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 5.2 - Actividad Integral sobre el uso de códigos hash (Evidencia Competencia)

04/02/2026

Monterrey, Nuevo León

Reflexión Individual - Actividad 5.2

Veníamos de trabajar con grafos en la actividad pasada y aunque ya podíamos mapear toda la red de conexiones, nos dimos cuenta que buscar los datos de una sola IP era lento si teníamos que recorrer toda la estructura cada vez. Aquí es donde entró la tabla hash. Entre nosotros platicamos que no tenía sentido tener una base de datos gigante si cada query iba a tardar una eternidad. Es decir, necesitábamos acceso instantáneo que tuviera O(1).

Para implementar esto, nos fuimos por el método de direccionamiento abierto con prueba cuadrática. Al leer la documentación y ver el código, decidimos usar esta estrategia en lugar de la prueba lineal porque queríamos evitar el clustering primario. Básicamente si una posición ya estaba ocupada, en lugar de intentar poner el dato en la casilla de al lado, la prueba cuadrática nos permitía saltar más lejos y distribuir mejor los datos en la tabla. También tuvimos que programar la función de hashing en IPGraph.cpp. Para eso tuvimos que convertir la IP a un número (long long) para poder generar el índice. Al correr el programa fue claro el impacto del tamaño de la tabla. Hicimos pruebas cambiando el tamaño y viendo el contador de colisiones. Cuando pusimos un tamaño de tabla muy ajustado al número de datos, las colisiones se dispararon y el rendimiento se empezó a sentir más pesado. Si el factor de carga alpha sube mucho, la complejidad deja de ser constante y se cambia hacia O(n) en el peor caso, porque el programa pierde tiempo resolviendo choques en lugar de insertar o buscar directo.

El método getIPSummary() fue el método que se encargó con la mayoría de nuestro código. Gracias a la tabla hash, pudimos meter el IP, calcular su llave y sacar sus degrees (entrada y salida) de golpe. Si hubiéramos seguido usando solo listas o vectores sin indexar, buscar ese resumen hubiera requerido iterar miles de registros. Ver que la respuesta aparece en consola al instante te hace valorar la eficiencia real de estas estructuras. Me voy de esta actividad con la seguridad de que las tablas hash son una herramienta de eficiencia que se utiliza en casi todo lo que usamos. Desde los índices en una base de datos SQL hasta cómo un compilador guarda nombres de variables. No basta con que funcione, sino que tiene que ser escalable. Si no manejas bien las colisiones o eliges mal tu función hash, terminas con una lista ligada glorificada y lenta.

Referencias:

- GeeksforGeeks. (2024, 26 de enero). Hashing in Data Structure. https://www.geeksforgeeks.org/hashing-data-structure/
- Programiz. (2024). Hash Table. https://www.programiz.com/dsa/hash-table

- OpenDSA. (2023). Quadratic Probing. CS3 Data Structures and Algorithms. https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/QuadraticProbing.html

- Brilliant.org. (2024). Hash Tables. https://brilliant.org/wiki/hash-tables/