Jorge Daniel Cruz Case – A01634536

Fernanda Monserrat Galván Romero – A00344712

Reflexión Actividad integradora 5.2

Durante la realización de la actividad integradora se nos dio como tarea una vez más analizar una bitácora de IPs para tratar de resolver un problema del ataque cibernético que hemos estado trabajando a través del semestre.. Para esto fue necesario de nuevo la lectura de una bitácora con más de 100,000 datos dentro de ella. Para esta tarea, esta vez fue solucionada con la estructura de datos *Hash Table,* combinada con el grafo realizado en la actividad anterior.

Para entender el método de solución de la actividad es necesario tener un buen entendimiento de en que consiste una tabla hash, así como sus características y métodos básicos.

Para esto emplearemos a nuestro amigo Geeks for Geeks, el cual dice, una *Hash Table* es una estructura de datos para el almacenamiento de datos y su acceso de forma más rápida. Esta utiliza el llamado *Hashing,* el cual es una técnica de almacenamiento de datos hacia la misma Hash Table mediante una función Hash. (Geeks for Geeks, 2021).

La eficiencia de la tabla hash depende directamente de que método de hashing se esté utilizando dentro de la tabla. Existen varios métodos de dirección hacia una tabla hash, pero en este caso nos centraremos en el método de dirección abierta, ya que este fue el utilizado para la solución de la problemática.

La *Dirección abierta,* cuenta con distintos métodos para su realización, *Lineal, Cuadrática, Resignación aleatoria y Double Hashing.* Estos te permiten lidiar con las *colisiones,* estas ocurren cuando el índice de un dato es igual a otro, por lo cual, al intentar insertar 2 datos hacia una celda, ocurre una colisión, estos métodos de dirección abierta ayudan a lidiar con esto.

El método *cuadrático,* el cual fue el utilizado para la solución de la problemática, funciona al utilizar un iterador empezando en 1, si hay una colisión le suma el cuadrado del iterador al índice original, lo cual se vería así: $currentIndex = (hashIndex + iterator * iterator.)$.

El método add de la *Hash Table*, consiste en encontrar al estado de la celda, es decir si se encuentra ocupada. Si no lo está, inserta directamente en la celda el nodo con la información, de lo contrario ira buscando por muestreo cuadrático una celda vacía.

Una vez que ya conocemos lo esencial de una *Hash Table*, podemos pasar a algunas consideraciones de esta. Existe una medida para determinar si la tabla es de un tamaño correcto, ya que para obtener el índice de la tabla se está realizando un modulo de la cantidad de datos con el tamaño de la tabla, es necesario que el tamaño de la tabla sea un número primo. Aquí es donde entran las siguientes ecuaciones:

$$a = \frac{n}{m}, n = número\ de\ datos, m = tamaño\ de\ la\ tabla$$

$$S = \frac{1}{2}\left(1 + \frac{1}{(1-a)}\right) = Exito\ aproximado\ al\ insertar\ un\ nodo$$

$$U = \frac{1}{2}\left(1 + \frac{1}{(1-a)^2}\right) = Porcentaje\ aproximado\ de\ colisión\ al\ insertar\ un\ nodo$$

Observándolas podemos deducir que el porcentaje de colisión o de éxito dependen directamente de Alpha, la cual depende del tamaño de la tabla hash. Más colisiones, significan más tiempo de ejecución, por lo cual minimizar estás es ideal al trabajar con tantos datos, esto se obtiene mediante un buen balance entre tamaño de la tabla y cantidad de datos.

a	*S*	*U*
50%	1.5	2.5
75%	2.5	8.5
90%	5.5	50.5

Idealmente tu código debería de estar por debajo del 75% en *a,* con sus demás resultados.

Realizando pruebas con diversos datos, te puedes dar cuenta que entre más te alejes de la cantidad de datos, menor será *a, S y U,* pero la tabla será mayor, en otras palabras, habrá muchas celdas vacías.

Para la solución de la actividad se tomaron 3 números en cuenta, 19387, 20399 y 33863, estos fueron los resultados tomando en cuenta las fórmulas anteriores:

n	*m*	*a*	*a %*	*S*	*U*
13370	19387	0.689637	68.96%	2.11102	5.69076
13370	20399	0.655424	65.54%	1.95106	4.71115
13370	33863	0.394826	39.48%	1.32621	1.86524

Para la solución, terminamos quedándonos con 20,399, que, aunque no nos daba los mejores resultados hablando solamente en cantidades, su tamaño era suficiente. Decidimos en la regla de no sobrepasar *2n*, en otras palabras, el doble de la cantidad de datos, el cual 33,963, se acercaba bastante. Tomando el número elegido obtuvimos una buena complejidad y eficiencia.

Por último, vale la pena mencionar el procesamiento que se le estaba dando a cada ip para su uso en la función hash. Decidimos optar por un método sugerido por el profe, el cual consistía en partir la ip en 4 partes, realizando lo siguiente en ella:

$$IP4 * 256^3 + IP3 * 256^2 + IP2 * 256 + IP1$$

Esta fórmula nos resultará en un número inmenso, por lo cual fue necesario tener en cuenta el límite de datos que le caben a un int. Para esto obtuvimos la siguiente tabla de CCIA:

Tipo de dato	Bits	Rango
int	*8*	*-32768 a 32767*
Long int	*32*	*-2147483648 a 2147483647*
Unsigned long int	*32*	*0 a 4294967295*

Debido a la naturaleza de ciertas ips, el número resultante estaba fuera del rango de un *int* e incluso un *Long int*, por lo que el resultado sería un número negativo. Al final terminamos utilizando el *Unsigned Long Int.*

Por fin la lucha por la eficiencia y contra los ataques cibernéticos termino. Utilizamos la última estructura de datos que el curso incluía, y aunque hayamos localizado al boot master en la actividad anterior, aprendimos como manejar grandes cantidades de datos en una tabla hash de manera eficiente.

Referencias

CCIA. (s.f.). *Capítulo 2: Léxico de C. Tipos básicos de datos, visibilidad y almacenamiento. 2.4) Tipos de datos.* Recuperado de: https://ccia.ugr.es/~jfv/ed1/c/cdrom/cap2/cap24.htm

Geeks for Geeks. (2021). *Quadratic proving in Hashing.* Recuperado de: https://www.geeksforgeeks.org/quadratic-probing-in-hashing/

Geeks for Geeks. (2021). *Hashing data structure.* Recuperado de: *https://www.geeksforgeeks.org/hashing-data-structure/*

Hacker Earth. (2021). *Basics of hash tables.* Recuperado de: https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/