Tecnológico de Monterrey

Act 6.2 - Reflexión Final de Actividades Integradoras de la Unidad de Formación TC1031 (Evidencia Competencia)

Nikole Morales Rosas - A01782557

TECNOLÓGICO DE MONTERREY

Campus Santa Fe

Grupo: 570

Fecha de entrega: 28 de julio de 2023

Profesor: Dr. Eduardo A. Rodríguez Tello

Programación de estructuras de datos y algoritmos fundamentales

Reflexión Final

Introducción

A lo largo de este curso se desarrollaron ciertas competencias de estructuras de datos y algoritmos para poder abordar una situación problema para detectar redes infectadas o ataques cibernéticos. Se desarrollaron 5 actividades donde se utilizaron diferentes algoritmos para obtener el programa con mayor eficacia para poder resolver esta problemática.

Cada actividad ha explorado diferentes algoritmos y estructuras de datos con el objetivo de resolver diversos problemas en el campo de la informática.

- ➢ **Actividad 1.3: Algoritmos de ordenamiento y búsqueda**

Los algoritmos de ordenamiento y búsqueda son fundamentales para desarrollar sistemas, ya que permiten organizar la información según un criterio. Existen diversos tipos de estos algoritmos, cada uno con diferentes eficacias según el caso y la cantidad de datos. En esta actividad, se utilizaron dos algoritmos de ordenamiento, Bubble Sort y MergeSort, para comparar su rendimiento. Los resultados mostraron que MergeSort fue más rápido, realizando menos comparaciones.

“La complejidad temporal de Bubble Sort es $O(n^2)$, lo que lo hace menos adecuado para grandes cantidades de datos, ya que su tiempo de ejecución aumenta cuadráticamente. En cambio, MergeSort tiene una complejidad temporal de O(n log n), lo que lo hace más adecuado para grandes cantidades de datos.” (GeekForGeeks, s/f)

Para organizar una lista de registros de bitácoras de acceso a una red. Se utilizó una Binary Search que es un algoritmo de búsqueda para encontrar los elementos en la lista ordenada, su enfoque se basa en dividir repetidamente la lista a la mitad y comparar el valor del elemento buscado con el elemento central.

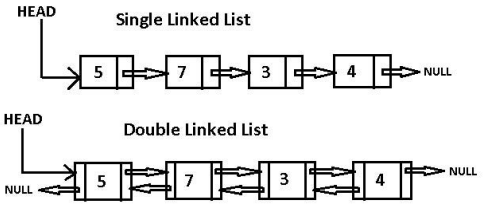
<u>Análisis de eficiencia:</u>

En términos de ordenación, el MergeSort demostró ser una de las opciones más eficientes con una complejidad temporal de O(n log n). Su capacidad para dividir y combinar listas de manera recursiva le proporciona un rendimiento estable en diferentes escenarios. Sin embargo, es importante tener en cuenta que puede requerir más espacio en memoria debido a la necesidad de almacenar listas temporales durante su proceso de fusión.

<u>Mejoras potenciales:</u>

En el caso del MergeSort, aunque su complejidad temporal es buena, se podría considerar una implementación iterativa en lugar de recursiva para reducir la sobrecarga asociada con las llamadas a funciones.

➢ Actividad 2.3: Estructura de datos lineales

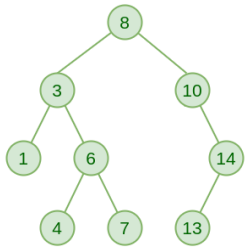


Las estructuras de datos lineales son eficientes para almacenar y organizar información en programación. Se utilizó una "double linked list" en esta actividad debido a su facilidad para insertar y eliminar datos. Las operaciones de inserción en la lista tienen complejidad O(1), mientras que el borrado tiene complejidad O(n) debido a que recorre todos los nodos. Para la búsqueda, se utilizó la búsqueda binaria con complejidad O(log n), que resultó conveniente para encontrar elementos específicos en la lista ordenada.

Análisis de eficiencia:
En las estructuras de datos lineales, las double linked lists mostraron ser una opción adecuada para la inserción y eliminación eficiente de elementos, con complejidad O(1) y para otros algoritmos se tuvo una complejidad de O(n). Sin embargo, su búsqueda lineal requiere recorrer todos los elementos de la lista, lo que puede volverse ineficiente para conjuntos de datos muy grandes.

Mejoras potenciales:
Para las double linked lists, se podría utilizar un índice adicional o un hash table para mejorar las operaciones de búsqueda y evitar recorrer toda la lista en cada búsqueda.

➢ Actividad 3.3: Binary Search Tree (BST)



“Las estructuras de datos jerárquicas son fundamentales para organizar datos en forma de árbol, permitiendo representar relaciones "padre-hijo". Al estar conectados de manera jerarquizada, facilitan el acceso y organización de la información de forma eficiente.” (UAM, 2013)

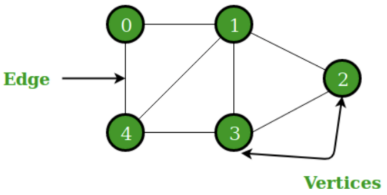
Para este programa se utilizó una estructura de datos jerárquica BST, junto con el algoritmo HeapSort para ordenar y almacenar registros de una bitácora. El HeapSort se implementó para ordenar la bitácora según su IP y se utilizó un *unordered_map* para contar la cantidad de accesos de cada dirección IP.

El proceso consistió en construir un Max Heap a partir del *unordered_map* para encontrar las 10 direcciones IP con mayor número de accesos. La complejidad del algoritmo HeapSort es O(n log n), mientras que la complejidad global del programa depende del tamaño del vector de registros y del Max Heap.

Análisis de eficiencia:
En cuanto al algoritmo HeapSort, se menciona que su complejidad temporal es de O(n log n), lo que lo convierte en una opción eficiente para ordenar los registros de la bitácora por IP. La complejidad computacional global del programa depende del tamaño del vector "*vectorRegistros*" y del Max Heap. Si el número de objetos en el vector es mayor que 10, la complejidad dominante es O(n) debido a la función "*cuentaAccesos*". Por otro lado, si el tamaño de "heapIp" es grande, la complejidad dominante sería O(log n) debido a la función "*obtieneMayorIp*".

Mejoras potenciales:
En la implementación tradicional de BST, la eliminación de un nodo puede requerir reorganizar el árbol, especialmente si el nodo tiene dos hijos. En algunos casos, esto puede ser costoso. Una mejora potencial es implementar una eliminación que simplemente marque el nodo como eliminado sin reorganizar el árbol de inmediato. Posteriormente, en una etapa de reorganización por lotes, se puede realizar la reorganización para mantener la estructura balanceada del árbol.

- **Actividad 4.3: Grafos**



Según se menciona en un análisis de la Universidad de Granada: los grafos son estructuras de datos no lineales que consisten en vértices y aristas que los conectan, permiten representar relaciones binarias entre elementos de un conjunto.
En esta actividad se utilizó una estructura de datos jerárquica conocida como grafo, compuesto por vértices y aristas, para almacenar y analizar información relacionada con direcciones IP y sus conexiones en una red. Se emplearon grafos dirigidos y no dirigidos según la naturaleza de las conexiones. Para el análisis, se utilizó una estructura de lista enlazada y un algoritmo HeapSort para ordenar y contar los grados de salida de los nodos. Se generaron archivos con los resultados, como *"grados_ips.txt"* y *"mayores_grados_ips.txt".*

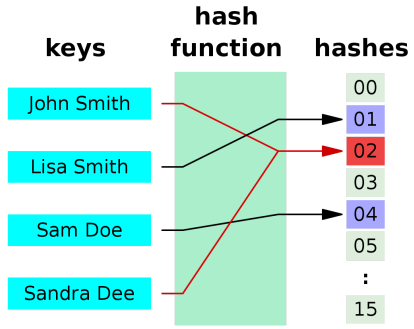
El algoritmo HeapSort tuvo una complejidad temporal de O(n log n), siendo más eficiente que otros métodos de ordenamiento cuadráticos como BubbleSort o InsertionSort. La representación gráfica de la red permitió detectar comportamientos sospechosos y encontrar caminos más cortos utilizando el algoritmo de Dijkstra.

Análisis de eficiencia:

Las estructuras jerárquicas como los grafos demostraron ser útiles para representar relaciones complejas y encontrar caminos más cortos, aunque algoritmos como el de Dijkstra tienen una complejidad temporal de O((n + m) * log(n)), lo que podría afectar el rendimiento en grafos de gran tamaño. El uso de grafos proporcionó una representación eficiente de los datos y facilitó la detección de patrones y nodos críticos. Además, permitió realizar análisis exhaustivos de la red para identificar posibles amenazas de seguridad y tomar decisiones informadas para protegerla.

Mejoras potenciales:
En el algoritmo de Dijkstra, se podría utilizar una implementación mejorada como la cola de prioridad de Fibonacci para reducir el tiempo de ejecución en grafos grandes.

➢ Actividad 4.3: Hashing



Las tablas hash son estructuras de datos asociativas que relacionan una llave con un valor mediante una función de hash, permitiendo un acceso rápido a los datos asociados con esas claves. “Estas tablas utilizan buckets para almacenar los elementos, y cuando ocurre una colisión, se utilizan técnicas de resolución de colisiones, como hashing abierto y cerrado, para manejarlas.” (UNIOVI, s/f)

En esta situación, se utilizó una tabla hash para almacenar un resumen de información sobre direcciones IP y sus conexiones en un grafo dirigido representado por una lista de adyacencias. El uso de la tabla hash permitió acceder a la información de manera eficiente, con una complejidad de O(1), evitando duplicación de datos y mejorando la administración de la información.

Las operaciones en la tabla hash, como inserción, búsqueda y eliminación, tienen una complejidad promedio de O(1) en ausencia de colisiones importantes. Sin embargo, en el peor caso, cuando hay muchas colisiones, la complejidad puede acercarse a O(n). Por lo tanto, la elección adecuada de la función de hash y la técnica de resolución de colisiones es crucial para mantener un rendimiento óptimo.

En general, el uso de tablas hash en esta situación proporcionó una manera eficiente de manejar y acceder a la información relacionada con las direcciones IP y sus conexiones, lo que contribuyó al análisis y detección de patrones de comportamiento en el grafo dirigido. Sin embargo, es importante considerar factores como la función de hash y la cantidad de colisiones para asegurar un rendimiento óptimo en todas las operaciones.

Análisis de eficiencia:
En cuanto a las tablas hash, se presentan como una opción eficiente para acceder a la información asociada con las claves, con una complejidad promedio de O(1) en operaciones de búsqueda y eliminación, siempre y cuando no haya muchas colisiones. Sin embargo, la gestión de colisiones puede afectar la eficiencia, y al aumentar el número de colisiones, lo que significa que en el peor caso la complejidad computacional puede acercarse a O(n).

Mejoras potenciales:
En las tablas hash, se podría considerar una función de hash más robusta para reducir las colisiones, así como evaluar otras técnicas de resolución de colisiones para mejorar el rendimiento en el peor caso, de ahí en fuera fue una de las estructuras más convenientes para la resolución de esta situación problema.

Análisis de eficiencia general
Por otro lado, para esta situacion problema podemos observar que dentro de cada actividad había un algoritmo más eficiente o más óptimo que otro dependiendo de cada estructura de datos que usamos, sin embargo la estructura que considero más eficaz para una situación problema de esta naturaleza es la de Hashing, al compararlas con las estructuras usadas en las otras actividades, las tablas hash resultaron ser más óptimas para tener una mejor administración y mantenimiento de los datos, ya que se pueden realizar operaciones de inserción, búsqueda y eliminación de manera eficiente.

Conclusión
En esta unidad de formación se exploraron distintas técnicas y estructuras que son importantes para resolver problemas informáticos, como en este caso que se tuvo que crear un programa para detectar ataques cibernéticos. A lo largo de cada actividad se logró apreciar como se puede abordar esta situación problema con distintos algoritmos y estructuras de las cuales va a depender la eficiencia del programa.

Personalmente me gustó aprender sobre las estructuras de datos no lineales, me parece muy interesante cómo es que funcionan, así como también los algoritmo que se implementan en cada estructura.
En cuanto a aspectos que podrían mejorarse, considero que se podrían incluir más ejercicios para saber cómo solucionar un problema para luego pasar a la implementación del código.

Fuera del contexto de los algoritmos y estructuras de datos vistos en clase, logré darme cuenta de que es muy importante buscar la manera de hacer óptimo el programa, es decir, tomar en cuenta la complejidad computacional para elegir los algoritmos que se adapten mejor a las necesidades del programa.

Referencias

Sin Autor. *Hashing data structure*. (2015, diciembre 14). GeeksforGeeks. Recuperado el 27 de julio de 2023 en: https://www.geeksforgeeks.org/hashing-data-structure/

Cruz, M. (2013, marzo 4). *Método de Dispersión (Hashing) en C++*. Martincruz.me. Recuperado el 27 de julio de 2023 en: https://blog.martincruz.me/2013/03/metodo-de-dispersion-hashing-en-c.html

Sin Autor. (2023, abril 24). *Hash table data structure.* GeeksforGeeks. Recuperado el 27 de julio de 2023 en: https://www.geeksforgeeks.org/hash-table-data-structure/

Sin Autor. (2022, julio 4). *Introduction to hashing - data structure and algorithm tutorials.* GeeksforGeeks. Recuperado el 27 de julio de 2023 en: https://www.geeksforgeeks.org/introduction-to-hashing-data-structure-and-algorithm-tutorials/

Sin Autor. GRAFOS. (s/f). Ugr.es. Recuperado el 28 de julio de 2023, de https://ccia.ugr.es/~jfv/ed1/c++/cdrom4/paginaWeb/grafos.htm

Tait, J., Ripke, T., Roger, L., & Matsuo, T. (2018, December). Comparing Python and C++ efficiency through sorting. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)* (pp. 864-871). IEEE. Recuperado el 28 de julio de 2023 en: https://ieeexplore.ieee.org/abstract/document/8947654

Arifin, R. W., & Setiyadi, D. (2020). Algoritma Metode Pengurutan Bubble Sort dan Quick Sort Dalam Bahasa Pemrograman C++. *INFORMATION SYSTEM FOR EDUCATORS AND PROFESSIONALS: Journal of Information System*, *4*(2), 178-187. Recuperado el 28 de julio de 2023 en: http://www.ejournal-binainsani.ac.id/index.php/ISBI/article/view/1348

C++ program for merge sort. (2022, diciembre 15). GeeksforGeeks. Recuperado el 28 de julio de 2023 en: https://www.geeksforgeeks.org/cpp-program-for-merge-sort/

Bubble sort - data structure and algorithm tutorials. (2014, febrero 2). GeeksforGeeks. Recuperado el 28 de julio de 2023 en: https://www.geeksforgeeks.org/bubble-sort/

Aguilar, L. J., & García, L. S. (2006). *Programación en C++: un enfoque práctico*. McGraw-Hill Interamericana de España. Recuperado el 28 de Julio de 2023 en: http://repositoriokoha.uner.edu.ar/fing/pdf/5230.pdf

Whitney, T. (2023). *Destructores (C++)*. Microsoft. Recuperado el 28 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 28 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 28 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 28 de julio de 2023.

UAM. (2015). *Estructura de Datos: Estructura de Datos Lineales.* México. Recuperado el 28 de julio de 2023 de: http://ri.uaemex.mx/oca/view/20.500.11799/34615/1/secme-19001.pdf

Sin Autor. *Heap sort - data structures and algorithms tutorials.* (2013, marzo 16). GeeksforGeeks. Recuperado el 28 de julio de 2023 en: https://www.geeksforgeeks.org/heap-sort/

Anónimo. Bases de datos jerárquicas ¿Qué son? Ejemplos. (2020, septiembre 9). *Ayuda Ley Protección Datos; AyudaLeyProteccionDatos.* Recuperado el 28 de julio de 2023 en: https://ayudaleyprotecciondatos.es/bases-de-datos/jerarquicas/

Sin Autor. *Unordered_map in C++ STL.* (2016, marzo 24). GeeksforGeeks. Recuperado el 28 de julio de 2023 en: https://www.geeksforgeeks.org/unordered_map-in-cpp-stl/

Anónimo. std::unordered_map. (s/f). Cplusplus.com. Recuperado el 28 de julio de 2023 en: https://cplusplus.com/reference/unordered_map/unordered_map/

Anónimo. Uniovi. (s/f). *Grafos. 6.1 Conceptos previos y terminología*. España. Recuperado el 28 de julio de 2023 en: https://www6.uniovi.es/usr/cesar/Uned/EDA/Apuntes/TAD_apUM_07.pdf

Varios Autores. *Heap sort - data structures and algorithms tutorials.* (2013, marzo 16). GeeksforGeeks. Recuperado el 28 de julio de 2023 en: https://www.geeksforgeeks.org/heap-sort/

Sin Autor. *Graph data structure and algorithms.* (2016, mayo 17). GeeksforGeeks. Recuperado el 28 de julio de 2023 en: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/

Sin Autor. *Graph and its representations.* (2012, noviembre 13). GeeksforGeeks. Recuperado el 28 de julio de 2023 en: https://www.geeksforgeeks.org/graph-and-its-representations/