo tanto, para la solución de esta actividad integradora es preferible la implementación de las listas doblemente ligadas (Doubly Linked List) porque gracias a que tenemos un apuntador que apunta hacia el final de la lista o el «tail» además del «head», es posible mejorar la complejidad algorítmica de nuestra implementación y reducir de un O(n) a un O(1) que es mucho mejor.

Algunas de las funciones básicas de las listas doblemente ligadas son las siguientes:

- Insertion
- Deletion
- Search

Dichas funciones son muy importantes porque con ellas principalmente es que podemos comenzar a manejar datos en nuestra Doubly Linked List, para insertar, eliminar o buscar datos. La complejidad de estas funciones en parte depende de su implementación, ya que dependiendo de si tenemos un tail su complejidad mejora. Por ejemplo, para el insertion si es al principio de la lista su complejidad es de O(1), en cambio si es para añadir al final su complejidad con un apuntador a tail es de O(1) ya que reduce significativamente la cantidad de procesos que hay que realizar dentro de la función y en su peor caso es de O(n).

Para la función de borrado, en su mejor caso tiene una complejidad de O(1) y en su peor caso una complejidad de O(n) porque debe iterar en toda la lista y comparar por cada nodo la posición o el valor del dato y detenerse hasta que la condición de que el valor sea igual al valor del nodo en ese momento para proceder a eliminarlo. Y por último la función de búsqueda tiene una complejidad de O(n) en ambos casos porque tiene que iterar en la lista y por cada nodo que exista debe comparar si el valor del dato es igual al solicitado y hasta que se cumpla la condición es que termina esta función, entonces esto nos genera una complejidad espacial de O(n). En la figura 5 se puede observar un resumen de las complejidades de cada función.

Función	Promedio	Peor
Insertion	$O(1)$	$O(n)$
Deletion	$O(1)$	$O(n)$
Search	$O(n)$	$O(n)$

Figura 5. Complejidades de las principales funciones de las listas ligadas.

Para concluir, el desarrollo de estas funciones y su implementación son muy importantes para el desarrollo de la solución ya que una implementación no tan acertada se traduce en un uso ineficiente de los recursos como lo es la memoria. Es por ello que se debe cuidar la manera de implementar cada una y hacerlas lo más eficientes posibles de acuerdo con los requerimientos solicitados.

III. CONCLUSIÓN

En conclusión gracias a esta actividad logré comprender mejor el uso de nuevas estructuras de datos lineales que no había utilizado antes como las Linked List. Asimismo entendí mejor los conceptos relacionados con las estructuras de datos lineales y su importancia en el área computacional, aprendí las diferencias de estas y sus aplicaciones. También sobre la importancia de la complejidad computacional de cada algoritmo y el impacto que esto tiene una vez que se implementa en cuestión de recursos y eficacia.

REFERENCIAS

[1] Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) (2020). Big O Algorithm Complexities. https://www.bigocheatsheet.com/

[2] Difference between Linear and Non-linear Data Structures. (2021). Tutorialspoint. https://www.tutorialspoint.com/difference-between-linear-and-non-linear-data-structures

[3] GeeksforGeeks. (2020b, abril 22). Difference between Linear and Non-linear Data Structures. https://www.geeksforgeeks.org/difference-between-linear-and-non-linear-data-structures/

[4] GeeksforGeeks. (2020a). Linked List Data Structure. https://www.geeksforgeeks.org/data-structures/linked-list/

[5] GeeksforGeeks. (2021, 9 marzo). Doubly Linked List | Set 1 (Introduction and Insertion). https://www.geeksforgeeks.org/doubly-linked-list/

[6] Linear Linked List: Time and Space Complexity of Insertion and Deletion. (2020, 9 mayo). [Vídeo]. YouTube. https://bit.ly/3wPn2hx