

Reflexión Actividad 2.3

Programación de estructuras de datos y algoritmos fundamentales (TC1031.570)

Fausto Alejandro Palma Cervantes (A01639224)

16 de julio de 2022

Mientras que el bloque pasado de actividades de esta materia se enfocó en algoritmos de ordenamiento y búsqueda, este siguiente bloque de actividades se centra en distintas estructuras de datos lineales. Similar a la utilidad que tienen los algoritmos de ordenamiento y búsqueda, este tipo de estructura de datos son de gran uso dentro de la Ingeniería en Tecnologías Computacionales ya que facilitan la interpretación e interacción con conjuntos de datos al ordenarlos en una lógica secuencial lineal. Esto además disminuye la complejidad computacional de las operaciones de análisis a las que se pueden someter este tipo de conjuntos. Dentro de las estructuras de datos lineales, existen tres distintos tipos en los cuales se centró este bloque:

1. **Listas encadenadas:** Estructuras cuyos elementos no están almacenados en un espacio de memoria continuo y que se encadenan al referenciarse entre sí. (*Linked List Data Structure*, 2022)
2. **Pilas:** También conocidas como *Stacks*, este tipo de estructuras sigue el orden de operación *Last In First Out*. (*Stack Data Structure*, 2022)
3. **Filas:** También conocidas como *Queues*, este tipo de estructuras sigue el orden de operación *First In First Out*. (*Queue Data Structure*, 2022)

Además, al usar las listas encadenadas, se identificó que hay distintos tipos en los que se pueden clasificar este tipo de estructuras lineales dependiendo de sus propiedades presentes. Siendo más específicos, las *Linked Lists* y *Doubly Linked Lists* son dos de los tipos de listas encadenadas más conocidos y utilizados. En una lista encadenada tradicional (*Linked List*), los elementos normalmente tienen un único apuntador el cual usan para referenciar al siguiente elemento en la lista. Lo que hace a las *Doubly Linked Lists* distintas del modelo tradicional es que los elementos de esta utilizan un apuntador adicional el cual les permite referenciar al mismo tiempo al elemento previo y siguiente de la lista. (GeeksforGeeks et al, 2022, *Doubly Linked List*) Este apuntador adicional disminuye la complejidad espacial y temporal de operaciones que requieren moverse a través de la lista encadenada en dirección contraria a la de los apuntadores tradicionales. La siguiente tabla muestra visualmente la estructura de cada una de las listas encadenadas ya mencionadas:

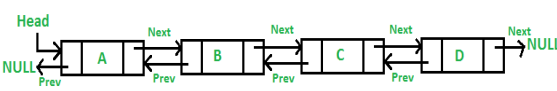
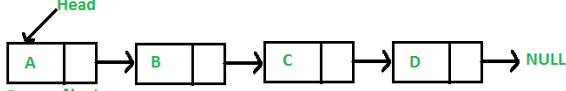
<i>Linked List</i>	<i>Doubly Linked Lists</i>
	

Tabla 1. Estructura de una *Linked List* y una *Doubly Linked List*
(GeeksforGeeks et al, 2022, *Linked List*) (GeeksforGeeks et al, 2022, *Doubly Linked List*)

Además de ayudar a ejemplificar la relación entre elementos de cada tipo de lista encadenada, los diagramas de la tabla anterior muestran que existe un apuntador *Head*. Ya que esta estructura de datos está compuesta por elementos individuales y no agrupados en la memoria, el propósito de este apuntador es referenciar al primer elemento de la lista y es este apuntador mediante el cual se accede a la lista. En caso de ser importante el acceder a la lista desde su último elemento, además del apuntador *Head* también se puede añadir un apuntador *Tail* al final de ella.

Como parte de la entrega para la Actividad 2.3 (*Actividad Integral estructura de datos lineales*) de esta unidad de formación, se solicitaron nuevamente las funcionalidades de organización y búsqueda de registros en una bitácora. La diferencia entre la actividad anterior y esta fue que la estructura de datos de la bitácora cambió de un vector a una *Doubly Linked List*. La razón por la cual se decidió utilizar esta estructura de datos lineal tiene que ver con los algoritmos que se escogieron para la búsqueda y organización de la bitácora. Como se mencionó en la actividad anterior, para su organización se utilizó el algoritmo *Merge Sort*, mientras que para la búsqueda dentro de ella se usó la búsqueda binaria (*Binary Search*). Cabe recordar que la razón detrás de esta decisión fue que ambos algoritmos tienen la complejidad computacional más simple de todos los algoritmos dentro de sus respectivos enfoques. Tanto para *Merge Sort* como para *Binary Search* es importante poder acceder al elemento anterior dentro de los procesos que llevan a cabo. Siendo que la *Doubly Linked List* es el único tipo de lista encadenada cuyos elementos tienen apuntadores al elemento anterior, este tipo de estructura de datos lineal fue la más calificada para utilizar en este caso.

Como también se habló en la entrega de la actividad anterior, la complejidad computacional juega un rol muy importante al escoger los algoritmos a implementar dentro de un proyecto. Como sabemos ahora, indirectamente también ayuda a determinar la estructura de datos de un conjunto dado que cada algoritmo tiene distintos requisitos que no todas las estructuras cumplen. En el caso de la bitácora, la inserción, borrado y búsqueda de registros fueron algunas de las operaciones básicas que se utilizaron dentro de la implementación de las funcionalidades solicitadas. Al hablar del desempeño de este programa, es importante considerar que cada una de estas operaciones básicas tiene su propia complejidad computacional la cual puede llegar a depender en la estructura del conjunto de información con el que interactúa.

De las tres operaciones mencionadas, la búsqueda probablemente sea la más usada. Se utiliza inclusive dentro de la inserción y borrado de registros de la bitácora para encontrar el elemento que se desea borrar o insertar. Ya que se implementó una búsqueda binaria, la complejidad computacional de esta operación por sí sola es $O(n \log n)$ justo como se discutió en el entregable pasado. Por otra parte, dentro de las operaciones de inserción y borrado, se utilizó una búsqueda secuencial, la cual (dado que la bitácora es un arreglo de registros ordenados por fecha) la complejidad computacional de estas operaciones es $O(N/2)$. La decisión de utilizar la búsqueda secuencial en las operaciones de borrado e inserción se debe a que la bitácora se importa al programa desde un archivo *.txt* el cual ya incluye todos los registros disponibles. Mientras que para insertar los registros a la bitácora dentro del programa es más fácil pasar por cada uno de los renglones del archivo *.txt* de manera secuencial, borrar los registros no es necesario porque dejaría incompleto al conjunto de información utilizado, lo cual no es recomendable dada la naturaleza de la situación

problema. Cabe mencionar que la acción de insertar o borrar un elemento de la bitácora como tal tiene una complejidad de $O(1)$ ya que sólo se necesitan hacer los cambios asociados a la operación a un sólo elemento.

Recordando la situación problema de esta unidad de formación, ella se enfoca en la detección de ataques cibernéticos de redes de bots. La descripción de la situación problema en su página de Canvas menciona explícitamente que “una forma de detectar este tipo de situaciones es a través de recopilación de toda la información pertinente que nos permitan hacer búsquedas optimizadas para la detección oportuna de accesos maliciosos a través de redes de robots”. Similar a lo que se mencionó en la actividad anterior con el uso de algoritmos de búsqueda y ordenamiento, el uso de estructuras de datos lineales es relevante a la problemática ya que permiten estructurar grandes cantidades de información de manera que la implementación de algoritmos sea mucho más computacionalmente simple. Al escoger un tipo de estructura de datos para representar a un conjunto de información, esta decisión debe ser una decisión informada la cual considere las distintas operaciones que las propiedades de cada estructura permite llevar a cabo y la complejidad computacional que requiere cada una de estas operaciones.

Bibliografía:

Linked List Data Structure. (2022). GeeksforGeeks.

<https://www.geeksforgeeks.org/data-structures/linked-list/#doublyLinkedList>.

Queue Data Structure. (2022). GeeksforGeeks.

<https://www.geeksforgeeks.org/queue-data-structure/?ref=lbp>.

Stack Data Structure. (2022). GeeksforGeeks.

<https://www.geeksforgeeks.org/stack-data-structure/?ref=lbp>.

GeeksforGeeks et al. (2022). *Doubly Linked List | Set 1 (Introduction and Insertion)*.

GeeksforGeeks. <https://www.geeksforgeeks.org/doubly-linked-list/>.

GeeksforGeeks et al. (2022). *Linked List | Set 1 (Introduction)*. GeeksforGeeks.

<https://www.geeksforgeeks.org/linked-list-set-1-introduction/>.