Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey

Programación de estructura de datos y algoritmos fundamentales

TC1031, Grupo 601

Nombre del profesor: Dr. Eduardo Arturo Rodríguez Tello

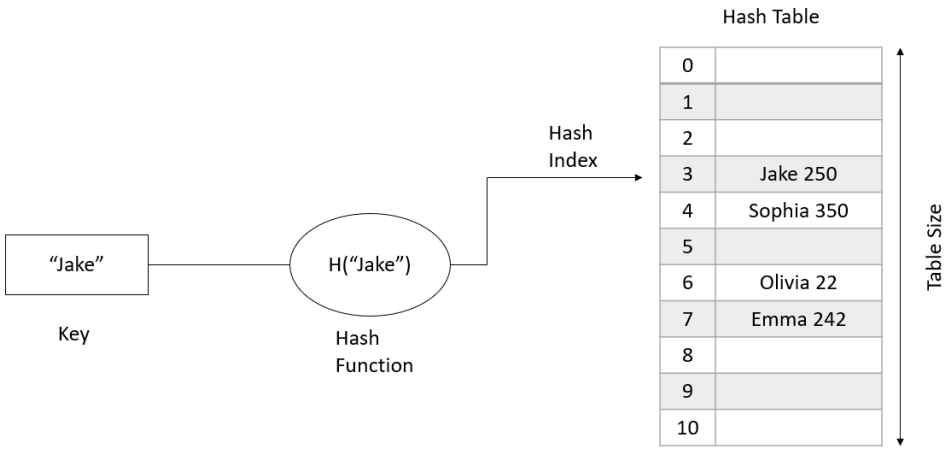
Actividad 5.2 – Reflexión personal

Samuel Acosta Ugarte | A00833547

19 de junio 2022

Hashing

A grandes rasgos, hashing es el proceso de convertir cualquier entrada de cualquier longitud a un numero o string usando un algoritmo. La idea general es poder usar una **función hash**, que puede ser determinada por el programador, que convierte una llave recibida a un número mas chico que se usara como índice de una tabla llamada tabla hash (Bansal, 2021).



Esta tabla contiene pares de llaves y valores, que pueden contener objetos diseñados por el programador.

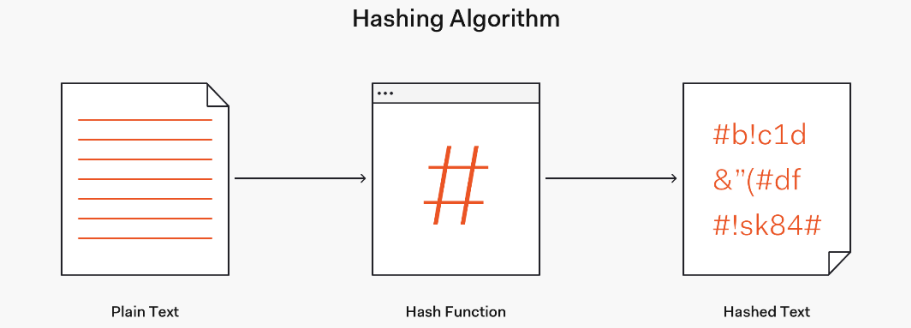
Métodos importantes de hashing:

- Insertar (add)
- Borrar (delete)
- Buscar (find)

Como mencionado antes, a la celda donde se coloca la información es en base a una función hash, que puede dar un mismo resultado para diferentes datos. A esto se le conoce como una colisión, el número de colisiones depende mucho de la función hash y el tamaño de la tabla hash. Uno no puede completamente eliminar las colisiones, sin embargo, si se pueden minimizar, teniendo una buena función hash y un tamaño de la tabla adecuada, generalmente del tamaño de un número primo arriba del número de elementos que necesitas insertar.

Aplicaciones

Hashing se usa en cualquier situación donde necesitas que la búsqueda o inserción de algún elemento se necesite realizar de manera muy rápida (Bansal, 2021). Como en promedio su búsqueda, inserción y borrado son en promedio en tiempo constante, se usan para problemas donde necesites contar la frecuencia, encontrar duplicados, etc. (GeeksforGeeks, 2022).



Las tablas hash se usan en verificación de contraseñas, en conectar archivos con directorios, incluso en C++, algunas estructuras de datos como el unordered_set y unordered_map se basan en tablas hash.

Realización de la actividad 5.2 y complejidades computacionales

En esta actividad se retomo la lectura y almacenamiento de la información a través de grafos dirigidos ponderados mediante listas de adyacencia. Durante la lectura además de encontrar el grado de salida de cada ip, se contabilizo el numero de llegadas a cada ip, o el grado de entrada. El código se queda con las implementaciones de vectores, mapas y pares, con la misma complejidad que la vez pasada. En este caso usando estas estructuras ya establecidas, creamos una tabla hash que nos permite encontrar la información en tiempo constante.

Recordemos nuestra complejidad para cargar el grafo dirigido ponderado:

- **loadGraphList (string)** **Complejidad: O ((V Log V) + (E Log V))**

    - Este es el que lee la bitácora completa y almacena todo en un grafo representado por lista de adyacencia. Por cada línea que describe nodos (hay V líneas, nodos), está insertándolo en un mapa que tiene como complejidad log n, en este caso log V por insertar hasta número de nodos. Juntos se multiplican generando **V Log V** para la primera parte.

    - Cuando llegamos a las líneas donde se encuentran las incidencias, o el número de aristas, por cada arista E se busca un valor donde recorre el número de nodos V, creando una complejidad para esta parte **E Log V**.

    - Juntos crean la complejidad final **O ((V Log V) + (E Log V))**

Además de este método, se implementaron dos métodos para la implementación de la tabla hash con cuadratic probing, que genera menos agrupamiento que linear probing.

- **buildHashTable ()** **Complejidad: O (V)**

    - Este método nos crea la tabla hash. Primeramente, se determina el tamaño de la tabla hash, que, a través de varias pruebas con diferentes tamaños, concluimos que una tabla del tamaño del siguiente número primo del doble del número de IPs, resulto en menos colisiones.
    - Probamos Con estos diferentes tamaños:

        - 17383, el siguiente número primo del 130% del numero de Ips. Generaba alrededor de 5210 colisiones.
        - 21521, el siguiente número primo del 160% del numero de Ips. Generaba alrededor de 4167
        - El valor de 32537 es el siguiente numero primo del 200% del numero de Ips. Este generaba solo 2657, decidimos optar por este tamaño.
    - Este método usa un bucle for donde se va agregando a tiempo constante gracias a un valor de Alpha, menor que uno, por cada número de vértices, o Ips, V, creando una complejidad de O (V).

- **IpSummary (string)** **Complejidad: O (log V + E)**

    - Este método es en realidad solo para llamar al método getIpSummary. Este método recibe una ip en formato string del usuario. Por medio de este string, se hace una conversión a objeto tipo Ip, para buscarlo en nuestra tabla hash por medio de si Ipvalue. Si es que no se encuentra la IP en la tabla hash, se le hace conocer al usuario, de otra manera, una vez encontrada la IP, se crea un vector lleno de las Ips, con la que esta esta relacionada. De esta manera se ordena y se imprime en pantalla en orden ascendente.

    - Este método inicialmente encuentra el índice de la IP para ser buscada en el vector de objetos IP, que en la actividad anterior determinamos que era una complejidad de O (log V) por el uso de mapa. En la segunda parte, se llena un vector por medio de un for que se repite dependiendo del tamaño de la lista ligada, o el numero de aristas E que tenga esa ip. Un for similar se usa para imprimirla mas adelante. Ambos tienen una complejidad de O (E). Esto crea una suma como una complejidad de **O (log V + E**).

Usar esta estructura de datos es sumamente útil para cuando se necesiten insertar, borrar o buscar elementos rápido. El poder realizar búsquedas con tiempo constante en un campo de datos tan amplio tiene muchas ventajas. Sin embargo, uno tiene que tener cuidado de como las maneja, porque una tabla hash mal implementada puede ser ineficiente. Al tener muchas colisiones puede convertirse en una complejidad de O (n) para búsqueda, inserción y borrado.

Referencias bibliográficas

GeeksforGeeks. (2022). Hashing | Set 3 (Open Addressing). 2022, de GeeksforGeeks Sitio web: https://www.geeksforgeeks.org/hashing-set-3-open-addressing/

Bansal, A. (2021). Hashing in Data Structures. 2022, de Section Sitio web: https://www.section.io/engineering-education/hashing-in-data-structures/#:~:text=Hashing%20is%20the%20process%20of,table%20called%20a%20hash%20table.

GeeksforGeeks. (2022). Applications of Hashing. 2022, de GeeksforGeeks Sitio web: https://www.geeksforgeeks.org/applications-of-hashing/

Varun, J. (2021). Time and Space Complexity of Hash Table operations. 2022, de opengenius Sitio web: https://iq.opengenus.org/time-complexity-of-hash-table/