“Instituto Tecnológico y estudios superiores de Monterrey”

EVIDENCIA

“Reflexión Actividad 2.3”

Materia:
Programación de estructuras de datos y algoritmos fundamentales
(Gpo 570)

Profesor:
Dr. Eduardo Arturo Rodríguez Tello

Alumno/a:
Ayetza Yunnuen Infante García | A01709011
Luis Carlos Rico Almada | A01252831

14/Julio/2023

Ayetza Yunnuen Infante García | A01709011

El uso de estructuras de datos lineales juega un papel importante en términos de importancia y eficiencia ya que son esenciales para organizar y almacenar grandes cantidades de datos de manera eficiente. En el caso de una bitácora, donde se registra información en secuencia temporal, es fundamental contar con una estructura adecuada para almacenar y manejar esos registros. Las estructuras de datos lineales permiten acceder, insertar, eliminar y buscar datos de forma rápida y eficiente; de igual forma que estas ofrecen diferentes ventajas en términos de eficiencia en la manipulación de registros *(Universidad de Los Andes. Facultad de Ingeniería, s.f)*. Por otro lado, una Doubly Linked List ofrece la ventaja adicional de un acceso eficiente hacia adelante y hacia atrás, lo cual es útil para realizar operaciones como el ordenamiento y la búsqueda en rangos de fechas.

En la aplicación realizada, se requiere buscar registros en un rango de fechas y ordenar la información por fecha y hora. Aquí es donde las estructuras de datos lineales juegan un papel clave. El uso de una estructura de datos lineal ordenada, como una Doubly Linked List ordenada por fecha y hora, proporciona un acceso eficiente a los registros en un rango específico *(Javatpoint, s.f)*. Además, el uso de algoritmos de búsqueda como la búsqueda binaria y el algoritmo de ordenamiento como Merge Sort, adaptados a la estructura de datos elegida, permiten realizar estas operaciones de manera rápida y eficiente *(Programiz, 2021)*.

En la solución de este reto, fue preferible emplear una Doubly Linked List en lugar de una Linked List, ya que la Doubly Linked List, al tener referencias tanto al siguiente nodo como al nodo anterior, permite un acceso eficiente tanto hacia adelante como hacia atrás. Esto es crucial para implementar el algoritmo de ordenamiento Merge Sort, ya que requiere dividir la lista en subconjuntos y combinarlos en orden *(Programiz, 2021)*. De igual forma para realizar la búsqueda de registros en un rango de fechas, se menciona el uso del algoritmo de búsqueda binaria. La Doubly Linked List proporciona un acceso eficiente hacia adelante y hacia atrás, lo que facilita la implementación de la búsqueda binaria. Al poder avanzar y retroceder en la lista de manera eficiente, se reduce el tiempo de búsqueda en comparación con una Linked List, que solo permite un acceso secuencial y por último la Doubly Linked List permite eliminar un nodo de forma eficiente, ya que solo requiere actualizar las referencias del nodo anterior y el nodo siguiente. En una Linked List, si se quiere eliminar un nodo específico, se necesita recorrer la lista hasta encontrar el nodo anterior, lo cual puede ser ineficiente en términos de tiempo. *(Sharma, A; 2021)*.

En cuanto al análisis de la complejidad computacional de las operaciones básicas (inserción, borrado y búsqueda) y cómo impactan en el desempeño de la solución se puede deducir que la complejidad computacional de la inserción y el borrado en una Doubly Linked List es O(1). Esto se debe a que la inserción se realiza al principio o al final de la lista, lo cual solo requiere actualizar las referencias de los nodos adyacentes, por lo que garantiza un tiempo de ejecución constante y eficiente; por otro lado en cuanto el borrado de un nodo sólo requiere actualizar las referencias de los nodos adyacentes *(Universidad de Valladolid, M,*

Mario; s.f,). En este caso, la complejidad de la búsqueda es O(log n), donde n es el número de elementos en la lista. La búsqueda binaria permite dividir el espacio de búsqueda a la mitad en cada iteración lo que permite encontrar rápidamente registros en un rango de fechas *(Khan Academy, 2023)*.

En conclusión, el uso de estructuras de datos lineales adecuadas, como Linked List o Doubly Linked List, junto con algoritmos eficientes de búsqueda y ordenamiento, permiten manipular grandes volúmenes de datos, realizar búsquedas en rangos de fechas y mantener la organización cronológica de los registros. La elección de una Doubly Linked List proporciona operaciones eficientes de inserción, eliminación y búsqueda. La preferencia por la Doubly Linked List se basa en su eficiencia en ordenamiento, búsqueda y eliminación, gracias a su capacidad de acceso bidireccional en la lista. Esto garantiza un rendimiento óptimo al manipular y acceder a los registros de la bitácora.

Luis Carlos Rico Almada | A01252831

En el desarrollo de nuestra aplicación para resolver un problema específico (botnets), optamos por emplear una estructura de datos lineales, específicamente una Doubly Linked List (DLL). Esta decisión no fue tomada a la ligera, sino que fue el resultado de una profunda reflexión sobre la importancia y eficiencia de las DLL y su superioridad en ciertos contextos en comparación con una Linked List simple *(GeeksforGeeks, 2021)*.

Una de las principales ventajas de usar una DLL es su capacidad para navegar de manera eficiente en ambas direcciones. En el contexto de nuestra tarea, esto resultó particularmente útil para ordenar la información por fecha y hora, un proceso que puede requerir la revisión de nodos tanto anteriores como posteriores. Este nivel de flexibilidad no estaría disponible con una Linked List simple *(The Java Programmers, 2021).*
En términos de complejidad computacional, las operaciones de inserción y borrado en una DLL pueden realizarse en tiempo constante O(1) si se tiene un puntero al lugar de la operación *(GeeksforGeeks, 2021)*. Esto es significativamente más eficiente que en una Linked List, donde dichas operaciones pueden tomar tiempo lineal O(n) en el peor de los casos. Sin embargo, en ambos tipos de listas, la búsqueda puede tomar tiempo lineal O(n) en el peor de los casos *(Wolfram MathWorld, 2021)*.

Al aplicar esta eficiencia en el contexto de nuestro problema, pudimos ver un impacto directo en el rendimiento de nuestra solución. Por ejemplo, la DLL nos permitió implementar un algoritmo de búsqueda binaria para encontrar fechas de inicio y fin en nuestra bitácora, lo cual no hubiera sido tan eficiente con una Linked List simple debido a la necesidad de recorrer la lista en una sola dirección *(Carrano & Henry, 2016).*
El uso de una Doubly Linked List en nuestra solución fue preferible debido a su flexibilidad y eficiencia en operaciones clave. Aunque implicó un mayor uso de memoria para almacenar punteros adicionales, este costo fue compensado por las mejoras significativas en la eficiencia de tiempo *(GeeksforGeeks, 2021).* Es importante recordar que la elección de la estructura de datos siempre debe estar basada en el contexto específico y los requerimientos del problema a resolver. En otras situaciones, otras estructuras de datos podrían ser más apropiadas. Esta actividad me ha enseñado la importancia de esta consideración y seguiré reflexionando sobre ella en futuros retos de programación.

En términos de la complejidad computacional, se puede observar que lo que son la inserción y el borrado en una Doubly Linked List tienen una complejidad constante de O(1), mientras que la complejidad de búsqueda depende de la implementación utilizada, siendo O(n) para la búsqueda lineal y O(log n) para la búsqueda binaria. Estas características de complejidad contribuyen al desempeño óptimo y eficiente de las operaciones en la manipulación de la bitácora *(R, Ferris; J Albert, s.f)*.

En conclusión, la elección de utilizar una Doubly Linked List (DLL) en el desarrollo de la aplicación para resolver el problema de las botnets resultó ser una decisión acertada. Esta estructura de datos demostró ser superior en términos de eficiencia y flexibilidad en

comparación con una Linked List simple. Como resumen, la elección de una Doubly Linked List fue respaldada por las ventajas claras que ofrecía en términos de eficiencia de tiempo y flexibilidad en la manipulación de la información de las botnets. Esta experiencia destaca la importancia de considerar cuidadosamente la elección de la estructura de datos en función del contexto y los requisitos específicos del problema a resolver.

Referencias

- Carrano, F. M., & Henry, T. (2016). *Data Abstraction & Problem Solving with C++: Walls and Mirrors. Pearson. https://www.pearson.com/us/higher-education/product/Carrano-Data-Abstraction-Problem-Solving-with-C-Walls-and-Mirrors-7th-Edition/9780134463971.html*

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

- GeeksforGeeks. (2021). *Singly Linked List | Introduction. GeeksforGeeks. https://www.geeksforgeeks.org/data-structures/linked-list/*

- Wolfram MathWorld. (2021). *Doubly-Linked List. Wolfram MathWorld. https://mathworld.wolfram.com/Doubly-Linked-List.html*

- R, Ferris; J, Alberto (s.f) *Introducción al Estudio de Algoritmos y su Complejidad http://informatica.uv.es/iiguia/2000/AED/teoria/apuntes/cuatr2/AED.Tema.09.pdf*

- The Java Programmers. (2021). *Difference between Singly Linked List and Doubly Linked List. The Java Programmers. https://www.thejavaprogrammer.com/difference-between-singly-linked-list-and-doubly-linked-list/*

- Universidad de Los Andes. Facultad de Ingeniería. (s.f). *Escuela de Sistemas. Departamento de Computación. Estructuras de datos.* http://www.webdelprofesor.ula.ve/ingenieria/ibc/ayda/c9estLin.pdf

- *Doubly Linked List - javatpoint. (s.f). Doubly linked list* https://www.javatpoint.com/doubly-linked-list

- Merge Sort (With Code in Python/C++/Java/C). (2021). *Merge Sort Algorithm* https://www.programiz.com/dsa/merge-sort#:~:text=Merge%20Sort%20is%20one%20of,to%20form%20the%20final%20solution.

- Univerdad de Valladolid (s.f) *Introducción a la Teoría de la Complejidad Computacional* https://uvadoc.uva.es/bitstream/handle/10324/19054/TFG-G1789.pdf?sequence=1

- Khan Academy. (2023).*Búsqueda binaria (artículo) Algoritmos* https://es.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search