Tecnológico de Monterrey

Reflexión final de la unidad de formación TC1031

Saúl Castañeda Carrillo A01541099

TC1031: Programación de estructuras de datos y algoritmos fundamentales

Grupo 701

16 de Junio del 2022

Los ataques informáticos se han vuelto aspectos cotidianos de la vida de un ser humano, pues vivimos en un mundo globalizado y en constante evolución. Por consiguiente es importante el poder prevenir o el detectar anomalías dentro de la red. Por eso es que durante este curso se nos planteó el objetivo de reconocer o resolver problemas alrededor del supuesto de un ataque informático. La resolución a estos problemas, la dábamos mediante el uso de distintas propiedades de las estructuras de datos que se nos mostraron durante el curso, siendo capaces de aprender el uso, características y debilidades de estas a partir de su implementación.

Reflexiones de Actividades Integrales (Evidencia competencia):

Act 1.3 - Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales:
Consideró que los métodos de búsqueda son de gran ayuda pues al tener bases de datos inmensas facilitan la búsqueda de datos. Las formas de orden también nos ayudan pues facilitan la lectura de los datos, por qué podemos identificar si algún dato está fuera de lugar y poder modificarlo. Por último el concepto del manejo de base datos es interesante pues puedes obtener información relevante sobre el proyecto, pues puedes deducir ciertos aspectos del comportamiento de los datos basados en la forma de orden y búsqueda.

Act 2.3 - Actividad Integral estructura de datos lineales:
Dado que en la programación existen distintos tipos de algoritmos de ordenamiento, se ha de utilizar el que en este caso nos genere una mayor eficacia en el programa. Por tal motivo decidimos utilizar un quickSort para el ordenamiento de nuestro archivo “Bitacora.txt”, el cual contiene datos de tipo Registro (clase), y al no ser una cantidad de datos exuberante. Optamos por usarlo debido a que, aunque el Merge sort realice menos comparaciones, requiere de un espacio de memoria adicional de O(n) el cual almacena la matriz adicional, mientras que en el quickSort necesita un espacio de O(registro n).

Y para este caso en específico, nos es mejor utilizar una Doubly Linked List, puesto que tenemos acceso al apuntar “prev”, el cual nos permite acceder, como su nombre lo dice, al nodo previo con el que estamos trabajando. Algo que nuestra Linkedlist no nos proporciona. Por consiguiente, esto para el ordenamiento mediante un quickSort iterativo, nos fue muy conveniente, porque pudimos trabajar con el adaptador de contenedor denominado Stack, el cual le da al programador la funcionalidad de una pila, específicamente, una estructura de datos LIFO(Last In, First Out). Y así hacer el ordenamiento de nuestros datos de tipo Registro, mediante apuntadores.

Y dado que en las estructuras de datos lineales, se involucra un “single level inheritance”. El cual nos permite recorrer todos los elementos en una sola ejecución. Y a su vez son fáciles de implementar porque la memoria de la computadora está organizada de forma lineal, lo cual hace más eficiente el uso de este tipo de estructura de datos para este caso en específico

Act 3.4 - Actividad Integral de BST :
Para la solución de este reto, fue de suma importancia el saber jugar con los diferentes tipos de estructuras de datos, pues utilizamos un HeapSort con un vector con la finalidad de acomodar nuestros datos con base a su IP. Pudiendo así trabajar con una base de datos bastante extensa, que de no haber sido por la implementación que utilizamos en este

HeapSort, probablemente hubiese utilizado mayor capacidad computacional para su ejecución.

Ya que pudimos realizar este ordenamiento, procedimos a hacer uso de una estructura BST, pues nos ordena nuestra base de datos de forma eficiente ya que cuenta con una complejidad de O(log2 N), colocando en la cúspide del mismo, el IP el cual tuvo mayor cantidad de accesos, y siendo sus hermanos (Left y Right) los siguientes IP 's con mayor cantidad de accesos y así sucesivamente. Aspecto que nos ayudó una vez que creamos la lista de las 5 IP 's con mayor cantidad de accesos. Este ejercicio con el BST se hace con la finalidad de conocer que IP's habían provocado la mayor cantidad de accesos, y de tener un control de esto, y nos permite el preguntarnos si dicho acceso desde tal IP, es un cliente recurrente del servidor, o si se trata de un bot, o incluso de un ataque DDOS, tomando en cuenta: el IP, la fecha en que accedió y el número de accesos.

En conclusión, la implementación de distintas estructuras de datos, nos provee de un código tanto más limpio, como más eficiente, y al hablar de bases de datos extensas, es algo que se ha de tomar en cuenta.

Act 4.3 - Actividad Integral de Grafos

Dado que buscamos establecer una relación entre nuestros nodos (IP 's) y un nodo en concreto (boot master), es de mucha utilidad trabajar con una estructura de datos de tipo grafo, más específicamente un grafo ponderado. Pues nos permite establecer dicha relación, mediante un conjunto de arcos, a los cuales se les asigna un peso. Y dado que buscamos encontrar el bot master, es correcto asumir que la IP, la cual tenga la mayor cantidad de arcos, es quien es la que está detrás del ataque, pues es quien más outputs está generando. De igual forma, el hecho de conocer este bot master, y del tener asignado un peso (para este caso representa la distancia) a cada uno de nuestros arcos, nos permite calcular cuáles IP son más o menos vulnerables, dependiendo de la distancia a la que se encuentren del bot Master. Esto fue logrado gracias al algoritmo de Dijkstra con una complejidad de O(|E| log |V|), lo cual para grandes cantidades de datos viene muy bien pues es una complejidad no muy pesada.

De igual forma este código representa una forma relativamente eficaz de solucionar la problemática puesto que, en el ordenamiento con priority queue del total de los arcos de cada nodo, cuenta con una complejidad O(nlogn), lo que se asume como aceptable. Pero mejora este ordenamiento cuando usamos Dijkstra para ordenar los nodos y la distancia al botmaster, con una complejidad de O(|E| log |V|), lo cual se asume que es más que aceptable. Y también tenemos la función de nuestra búsqueda de información, que nos proporcionan complejidad O(|E| + |V|), que se asume que ya representa una complejidad buena. (Estas asunciones sobre la categorización de las complejidad se baso del cheat sheet de “Big-O Cheat Sheet” link: https://www.bigocheatsheet.com/)

Act 5.2 - Actividad Integral sobre el uso de códigos hash

La estructura de tipo hash, es una herramienta sumamente útil cuando se requiere obtener un valor mediante una búsqueda. Pues al contrario de lo que haríamos con un array, que sería recorrerlo un valor a la vez, es algo ineficiente y que con grandes cantidades de datos,

se volvería algo básicamente obsoleto. La particularidad del hash es que mediante un key se puede acceder de forma directa a los datos que trae consigo este key, pudiendo ser información de cualquier tipo desde un entero o string, hasta clases, entre otras opciones. Sin duda es una gran ventaja también el tener una complejidad relativamente pequeña (O(1+alpha) donde alpha es igual a n/m), lo que nos permite ser sumamente eficientes con los métodos de búsqueda por ejemplo. También me gustaría destacar algo que me pareció interesante; el poder utilizar tanto un método de dirección cerrado como abierto, y el poder definir una función hash cualquiera, considero que esto abre un sin fin de posibilidades para poder jugar con el hash y eficientar lo más posible dado sea el caso. Pues puedes generar una función hash que provoque menos colisiones por ejemplo, o también la capacidad de utilizar el método de dirección cerrado, y, mediante linked list enlazar los nodos sobre un mismo index, omitiendo así la función hash para encontrar un espacio vacío.

Conforme nos acercamos a la recta final del curso, fuimos capaces de realizar actividades con un mayor grado de dificultad, lo que nos abrió el panorama para hacer mejoras a entregables pasados, pues aprendimos conceptos valiosos y aplicables que pudiésemos implementar en algunos de los entregables con el fin de eficientar alguna función o bien mejorar el código. Con cada actividad pudimos explorar a fondo tanto la implementación, como las utilidades de cada estructura de datos, pues al profundizar en cada una de ellas, nos permitió conocer en qué tipo de casos se puede utilizar, sus ventajas y/o desventajas sobre otras estructuras e incluso haciendo hincapié en el aprovechamiento de la eficiencia que presentan cada una de ellas para cierto tipo de tareas muy específicas. Esto, teniendo en cuenta la complejidad computacional que presentaba cada una de las estructuras para ciertos casos (búsqueda, ordenamiento, eliminación o introducción de datos), y creando así un criterio sobre la implementación del código de estas estructuras y el por qué optamos por esta.