Tecnológico de Monterrey

Reflexión personal

Javier Barron Vargas
A00842507

En esta actividad trabajamos con una bitácora y el principal reto fue no solo guardar los datos, sino poder analizarlos de forma rápida aunque la cantidad de información fuera muy grande. Algo que entendimos desde el inicio es que elegir bien las estructuras de datos sí hace la diferencia en el rendimiento del programa, sobre todo cuando el número de registros empieza a crecer mucho. Para representar la información usamos un grafo dirigido con lista de adyacencias. Esta decisión se tomó porque la bitácora básicamente representa relaciones entre direcciones IP, es decir, qué IP intenta conectarse con cuál. La lista de adyacencias fue una buena opción porque solo guarda las conexiones que realmente existen. Si hubiéramos usado una matriz de adyacencias, el uso de memoria crecería demasiado rápido, ya que sería $O(V^2)$. En cambio, con lista de adyacencias el costo es $O(V + E)$, lo cual es más eficiente cuando hay muchas IP pero no todas se conectan entre sí.
También vimos que recorrer las conexiones de una IP es eficiente porque solo se recorren sus conexiones directas. El problema es cuando se necesita buscar información de una IP muchas veces, porque ahí el grafo puede volverse más lento y acercarse a $O(V)$ en el peor caso. Por eso se decidió usar una tabla hash para guardar un resumen de cada IP. La ventaja principal es que permite hacer búsquedas e inserciones en promedio en $O(1)$, lo cual es muy importante cuando se manejan grandes cantidades de datos. La función hash convierte la IP en un índice dentro de la tabla. Técnicamente calcular el hash depende del tamaño del string de la IP, o sea $O(L)$, pero como las IP tienen tamaños parecidos, en la práctica se comporta casi constante. Para manejar las colisiones usamos direccionamiento abierto con prueba cuadrática. Se eligió este método porque ayuda a distribuir mejor los datos dentro de la tabla y evita que se acumulen en zonas como puede pasar con la prueba lineal. Esto ayuda a mantener el rendimiento más estable cuando la tabla empieza a llenarse. Algo que también fue importante ver es cómo el número de colisiones afecta el rendimiento real. Cuando hay pocas colisiones, las operaciones siguen siendo muy rápidas. Pero cuando aumentan, el tiempo empieza a crecer porque se necesitan más intentos para encontrar una posición libre o la información buscada. Esto depende mucho del tamaño de la tabla y del factor de carga. El método getIPSummary permite obtener toda la información de una IP de forma rápida. La búsqueda en la tabla hash se mantiene en $O(1)$ en promedio, mientras que ordenar las direcciones accesadas depende del algoritmo usado, normalmente $O(n \log n)$ si se usa un ordenamiento eficiente. En general, esta actividad nos ayudó a entender cómo las decisiones de diseño afectan directamente el desempeño del sistema. El grafo permitió representar las relaciones entre IP sin gastar memoria de más, mientras que la tabla hash permitió hacer consultas rápidas sin recorrer todo el grafo. También nos ayudó a entender que las colisiones no son solo algo teórico, sino algo que realmente puede afectar el rendimiento si no se controla el tamaño de la tabla. En lo personal, esta actividad me ayudó a ver de forma más clara cómo la complejidad computacional sí importa en problemas reales, especialmente en situaciones como el análisis de tráfico de red, donde la cantidad de datos puede crecer mucho y el sistema tiene que seguir funcionando de forma eficiente.