Tecnológico de Monterrey

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

Nikole Morales Rosas - A01782557

TECNOLÓGICO DE MONTERREY

Campus Santa Fe

Grupo: 570

Fecha de entrega: 28 de julio de 2023

Profesor: Dr. Eduardo A. Rodríguez Tello

Programación de estructuras de datos y algoritmos fundamentales

Hashing

Reflexión

Las tablas hash son una estructura de datos asociativa que relaciona una llave y un valor utilizando una función de hash, esta función de hash se utiliza para calcular el índice al que han de ir los elementos que se están guardando en la tabla, es decir, su implementación se basa en una función de hash que transforma las claves de búsqueda en índices de la tabla, lo que permite un acceso rápido a los datos asociados con esas claves.

Una tabla de hash guarda elementos en lo que se conoce como buckets, esta tabla puede tener un número arbitrario de buckets y es tarea de la función de hash determinar a qué bucket tiene que ir cada elemento.

Según se explica en un artículo de Geek For Geeks: “El valor hash se usa para crear un índice para la clave en la tabla hash. La función hash puede devolver el mismo valor hash para dos o más claves. Cuando dos o más claves tienen el mismo valor hash, ocurre una colisión. Para manejar esta colisión, utilizamos técnicas de resolución de colisiones.”

(GeekForGeeks, s/f)

Como se menciona en el blog de Martín Cruz, algunos de los métodos de manejo de colisiones son el hashing abierto y cerrado, cada uno aborda las colisiones de distinta forma. En el hashing abierto, cuando se produce una colisión, se intenta buscar otra ubicación dentro de la misma tabla para almacenar el elemento colisionado. En lugar de usar estructuras de datos externas (como listas enlazadas) para manejar las colisiones, se explora directamente la propia tabla hash para encontrar un nuevo lugar para el elemento.

Por otro lado, en el hashing cerrado, cuando ocurre una colisión, los elementos que tiene la misma clave hash se almacenan en estructuras de datos externas asociadas con cada celda de la tabla hash. Por lo general, se utilizan listas enlazadas, aunque otras estructuras como árboles también pueden ser utilizadas.

Cuando se busca un elemento en la tabla hash, la función de hash calcula el índice, y luego se busca en la estructura de datos asociada (lista enlazada, árbol, etc.) para encontrar el elemento deseado.

Para esta actividad se tuvo una lista de adyacencias que representa un grafo dirigido con información sobre direcciones IP y sus conexiones. La tabla hash es utilizada para almacenar un resumen de la información de todas las IP, incluyendo el total de direcciones accesadas desde cada IP y el total de direcciones que intentaron acceder a cada IP. Además, se implementa el método de dirección abierta con prueba cuadrática para resolver colisiones.

En este caso para acceder a cualquier información de culturizar IP se tiene una complejidad computacional de O(1), con esto el programa se vuelve más eficiente.

En el programa implementado se utilizó un método llamado ***getIPSummary()*** cuyo objetivo es proporcionar un resumen detallado de la información relacionada con una dirección IP específica que se almacena en una tabla hash. El método toma como entrada la dirección IP que se desea consultar y busca su resumen correspondiente en la tabla hash.

Esto es útil para analizar patrones de comportamiento de una dirección IP en particular y obtener una visión general de sus interacciones en el grafo dirigido representado por la tabla hash. En cuanto a la complejidad temporal es de O(1) debido a que no se generan estructuras de datos adicionales.

Una de las ventajas de usar tablas hash en una situación de este tipo es que se ayuda a evitar que la información se duplique ya que solo se almacena un resumen por IP. Además, mejora la administración y mantenimiento de los datos, ya que se pueden realizar operaciones de inserción, búsqueda y eliminación de manera eficiente.

A continuación se muestran las complejidades en operaciones de la tabla hash:

Add	En el mejor caso, cuando no hay colisiones, la complejidad de inserción es O(1), ya que simplemente se calcula el índice y se inserta el valor en la tabla.
	En el peor caso, cuando hay muchas colisiones, la complejidad puede aproximarse a O(n), donde n es el tamaño de la tabla.
Search	La búsqueda tiene una complejidad promedio de O(1) si no hay colisiones importantes.
	En el peor caso, si hay muchas colisiones, la complejidad puede ser cercana a O(n) en búsqueda.
Delete	Al igual que las demás operaciones en el mejor caso se tiene una complejidad promedio de O(1) si no hay colisiones importantes.
	En el peor caso, cuando hay muchas colisiones y se debe buscar el elemento a eliminar, la complejidad puede aproximarse a O(n), donde n es el tamaño de la tabla hash.

Tabla 1.1: Operaciones de las tablas hash y sus complejidades

Al aumentar el número de colisiones, la complejidad computacional de las operaciones en la tabla hash ya que sufren un cambio, En el caso del ***Add,*** al haber mayor número de colisiones, la velocidad para hacer la inserción de algún elemento puede disminuir, esto significa que el programa se vuelve más lento. Cuando ocurre una colisión, el algoritmo de resolución de colisiones debe buscar un lugar vacío en la tabla, lo que implica revisar varias posiciones hasta encontrar espacio disponible. Cuantas más colisiones haya, mayor será la probabilidad de que esto suceda, aumentando así la complejidad de inserción. En promedio, la complejidad seguirá siendo cercana a O(1), pero podría haber una ligera reducción en el rendimiento pero como se mencionan en la *Tabla 1.1* en el peor caso esta complejidad se puede volver O(n).

Así mismo el ***Search***, si el número de colisiones es alto, la búsqueda también puede volverse más lenta. En el peor caso, cuando hay muchas colisiones, el algoritmo de búsqueda debe revisar secuencialmente múltiples posiciones en la tabla hasta encontrar el elemento deseado. Esto aumenta la complejidad en el peor caso, que puede aproximarse a O(n).

Sucede algo similar en el caso de eliminación ***Delete*** ya que tiene la misma complejidad que el search y el find que en el mejor caso se tiene una complejidad de O(1), mientras que en el peor caso se tiene una complejidad de O(n).

En general, la complejidad computacional puede aumentar significativamente en el peor caso si no se consideran factores como la función de hash, la técnica de resolución de colisiones y el tamaño de la tabla. Sin embargo, con una elección adecuada de estos elementos, las tablas hash pueden mantener un rendimiento óptimo en una amplia variedad de situaciones problemáticas.

Referencias

Sin Autor. *Hashing data structure*. (2015, diciembre 14). GeeksforGeeks. Recuperado el 27 de julio de 2023 en:
https://www.geeksforgeeks.org/hashing-data-structure/

Cruz, M. (2013, marzo 4). *Método de Dispersión (Hashing) en C++*. Martincruz.me. Recuperado el 27 de julio de 2023 en:
https://blog.martincruz.me/2013/03/metodo-de-dispersion-hashing-en-c.html

Sin Autor. (2023, abril 24). *Hash table data structure.* GeeksforGeeks. Recuperado el 27 de julio de 2023 en:
https://www.geeksforgeeks.org/hash-table-data-structure/

Sin Autor. (2022, julio 4). *Introduction to hashing - data structure and algorithm tutorials.* GeeksforGeeks. Recuperado el 27 de julio de 2023 en:
https://www.geeksforgeeks.org/introduction-to-hashing-data-structure-and-algorithm-tutorials/