Tecnológico de Monterrey

Act 5.3 – Actividad Integral Sobre el uso de Códigos Hash

Programación de Estructuras de Datos y Algoritmos Fundamentales

Instituto Tecnológico y de Estudios Superiores de Monterrey

Fecha de entrega: 27 de julio de 2023

Profesor: Dr. Eduardo Arturo Rodríguez Tello

Estudiante: Alyson Melissa Sánchez Serratos - A01771843

REFLEXIÓN ACTIVIDAD 5.2 FINAL

En el ámbito de la ciberseguridad y el almacenamiento de información, las tablas hash se han destacado como una herramienta fundamental y altamente eficiente. Estas estructuras de datos proporcionan una manera rápida y segura de vincular claves (llaves) con sus respectivos valores asociados, lo cual es especialmente valioso en el análisis y manejo de direcciones IP en entornos digitales. En el presente escrito, se explorará la importancia crucial de las tablas hash al enfrentar problemas que requieren un almacenamiento, análisis y procesamiento eficiente de información relacionada con direcciones IP y sus actividades.

De acuerdo con Knuth, D. E. (2022), una tabla hash es definida como:

> Una estructura de datos que permite el almacenamiento, búsqueda y recuperación eficiente de información, donde cada elemento es mapeado a una posición única en la tabla mediante una función de hash que convierte la clave en un valor numérico, permitiendo el acceso directo a la información asociada. (p.314)

Como se comenta en la cita, el funcionamiento de esta estructura se basa en el uso de una función de hash que toma una clave y la convierte en un valor numérico fijo, conocido como el hash. Este hash se utiliza para indexar y almacenar el valor asociado a la clave en una posición única en la tabla. La eficacia de las tablas hash radica en su capacidad para acceder directamente a la información asociada a una clave, evitando búsquedas exhaustivas y logrando una velocidad de acceso impresionante. En el ámbito de la ciberseguridad, la rapidez y la precisión son esenciales para la detección y prevención de amenazas. Las tablas hash permiten realizar búsquedas y comprobaciones de direcciones IP de manera sumamente eficiente. Al calcular rápidamente el hash y acceder directamente a la posición en la tabla, se pueden verificar en tiempo real si una dirección IP está asociada a comportamientos maliciosos o sospechosos.

En la Actividad Integradora 5.2 se decidió implementar la tabla hash utilizado el método de dirección abierta con prueba cuadrática.

De acuerdo con Yadav, P (2022):

> El concepto principal del hashing de dirección abierta es mantener todos los datos en la misma tabla hash, lo que requiere una tabla hash más grande. Cuando se utiliza dirección abierta, una colisión se resuelve mediante la búsqueda de celdas alternativas en la tabla hash hasta encontrar la celda objetivo. Se recomienda mantener el factor de carga (α) por debajo de 0.5, donde α se define como $\alpha=n/m$, donde "n" es el número total de entradas en la tabla hash y "m" es el tamaño de la tabla hash.

Aplicando el enfoque cuadrático, cuando se produce una colisión (es decir, dos claves diferentes se asignan a la misma posición en la tabla hash), se realiza una secuencia de intentos de búsqueda en ubicaciones alternativas utilizando una función de prueba cuadrática hasta encontrar una posición vacía para almacenar el nuevo valor.

Dicha prueba se basa en una función que determina la siguiente posición en la tabla a ser probada mediante una serie de incrementos cuadráticos. La función de prueba cuadrática se define de la siguiente manera:

$$hash(key, i) = (hash(key) + i * i) \% table\_size$$ **Ecuación 1: prueba cuadrática**

De donde:

- hash(key): es el valor de hash original de la clave.
- i: es el número de intento de búsqueda, comenzando con i = 0.
- table_size: es el tamaño total de la tabla hash.

En cada intento, se calcula el siguiente índice en la tabla utilizando la función de prueba cuadrática, es decir, el resultado del valor de hash original más el cuadrado del número de intento, todo esto módulo el tamaño de la tabla. Se repite este proceso hasta encontrar una posición vacía en la tabla o hasta que se haya intentado un número máximo de intentos sin éxito.

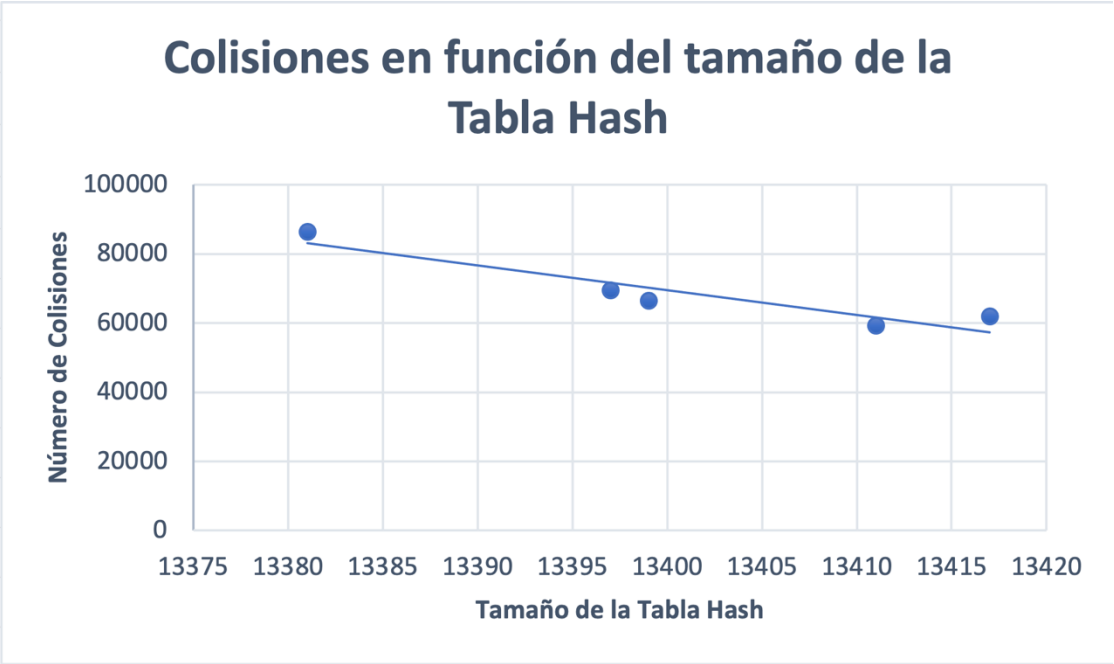
En cuanto a ventajas, el método de dirección abierta con prueba cuadrática para implementar tablas hash ofrece varios beneficios significativos. Entre ellos se destacan su eficiencia en búsquedas, ya que permite explorar posiciones alternativas en la tabla de manera rápida y eficiente, reduciendo el número de colisiones y mejorando el acceso a los elementos. Además, el método no requiere espacio adicional, lo que lo hace más eficiente en términos de memoria y simplifica su implementación. La prueba cuadrática también tiende a reducir la fragmentación en la tabla, distribuyendo de manera más uniforme los elementos. Además, puede adaptarse bien a diferentes factores de carga.

No obstante, también hay limitantes, si el factor de carga es alto o el tamaño de la tabla es demasiado pequeño en comparación con la cantidad de elementos, las colisiones pueden aumentar, lo que podría disminuir la eficiencia del método.

La claridad de este fenómeno se destaca en la actividad realizada al analizar el número de colisiones al usar un tamaño de tabla hash específico. Para realizar este análisis, se implementó un método que calcula los cinco números primos más cercanos, pero mayores, al número de direcciones IP encontradas en la bitácora. Posteriormente, el usuario tiene la opción de elegir el tamaño de tabla hash deseado y observar el número de colisiones resultante. A continuación, en la **tabla** y **gráfica 1** se observa la variabilidad de las colisiones en función del tamaño de la tabla hash.

Tabla 1 Número de colisiones de acuerdo con el tamaño de la tabla hash

Tamaño de la Tabla Hash	Colisiones
13381	86394
13397	69535
13399	66518
13411	59247
13417	62040



Gráfica 1 Número de colisiones de acuerdo con el tamaño de la tabla hash

Como se puede observar, en la mayoría de los casos se observa una correlación entre el tamaño de la tabla hash y el número de colisiones. Esto puede ocurrir al aumentar el tamaño de la tabla debido a que un tamaño mayor proporciona más posiciones disponibles para almacenar las claves. Esto permite una mejor distribución de las claves en la tabla y reduce la probabilidad de que dos claves diferentes se mapeen a la misma posición.

No obstante, se debe destacar que para el valor 13417 dicho patrón no se cumple. Esta anomalía se considera extraña pues dicho número es primo, lo cual, en teoría, ayuda a minimizar los patrones de agrupación de las claves en ciertas posiciones, lo que disminuye la probabilidad de que múltiples claves colisionen en la misma ubicación. Por lo tanto, se puede teorizar que esto ocurre debido a la función hash la cual podría ser ineficiente para ese número en específico.

Para comprobar, se realizaron pruebas adicionales con los números primos 13421 y 13441, los cuales generaron 58824 y 52157 colisiones, respectivamente, cumpliendo así con la correlación hallada inicialmente.

Al analizar los resultados, se pudo concluir que aumentar el tamaño de la tabla hash puede ser beneficioso para reducir el número de colisiones y mejorar el rendimiento en términos de acceso a elementos. Sin embargo, esto también tiene implicaciones en el uso de memoria y puede requerir más recursos computacionales para el redimensionamiento. Por lo tanto, la elección del tamaño de la tabla debe equilibrar estos factores y adaptarse a los requisitos específicos del problema.

Finalmente, a nivel general, los métodos utilizados para la creación de la tabla hash poseen una eficacia temporal óptima la cual va desde O (1) hasta O(n). Los peores casos de los métodos implementados están estrechamente relacionados con la existencia de colisiones.

Para ello, las clases utilizadas para la creación de una tabla hash implementa un vector llamado *overflow* el cual indica en qué posiciones de la tabla quedaron los elementos colisionados en determinada celda. Esta es una estrategia eficiente y práctica para manejar colisiones en una tabla hash ya que proporciona un acceso rápido a los elementos colisionados de cada celda, mantiene la integridad de la tabla y evita rehashing, y búsquedas innecesarias, lo que mejora la eficiencia y el rendimiento general de la tabla hash.

Asimismo, el mantener un estado para cada celda de la tabla hash, indicando si está "ocupada", "libre" o "eliminada", mejoró significativamente la eficiencia del programa en diversas áreas. En primer lugar, esta estrategia permite una resolución más eficiente de colisiones. Cuando dos o más elementos tienen el mismo índice hash y deben almacenarse en la misma posición de la tabla, se produce una colisión. Al conocer el estado de cada celda, el programa puede buscar rápidamente celdas libres para insertar nuevos elementos sin necesidad de realizar búsquedas adicionales o rehashing, lo que agiliza el proceso y garantiza que todos los elementos se conserven correctamente. A su vez, el estado de cada celda permite una eliminación más eficiente de elementos en la tabla hash. Cuando un elemento se elimina, en lugar de eliminarlo físicamente, se puede marcar la celda como "eliminada". Esto evita la necesidad de reorganizar toda la tabla, lo que puede ser costoso en términos de tiempo y recursos, y permite una eficiente recuperación de espacios liberados para futuras inserciones.

Para finalizar, en la **tabla 2** y **3** se muestra el resumen de las complejidades temporales para una tabla hash implementada con el método de dirección abierta, así como sus motivos de complejidad:

Tabla 2 complejidades temporales para el método de dirección abierta. Knuth, D. E. (2022).

ACTIVITY	BEST CASE COMPLEXITY	AVERAGE CASE COMPLEXITY	WORST CASE COMPLEXITY
Searching	O (1)	O (1)	O (n)
Insertion	O (1)	O (1)	O (n)
Deletion	O (1)	O (1)	O (n)
Space Complexity	O (n)	O (n)	O (n)

Tabla 3 Razones para la complejidad de operaciones de Tabla Hash con método de dirección abierta

ACTIVIDAD	MOTIVO
Insertion	En el caso promedio, la inserción es constante O (1) porque se asume que las colisiones son mínimas y las posiciones disponibles en la tabla son fáciles de encontrar. Sin embargo, en el peor caso, la inserción puede requerir recorrer la tabla completa para encontrar una posición libre cuando se producen muchas colisiones (por ejemplo, si se insertan muchos elementos con el mismo hash).
Searching	En el caso promedio, la búsqueda es constante O (1) porque se asume que las colisiones son mínimas y los elementos se pueden encontrar rápidamente.

	En el peor caso, cuando se producen muchas colisiones, la búsqueda puede requerir recorrer la tabla completa para encontrar el elemento deseado, lo que lleva a una complejidad lineal O (n).
Deletion	Al igual que con la inserción y búsqueda, la eliminación es constante en el caso promedio O (1) cuando no hay muchas colisiones. Sin embargo, en el peor caso, si se producen colisiones y el elemento a eliminar no está en la primera posición calculada por la función hash, la tabla podría tener que recorrerse para encontrarlo, llevando a una complejidad lineal O (n).

Fuentes de información:

Knuth, D. E. (2022). Art of computer programming, the: Combinatorial algorithms, volume 4B. Addison-Wesley Educational.

Varun Iyer, J. (2021, diciembre 6). Time and Space Complexity of Hash Table operations. OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/time-complexity-of-hash-table/

Yadav, P. (2022, abril 1). Open addressing. Scaler Topics. https://www.scaler.com/topics/data-structures/open-addressing/