Tecnológico de Monterrey

Act 3.3 – Actividad Integral de BST (Evidencia Completa)

Programación de Estructuras de Datos y Algoritmos Fundamentales

Instituto Tecnológico y de Estudios Superiores de Monterrey

Fecha de entrega: 22 de julio de 2023

Estudiante: Alyson Melissa Sánchez Serratos - A01771843

Profesor: Dr. Eduardo Arturo Rodríguez Tello

REFLEXIÓN: ACTIVIDAD INTEGRAL 3.3

En el mundo digital actual, donde la información fluye y se genera a velocidades descontroladas y exorbitantes, el análisis de grandes volúmenes de datos se ha convertido en una tarea esencial para tomar decisiones informadas en diversas áreas, como la seguridad informática. En este aspecto, las estructuras de datos jerárquicas juegan un papel crucial al permitir manejar, organizar y analizar eficientemente la información, especialmente en situaciones donde se necesita identificar patrones, detectar anomalías y responder a problemas complejos.

Una de las aplicaciones más relevantes del uso de estructuras de datos jerárquicas se encuentra en el análisis de accesos de IPs a servidores o sistemas, donde es fundamental clasificar, ordenar, y contabilizar los accesos de manera eficiente para detectar actividades sospechosas, identificar patrones de comportamiento y mejorar la seguridad. Para el cumplimiento adecuado de dicha tarea, es fundamental poseer conocimientos sólidos de algoritmos de ordenamiento, particularmente del algoritmo Heap Sort.

El algoritmo Heap Sort se basa en la construcción y mantenimiento de una estructura de datos conocida como "heap". Un heap, en el contexto de estructuras de datos y algoritmos, es una estructura de datos jerárquica que se basa en un árbol binario completo con una propiedad especial. En un heap, cada nodo tiene un valor asociado y satisface una propiedad de orden la cual, de acuerdo con Srivastava, P. (2020), se puede expresar de dos formas dependiendo del tipo de heap:

- **Max Heap:** En un Max Heap, el valor de cada nodo es mayor o igual que los valores de sus nodos hijos. En otras palabras, el valor máximo está en la raíz del árbol binario y cada nodo padre es mayor o igual a sus nodos hijos.

- **Min Heap:** En un Min Heap, el valor de cada nodo es menor o igual que los valores de sus nodos hijos. En este caso, el valor mínimo se encuentra en la raíz del árbol binario y cada nodo padre es menor o igual a sus nodos hijos.

Esta propiedad única del heap permite realizar un ordenamiento eficiente de los datos en tiempo O (n log n), donde "n" representa el número de elementos a ordenar.

Cuando se aplica Heap Sort al análisis de bitácoras de acceso, en primer lugar, se debe construir un Max Heap a partir del vector de registros. Esto asegura que el elemento más grande del vector (según la clave seleccionada, en este caso, la dirección IP) esté en la posición de la raíz.

Para ello se inicializa un for que recorre los índices de los nodos padres del heap. La idea es empezar desde el último nodo padre y descender hasta el primer nodo del vector. En cada iteración, se llama a la función *heapifyHelper* para asegurar que el subárbol con raíz en el nodo actual cumpla con la propiedad del Heap Max, es decir, que el valor del nodo sea mayor o igual que los valores de sus nodos hijos. Este proceso de construcción del Heap Max se realiza de manera eficiente y se garantiza que el elemento más grande esté en la raíz del árbol.

Después de completar el primer bucle for, el vector vectorRegistros se encuentra organizado como un Heap Max, con el registro en la dirección IP más grande en la posición de la raíz. Sin embargo, para obtener la lista ordenada final ascendentemente, es necesario extraer el elemento más grande y colocarlo al final del vector. Esto se logra mediante el segundo bucle for en la función *heapSort*. En cada iteración, el algoritmo intercambia el primer elemento del vector (el más grande) con el elemento actual que se está evaluando (el último), y luego se llama a *heapifyHelper* para reorganizar el Heap y encontrar el nuevo elemento más grande. Este proceso se repite hasta que todos los elementos se hayan extraído del Heap, dejando el vector completamente ordenado.

Una vez que se ha aplicado Heap Sort y se ha obtenido el registro ordenado ascendentemente por el valor de las direcciones IPs, se pueden realizar diversas operaciones analíticas. Una de las más relevantes es contar eficientemente la cantidad de accesos para cada IP. Para ello fue necesaria la utilización de un *unordered map*, una implementación la cual hace uso de las Tablas Hash. De acuerdo con la página oficial de cplusplus (s/f):

> Los mapas desordenados son contenedores asociativos que almacenan elementos formados por la combinación de un valor clave y un valor mapeado, y que permite la recuperación rápida de elementos individuales en función de sus claves.
>
> En un mapa desordenado, el valor clave generalmente se usa para identificar de manera única el elemento, mientras que el valor asignado es un objeto con el contenido asociado a esta clave.

En el proyecto integrador realizado, se optó por aprovechar las propiedades de los *unordered maps* específicamente para contar eficientemente la cantidad de accesos para cada IP. El proceso de conteo se llevó a cabo mediante un recorrido por la lista de registros ordenada por IPs, y en cada iteración, se insertaba la IP como clave en la dicha estructura. A continuación, en la **tabla 1** se explican los detalles de cómo se implementó este proceso de conteo utilizando la estructura de datos *unordered map*.

Tabla 1 Pasos para obtención de conteos de apariciones por IP

Paso 1	Tras aplicar el algoritmo de ordenamiento Heap Sort a la bitácora de acceso, se logró obtener una lista ordenada en forma ascendente por las direcciones IP. A continuación, utilizando el método getIp (), que extrae la dirección IP de cada registro, se procedió a realizar un conteo eficiente de la cantidad de accesos para cada una de ellas.
Paso 2	Se creó un *unordered map* llamado “contador” donde las claves son las IPs y los valores asociados representan la cantidad de apariciones de cada IP en la bitácora de acceso.
Paso 3	En cada iteración del recorrido por el vector de registros ordenados por IPs, se realizaba una inserción o actualización en el *unordered map*.

	Si la IP extraída a través de getIp () ya existía como clave en el mapa, su valor asociado se incrementaba en uno para reflejar la cantidad de repeticiones de la IP. Por otro lado, si la IP era nueva (no se había encontrado previamente en la bitácora), se agregaba como una nueva clave con un valor inicial de 1, indicando que se encontró una aparición de esa IP.
Paso 4	Posteriormente, se construyó un Max Heap de objetos de la clase AccessTracker la cual posee como atributos, la dirección IP, y el número de apariciones en la bitácora. Esto permitió identificar de manera competente las direcciones más activas o con mayor número de intentos de acceso al sistema o a la red.

Como se observa, la utilización de una estructura de datos basada en Tablas Hash, como el *unordered map*, brindó una solución eficiente y sencilla de implementar para contar la cantidad de accesos para cada dirección IP en la bitácora de accesos. La elección de esta estructura se fundamentó en varias consideraciones clave que optimizaron el proceso de análisis de la bitácora y la construcción del Heap Tree.

En primer lugar, la complejidad promedio constante O (1) de inserción y actualización individual del *unordered map* permitió contar eficientemente las apariciones de cada IP en tiempo lineal O (n), donde "n" es el número de registros en la bitácora. Esto resultó esencial para trabajar con grandes conjuntos de datos, permitiendo un conteo rápido de las IPs más activas y evitando operaciones costosas de búsqueda o inserción en estructuras más complejas.

Asimismo, el uso de un *unordered map* en lugar de un contenedor ordenado, como un *map*, fue una elección adecuada debido a que el objetivo final era construir un Heap Tree ordenando los elementos por cantidad de apariciones. De acuerdo con la página oficial de cplusplus (s/f). “Los mapas normalmente se implementan como árboles de búsqueda binarios.” Sin embargo, al buscar almacenar la información en una estructura de tipo Max Heap, se consideró innecesario y temporalmente más costoso el uso de un mapa.

Como el *unordered map* no garantiza un orden específico de las claves, el programa ahorró tiempo al no requerir una fase previa de ordenamiento antes de insertar las IPs en el Heap Tree. Esta ventaja es especialmente relevante cuando el orden exacto no es crítico para dicha operación y se prioriza el rendimiento en términos de tiempo de ejecución.

Al utilizar esta herramienta, se logró un análisis preciso y rápido de las IPs más activas, lo que facilitó la detección de patrones y la identificación de comportamientos sospechosos. Este procesamiento de información permite obtener pistas valiosas sobre posibles actividades maliciosas, incluyendo indicios sobre si una red podría estar infectada o no. A continuación, en la **tabla 2** se muestran algunas pautas las cuales podrían ayudar a determinar la infección de una red basándose en el conteo de accesos de una dirección IP.

Tabla 2 Lista de posibles vías de ataque de red basada en el número de accesos de una IP

Caso 1	Si una dirección IP intenta acceder repetidamente a una red, esto podría indicar un intento de fuerza bruta para obtener acceso no autorizado. El análisis de estos patrones inusuales puede ayudar a identificar posibles ataques.
Caso 2	En el hipotético caso de que se contara con una lista de direcciones IP maliciosas conocidas, el análisis podría revelar si alguna de estas IPs está intentando acceder al sistema. Esto podría indicar que la red está siendo atacada por actores maliciosos.
Caso 3	Comparar la cantidad de intentos de acceso actuales con registros históricos de actividad puede ayudar a detectar aumentos significativos o anomalías en el tráfico de accesos, lo que podría indicar una infección o ataque en curso.

En conclusión, se puede afirmar que las estructuras de datos jerárquicas, como el Max Heap utilizado en este contexto, son imprescindibles al buscar detectar anomalías o actividades maliciosas en el acceso de una red. Estas estructuras proporcionan una organización eficiente de la información, permitiendo un análisis rápido y efectivo de grandes volúmenes de datos. La combinación de algoritmos de ordenamiento, como el Heap Sort, con la capacidad de contar eficientemente las apariciones de cada dirección IP mediante el uso de un *unordered map*, es una estrategia poderosa para identificar patrones sospechosos o inusuales en la actividad de la red.

Fuentes de información:

Maps. (s/f). Cplusplus.com. Recuperado el 23 de julio de 2023, de https://cplusplus.com/reference/map/map/

Srivastava, P. (2020, noviembre 6). Difference between min heap and max heap. GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-min-heap-and-max-heap/

Std::Unordered_map. (s/f). Cppreference.com. Recuperado el 22 de julio de 2023, de https://en.cppreference.com/w/cpp/container/unordered_map