Valeria Salazar Pinto
A01736652

Reflexión sobre el uso de Estructuras de Datos Jerárquicas y Eficiencia Algorítmica

Hoy en día, la ciberseguridad es uno de los pilares fundamentales de la tecnología. Con el aumento exponencial de dispositivos conectados, los servidores reciben millones de peticiones por segundo, y distinguir entre tráfico legítimo y un ataque (como un DDoS) se ha vuelto una tarea crítica. No basta con almacenar la información; es necesario procesarla, ordenarla y priorizarla en tiempo real.En esta actividad, nos enfrentamos a una situación problema donde una bitácora de servidor registra accesos que podrían indicar una intrusión. El reto no fue solo leer el archivo, sino procesarlo eficientemente para encontrar las IPs con mayor número de accesos. Para lograr esto de manera óptima, fue necesario salir de las estructuras lineales y utilizar estructuras de datos jerárquicas, específicamente el Binary Heap (Montículo Binario), tanto para el ordenamiento (Heap Sort) como para la selección de los elementos más frecuentes (Priority Queue).

En esta reflexión voy a analizar la importancia de estas estructuras, por qué un Heap fue la elección correcta sobre otras opciones como un BST, y cómo la complejidad computacional define el éxito de una solución en un entorno de seguridad real.

Importancia de las estructuras jerárquicas en este contexto

Cuando trabajamos con grandes volúmenes de datos en una bitácora, las estructuras lineales (como listas o arreglos simples) suelen ser ineficientes para ciertas tareas. Si quisiéramos encontrar el elemento mayor en un arreglo desordenado una y otra vez, tendríamos que recorrerlo completo múltiples veces.Aquí es donde entran las estructuras jerárquicas.

En la solución implementada, el uso de un Binary Heap permitió organizar la información no por posición, sino por prioridad. Esto es crucial en ciberseguridad: no nos interesa necesariamente quién llegó primero (orden cronológico), sino quién está generando más tráfico (orden de prioridad). Al utilizar std::priority_queue para filtrar el "Top 10", logramos que el sistema se enfoque automáticamente en los datos más relevantes sin desperdiciar recursos.

Reflexión: ¿Por qué un Binary Heap y no un BST? Una de las decisiones de diseño más importantes en este reto fue preferir un Binary Heap sobre un Binary Search Tree (BST). A primera vista, ambos son árboles y parecen similares, pero su propósito es distinto:Objetivo de la estructura: Un BST está diseñado para búsquedas arbitrarias (encontrar cualquier elemento rápidamente). Sin embargo, en este reto no necesitábamos buscar una IP específica aleatoria sino que necesitábamos extraer repetidamente el máximo (la IP con más accesos). El Binary Heap está optimizado matemáticamente para encontrar el extremo (O(1)) mucho mejor que un BST

Eficiencia en memoria y caché

En la implementación, el Heap Sort y la Priority Queue trabajan sobre arreglos (vectores) de manera implícita. Esto hace que los datos estén contiguos en la memoria RAM, mejorando la velocidad de acceso (cache locality). Un BST, por el contrario, utiliza nodos dispersos conectados por punteros, lo cual fragmenta la memoria y suele ser más lento en la práctica

Valeria Salazar Pinto
A01736652

para este tipo de operaciones.Costo de balanceo: Para que un BST sea eficiente, debe estar balanceado (como un AVL). Mantener ese balanceo tras cada inserción es costoso.

El Heap, por su naturaleza de árbol completo, siempre mantiene su forma óptima sin necesidad de algoritmos de rotación complejos.

Análisis de Complejidad y Desempeño

Algo importante de esta actividad es que la teoría se vio reflejada en el código. Analizando las operaciones de la estructura Binary Heap (implementada en la priority_queue y en el Heap Sort manual), podemos ver su impacto:Obtener el máximo (top): Tiene una complejidad de O(1). Esto significa que saber cuál es la IP más peligrosa es una operación instantánea, sin importar si tenemos 100 o 1 millón de registros.Insertar y Eliminar (push pop): Ambas operaciones tienen una complejidad de O(log n). Al insertar un registro, el algoritmo solo necesita recorrer la altura del árbol (que es logarítmica) para acomodarlo.En la práctica, esto impacta el desempeño drásticamente. Si hubiéramos usado un ordenamiento simple como Bubble Sort (O(n2)) para sacar el Top 10, el programa podría haber tardado minutos en procesar una bitácora grande.

Además de Heapsort, existen otros algoritmos de ordenamiento que podrían aplicarse en este tipo de problema, como Quicksort y Mergesort.

Quicksort tiene una complejidad promedio de O(n log n), pero en el peor caso puede llegar a ser O(n2) si los pivotes no se eligen bien. Mergesort, en cambio mantiene siempre una complejidad de O(n log n), pero la desventaja es que utiliza memoria adicional porque no ordena en el lugar. En comparación, Heapsort también es O(n log n) en todos los casos, pero es más eficiente en cuanto al uso de memoria.

En cuanto a algoritmos de búsqueda, una opción sería la búsqueda binaria, que tiene complejidad O(log n) pero requiere que los datos estén previamente ordenados. En estructuras como listas enlazadas, la búsqueda secuencial tiene una complejidad de O(n), lo que la hace menos eficiente para grandes volúmenes de datos.En este caso, el uso del heap no solo ayuda a mantener el orden de prioridades (por ejemplo, IPs más grandes al tope), sino que también simplifica operaciones como obtener el elemento más importante (getTop), lo cual sería más costoso en otras estructuras.

Con Heap Sort y Priority Queue (O(n log n)), el procesamiento es casi inmediato, lo cual es muy importante para detener un ataque antes de que colapse el servidor.Detección de amenazas: ¿Cómo saber si la red está infectada?Más allá del código, esta actividad permite reflexionar sobre la seguridad de redes. Basándome en los datos obtenidos en el archivo ips_con_mayor_acceso.txt, podría determinar si la red está infectada observando lo siguiente

Frecuencia desproporcionada: Si una sola IP tiene miles de accesos en comparación con el promedio (que podrían ser decenas), es un indicador claro de un bot intentando fuerza bruta.

Patrones temporales: Si los accesos ocurren en milisegundos de diferencia (como se ve al ordenar la bitácora), no es un humano, es un script automatizado.Puertos destino: Si las IPs

Valeria Salazar Pinto
A01736652

del "Top 10" están atacando puertos inusuales (distintos al 80 o 443), es una señal de escaneo de vulnerabilidades.

Conclusión

Esta actividad me ayudó a entender que elegir la estructura de datos correcta es tan importante como escribir el código. En situaciones críticas como la detección de intrusos, la diferencia entre usar un Binary Heap o una lista desordenada puede ser la diferencia entre detener un ataque a tiempo o que el sistema falle.Aprendí que la complejidad computacional no es solo un concepto teórico, sino una herramienta de predicción. El uso de algoritmos O(n log n) como Heap Sort garantiza que nuestra solución sea escalable y profesional.

Referencias:

-Codecademy. (s. f.). *What is a Max-Heap? Complete Guide with Examples*. Codecademy. https://www.codecademy.com/article/max-heap

-Tech, M. (s. f.). *Mentores Tech - Acelera tu Carrera en Tecnología*. Mentores Tech. https://www.mentorestech.com/resource-algorithms-heap-sort.php

-Speck, J. (2020, May 15). *Priority Queue O(log n) via Heap Sort - John Speck - Medium*. Medium. https://medium.com/@eltocino/priority-queue-o-log-n-via-heap-sort-8987d52c21ca

-ryan. (2024, October). *C++ priority_queue: Practical Guide*. Medium. https://medium.com/@ryan_forrester_/c-priority-queue-practical-guide-cf5291bc1fcd

-Martinezagarza, R. (2022, January 21). *¿Por qué es tan Importante la Ciberseguridad Hoy en Día?* Centromexico.digital. https://centromexico.digital/por-que-es-tan-importante-la-ciberseguridad-hoy-dia/