Emilia Victoria Jácome Iñiguez.

A00828347 – ITD

Estructura de Datos y Algoritmos Fudamentales

1/12/2020

Reflexión Actividad 5.2: Hash Tables

Las Hash tables son consideradas como una de las principales estructuras de datos ordenados que utiliza una técnica efectiva para asignar una identidad propia a cada objeto perteneciente a una lista de datos similares y del mismo tipo. Dicha identidad propia es llamada como la llave de dicho dato la cual debe ser de carácter numérico entero (int). Tanto la búsqueda como el llenado de cualquier tabla hash se realiza en función a la llave de cada elemento. Adicionalmente, también se emplean las llaves para acceder a un elemento en particular ya que se puede utilizar la llave para hacer un cálculo del índice en donde se encuentra dicho dato. Las aplicaciones de la Hash table son casi infinitas. Comúnmente se utilizan al implementar distintos tipos de tablas de memoria para almacenar posiciones. En adición, tienen un rol importante en la indexación de bases de datos para encontrar una determinada tupla dentro de una base de datos con efectividad. Por otra parte, las tablas hash facilitan la implementación de catchés que permiten un acceso de datos casi instantáneo. En efecto, las tablas de hash podrán ser útiles para la representación y organización de objetos dentro de cualquier lenguaje de programación dinámicos. (Garg, 2020)

Dentro de las principales ventajas de las Hash Tables cabe recalcar su tiempo de sincronización de datos e información a una estructura de fácil acceso que no requiere de un recorrido para encontrar un solo elemento (tal y como pasa en las linked lists). En efecto, las Hash Tables han demostrado ser más eficientes que nos árboles de búsqueda al manejar y almacenar conjuntos de datos debido a sus propiedades de almacenamiento directo. Otra de las ventajas que pude identificar al realizar esta actividad consiste en la personalización del criterio de organización de los elementos en función de la construcción de la llave. En otros términos, es posible crear distintas versiones de la función hashing lo que puede crear un orden de llenado de la tabla de hashing al igual que generar distintos índices para cada uno de los datos. Por el contrario, algunas de las desventajas primordiales de esta estructura de datos en particular corresponden a la inevitabilidad de las colisiones ya que ningún método de hashing garantiza minimizar la cantidad de colisiones que se producen debido a la existencia de llaves repetidas. Por otra parte, las tablas de Hash no permiten valores nulos por lo que cada casilla de la tabla debe contener un elemento. Y finalmente, que la eficiencia de este algoritmo hash dependerá

netamente de la cantidad de colisiones que se produzcan, por ende, este método es bastante inefectivo cuando hay demasiadas colisiones en los datos. (Khandelwal, 2017)

En lo que respecta esta actividad, el uso de hash tables para almacenar los datos de los vértices y colocarlos en una posición estratégica en base a una llave me permitió fácilmente identificar y sintetizar la búsqueda de los elementos al momento de obtener las incidencias de una IP determinada. A diferencia de otros métodos donde la búsqueda de un elemento se hacía mediante iteraciones (con una complejidad de O(n)) o recursividad (Con una complejidad de O(log(n)), en este caso es posible acceder a la posición del dato con una complejidad de O(1) mediante un cálculo matemático o bien con una complejidad máxima de O(n) para el caso de una colisión hasta encontrar el elemento buscando siguiendo el tipo de recorrido (ya sea lineal o cuadrático). En este caso en específico, considero que el uso de Hash Tables podrá ser esencial para administrar la cantidad de incidencias que ocurren para cada una de las IP´s propuestas con mayor rapidez de acceso dado que estas incidencias representan posibles amenazas a un puerto en específico, se deberá priorizar el tiempo de identificación y buscar una forma de organizar la información a partir del registro dado para comprobar los IP´s pertenecientes a los dispositivos que están atentando contra una red específica. El uso de Hash Tables permite organizar los resultados de mayor manera y tratarlos como parte de una estructura de organización compleja pero muy fácil de implementar y fácilmente modificable también.

Nombre del Método:	**Complejidad:**	**Descripción:**
`Hash(vector<T> &lista)`	O(n)	Constructor por parámetros
`void lineal(vector<T> lista)`	$O(n^2)$	Método de direccionamiento abierto lineal
`void quadratic(vector<T> lista)`	$O(n^2)$	Método de direccionamiento abierto cuadrático
`void updateVertexes(vector<T> &vertexes)`	O(1)	Método para copiar el orden de la tabla a la lista de vectores
`void printTable()`	O(n)	Método para imprimir todos los elementos de la tabla
`int findVposLin(T vertex)`	O(n)	Método de búsqueda de un elemento por el método de direccionamiento lineal

`int findVposQua(T vertex)`	O(n)	Método de búsqueda de un elemento por el método de direccionamiento cuadrático

A continuación, una tabla con las complejidades de cada algoritmo utilizado en la clase Hash:

Referencias:

Garg, P. (2020). *Basics of Hash Tables.* Recuperado el 1 de Diciembre de 2020, de hackerearth.com: https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/

Khandelwal, A. (24 de Abril de 2017). *Advantages and disadvantages of Hash Table.* Recuperado el 1 de Diciembre de 2020, de geeksforgeeks.org: https://practice.geeksforgeeks.org/problems/advantages-and-disadvantages-of-hash-table