Valeria Salazar Pinto
A01736652

Reflexión – Act 5.2

En esta actividad trabajamos con una bitácora de tráfico de red donde aparecen muchísimas direcciones IP conectándose entre sí. En un problema así, el reto real no es “guardar datos”, sino poder consultarlos rápido y de forma consistente, porque en ciberseguridad lo que importa es detectar patrones (quién se conecta con quién, cuántas veces, qué IPs resaltan, etc.) sin que el programa se vuelva lento conforme crece el archivo. Por eso, además del grafo para modelar relaciones entre IPs, las tablas hash son clave: nos permiten ubicar y acceder a la información asociada a una IP casi de inmediato, sin tener que recorrer listas enormes cada vez.

Lo valioso de una tabla hash es que transforma una “búsqueda por recorrido” en una “búsqueda por dirección”: aplicamos una función hash a la IP (o a la llave que usemos), y eso nos da un índice donde debería quedar su información. En promedio, operaciones como insertar y buscar son O(1), o sea, tiempo constante: no depende directamente de cuántas IPs haya, sino de qué tan bien se comporta la tabla. Esto, en una bitácora realista con miles o millones de registros, hace una diferencia enorme, porque el programa puede estar consultando repetidamente contadores, conexiones, o resúmenes por IP, y hacerlo con listas lineales sería demasiado costoso.

Pero esa eficiencia no es “mágica”: depende de las colisiones. Una colisión ocurre cuando dos llaves diferentes caen en el mismo índice. Cuando hay pocas colisiones, todo fluye como se espera y seguimos cerca de O(1). El problema es cuando la tabla se empieza a llenar (sube el factor de carga) o la función hash no distribuye bien: ahí aumentan las colisiones y cada acceso deja de ser “llegar y listo”, porque hay que resolver el choque (por ejemplo, recorriendo una lista en ese índice o siguiendo reubicaciones). En ese escenario, el tiempo por operación ya no se mantiene constante y puede degradarse hasta acercarse a O(n) en el peor caso, que sería básicamente volver a una búsqueda lenta, justo lo que queríamos evitar. Por eso tiene sentido analizar cómo cambian las colisiones al modificar el tamaño de la tabla: no

es un detalle menor, es literalmente lo que define si el enfoque escala bien o se vuelve pesado.

En conclusión, para un análisis de red como el de esta actividad, las tablas hash son una herramienta muy eficiente porque aceleran las consultas repetidas por IP y ayudan a que el sistema se mantenga rápido aunque el volumen de datos crezca. Eso sí: para que realmente funcionen "como prometen", hay que cuidar el dimensionamiento, el factor de carga, el método de manejo de colisiones y, sobre todo, una función hash que distribuya bien. Con esos cuidados, el beneficio es directo: mejor rendimiento, respuestas más rápidas, y más capacidad de detectar actividad relevante o sospechosa sin que el análisis se vuelva impráctico cuando la bitácora aumenta.

Appa:

GeeksforGeeks. (2026b, enero 26). ***Hashing in Data Structure*****. GeeksforGeeks.** https://www.geeksforgeeks.org/dsa/hashing-data-structure/

GeeksforGeeks. (2025, 23 julio). ***Hash table data structure*****. GeeksforGeeks.** https://www.geeksforgeeks.org/dsa/hash-table-data-structure/

Gupta, A. (2013, 16 noviembre). ***How to efficiently hash the ip-address*****. Stack Overflow.** https://stackoverflow.com/questions/20016165/how-to-efficiently-hash-the-ip-address