Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Programación de estructuras de datos y algoritmos fundamentales (Gpo 573)

Profesores: Dr. Eduardo Arturo Rodríguez Tello

Andres Romo Castañeda A01234579

Act 3.3 - Actividad Integral de BST (Evidencia Competencia)

28/01/2026

Monterrey, Nuevo León

Reflexión Individual - Actividad 3.3

Recuerdo que cuando trabajamos en la Actividad 1.4, me acordé de cómo el tiempo de ejecución es un recurso real y limitado. En aquel entonces, ver la diferencia entre un ordenamiento cuadrático y uno logarítmico al manejar la bitácora me llamó mucho la atención y me ayudó a entender más sobre las complejidades. Sin embargo, en esta nueva entregable con la Actividad 3.3 subió la dificultad y la verdad es que se sintió fuerte el cambio, ya que necesitábamos jerarquizar y priorizar la información de manera dinámica.

En esta actividad, trabajamos con una bitácora de accesos donde la IP era nuestra llave principal. El primer problema fue el ordenamiento. Implementamos heapsort, un algoritmo que, al igual que el Merge Sort que usamos antes, mantiene una complejidad de O(n log n). Lo interesante aquí fue ver cómo este algoritmo aprovecha una estructura de árbol para organizar los datos. Al igual que en la entrega pasada, recurrimos a la sobrecarga de operadores en la clase Registro. Si no hubiéramos hecho esto, comparar las IPs habría sido un desorden total de strings. Al final, logramos que el programa entendiera cómo comparar objetos complejos de forma natural, permitiendo que el Heap Sort hiciera su trabajo de manera eficiente.

Donde tuvimos que dar nuestro mayor tiempo y pensamiento en el desarrollo del código fue cuando pasamos al Binary Heap para obtener las 10 IPs con más accesos. Aquí es donde tuvimos que pensar por qué usar un Binary Heap y no un BST (Binary Search Tree). Como bien comentaba con mi equipo, la respuesta está en el objetivo final. Un BST es excelente si nuestra prioridad es buscar, insertar y borrar cualquier elemento manteniendo un orden total. Es tipo una biblioteca donde todo está en su lugar. Pero en este caso, queríamos los valores más altos. El Binary Heap es eficiente para esto. Mientras que en un BST balanceado buscar el máximo te toma O(log n), en un Maxheap el acceso al elemento más grande es una operación constante O(1). Así siempre sabemos quién está a la cabeza. Además, para obtener el Top 10, simplemente extraemos la raíz 10 veces. Cada extracción (pop) nos cuesta O(log n) porque el Heap tiene que reacomodarse, pero sigue siendo mucho más directo y eficiente que recorrer un árbol de búsqueda que podría estar desbalanceado si no tenemos cuidado. En nuestro código, el uso de la priority_queue de C++ facilitó mucho este proceso.

Además, es fascinante ver cómo impacta el desempeño. La operación de push para insertar una nueva IP con su contador de accesos es O(log n), y como ya mencioné, el pop para sacar la IP con más accesos también es O(log n). Si comparamos esto con una lista simple donde tendríamos que buscar el máximo cada vez (O(n)), la diferencia en una bitácora de miles de registros es la diferencia entre un programa que termina al instante y uno que parece que se trabó, como nos pasaba al principio del curso.

Más allá de la programación, esta actividad tiene una aplicación directa en la seguridad informática. Al analizar los resultados y ver qué IPs tienen miles de accesos, uno empieza a entender cómo se detecta una red infectada o un ataque en curso. Si yo veo que una IP externa tiene un pico de actividad anormalmente alto en cuestión de segundos, o si los mensajes de la bitácora muestran "Failed password" repetidamente desde un mismo origen, es casi seguro que estamos ante un ataque de fuerza bruta o que un bot está escaneando vulnerabilidades. Una red bien hecha tiene patrones de tráfico predecibles. Es decir, una red infectada presenta anomalías que estas estructuras de datos nos permiten identificar de inmediato. El Binary Heap nos permite poner esas flags hasta arriba de la lista sin perder tiempo procesando datos irrelevantes.

Trabajar de nuevo con mi equipo me confirmó que discutir las soluciones es tan importante como escribirlas. Hubo momentos donde dudamos sobre cómo separar el puerto de la IP sin afectar el rendimiento, pero al final la lógica de la bitácora ordenada nos permitió contar los accesos de forma lineal y luego mandarlos al Heap. Me voy de esta actividad con una idea mucho más clara de que no existe una mejor estructura de datos universal, sino que cada problema tiene una estructura que encaja como pieza de rompecabezas. Si lo que importa es la prioridad, el Heap es la mejor opción

Referencias:

- GeeksforGeeks. (2023). *Heap Sort - Data Structure and Algorithms Tutorials*. Recuperado de https://www.geeksforgeeks.org/heap-sort/
- Cplusplus.com. (2024). *std::priority_queue*. Recuperado de https://cplusplus.com/reference/queue/priority_queue/
- Cloudflare. (2024). ¿Qué es el análisis de logs y cómo ayuda en la seguridad?. Recuperado de https://www.cloudflare.com/es-es/learning/network-layer/what-is-log-analysis/