CAMPUS QUERÉTARO

TC1031. PROGRAMACIÓN DE ESTRUCTURAS DE DATOS Y ALGORITMOS FUNDAMENTALES

(GRUPO 826)

"Actividades Integradora 5: Uso de Hash Tables"

Evidencia presentada por la estudiante:

A01706095 - Naomi Estefanía Nieto Vega

Profesor:

Dr. Eduardo Arturo Rodríguez Tello

Fecha de entrega:

Jueves 3 de junio de 2021

Actividad Integradora 5: Uso de Hash Tables

Naomi Estefanía Nieto Vega

***Resumen*—En este documento se plantea la solución para la quinta actividad integradora, que da solución a la problemática presentada que es conocer las IP que son adyacentes mediante el uso de Hash Tables.**

***Index Terms*—data structures, hash table, adjacency, map.**

I. INTRODUCCIÓN

Hoy en día los algoritmos de ordenamiento y las estructuras de datos son muy importantes en el área computacional y en muchas más áreas en los que son utilizados ya que de acuerdo con la literatura un algoritmo es una serie o secuencia ordenada de pasos que dan solución a un determinado problema. No obstante, pueden existir distintos algoritmos que den solución a un mismo problema y es aquí donde entra la importancia de implementar el más eficiente en cuanto a tiempo y recursos ya que estos deben ser utilizados correctamente y de esta forma la computadora podrá darnos resultados muy buenos.

Asimismo, al conocer distintos tipos de estructuras de datos y su implementación podemos hacer uso de ellos para mejorar el manejo que tenemos sobre los algoritmos de acuerdo con las necesidades que tenga el problema, y así proveer una solución adecuada, eficiente y rápida para el problema presentado.

Para el desarrollo de esta cuarta situación problema, principalmente nos enfrentamos al problema del manejo de una bitácora con gran volumen de datos, aunado a esto debemos realizar búsquedas dentro de la bitácora haciendo uso de las hash tables para conocer las adyacencias que tiene una IP dada por el usuario, esto quiere decir, la cantidad de conexiones o "vecinos" que tiene dicha IP y mostrarlos en consola.

Por lo tanto es importante conocer las consideraciones o limitaciones que tenemos, ya que para este caso en específico contamos con un formato de bitácora con un gran volumen de datos y la solución se debe adecuar a esto ya que es un factor muy importante a considerar para plantear una solución sobretodo eficiente para reducir el tiempo de espera, dicha solución se explicará a continuación.

II. DESARROLLO

Las estructuras de datos son formas de organización y representación de datos o información, en el caso de la programación, son formas de organización que nos permiten utilizar estos datos de una forma efectiva. Existen dos tipos de estructuras de datos:

- Estructuras de datos lineales
- Estructuras de datos no lineales

Su principal diferencia es que la estructura de datos lineal tiene a sus elementos de datos dispuestos de manera secuencial y cada uno está conectado con su elemento anterior y siguiente. Por otra parte, las estructuras de datos no lineales no tienen una secuencia establecida para conectar a sus elementos, por lo que un elemento puede tomar varias rutas para conectarse con otros elementos. Este tipo de estructuras son un poco más complejas de implementar pero son más eficientes en cuanto a uso de memoria en la computadora.

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Figura 1. Complejidad de las principales estructuras de datos. Obtenido de: https://www.bigocheatsheet.com/

Las tablas hash o mejor conocidas como hash tables en inglés, son una estructura de datos que almacena datos de forma asociativa, esto quiere decir que se almacenan en un tipo o formato de matriz en la que cada valor de datos cuenta con un key o índice único, de esta forma el acceso a los datos se vuelve muy eficiente y rápido si conocemos la llave de los datos deseados.

Por otra parte, también existe la función de hashing, que en la mayoría de los casos funciona muy bien a comparación de otras estructuras de datos como Arrays, Linked List o BST, ya que con el hashing tenemos una complejidad de O(1) en tiempo de búsqueda en promedio y un O(n) en el peor de los casos.

La función hash es una función que asigna un número grande o una cadena a un entero pequeño que se puede utilizar como índice en la tabla hash. Una función eficiente de hash tendría las siguientes propiedades:

- Eficientemente computable
- Distribuye uniformemente las llaves (cada posición de la tabla es igualmente probable para cada llave)

Por otra parte, dada una función hash nos da un número pequeño para una llave grande, existe la posibilidad de que dos llaves den el mismo valor, cuando una llave recién asignada se asigna a un elemento ya ocupado en la tabla hash se le conoce como colisión, y para ello existen técnicas de manejo de colisiones:

- Encadenamiento (Chaining)

Esta técnica consiste en hacer que cada celda de la tabla hash apunte a una lista vinculada de registros que tengan el mismo valor de función hash. Esta técnica es simple pero requiere memoria adicional fuera de la tabla.

- Direccionamiento abierto (open addressing)

Por otra parte, en el direccionamiento abierto todos los elementos se almacenan en la propia tabla hash, de esta forma cada entrada de la tabla tiene un registro, y al buscar un elemento se va examinando uno por uno cada espacio de la tabla hasta encontrar el elemento deseado o no encontrarlo en caso de que no esté en la tabla. [3]

Entonces, hablando en términos de la eficiencia, las operaciones de inserción y búsqueda son muy rápidas independientemente del tamaño de los datos, ya que este tipo de estructura lo que hace es usar una hash table como matriz de almacenamiento y con la técnica de hashing genera los keys o índices únicos en el cual se insertan los datos. [2] En la siguiente figura se puede observar una tabla con las complejidades de las principales funciones mencionadas anteriormente.

Hash table	Average	Worst
Search	O(1)	O(n)
Insertion	O(1)	O(n)
Deletion	O(1)	O(n)

Figura 2. Complejidad de las principales funciones de las hash tables.

La importancia de las hash tables en una situación problema de esta naturaleza es que este tipo de estructura de datos es muy eficiente cuando tenemos que buscar algún elemento específico en una bitácora o set de datos muy grande, ya que tienen una complejidad de O(1). [4] Por lo que para el caso de la búsqueda de las IPs adyacentes a una IP dada por el usuario se convierte en una estructura ideal y bastante rápida para encontrar la información. Además son muy utilizadas en particular para matrices o arrays asociativos, indexación de bases de datos, cachés y conjuntos.

III. CONCLUSIÓN

En conclusión, gracias a esta actividad integradora pude comprender mejor el funcionamiento y la implementación de las hash tables, además de su importancia y aplicaciones en el mundo real. También me di cuenta de sus ventajas y desventajas para ciertos casos en los que utilizarlas resulta más efectivo y otros en los que no tanto. Me sorprendió lo rápidas y efectivas que son para una situación problema en la que hay que buscar datos en grandes volúmenes como lo es la bitácora que utilizamos ya que se vuelve muy efectiva la búsqueda de información.

REFERENCIAS

[1] Big-O Algorithm Complexity Cheat Sheet (Know Thy Complexities!) @ericdrowell. (2021). BigO. https://www.bigocheatsheet.com/

[2] Data Structure and Algorithms - Hash Table - Tutorialspoint. (2021). Tutorialspoint. https://bit.ly/3wTzxYH

[3] GeeksforGeeks. (2018, 6 septiembre). Hashing | Set 1 (Introduction). https://www.geeksforgeeks.org/hashing-set-1-introduction/

[4] What's the point of a hash table? (2010, 1 febrero). Stack Overflow. https://stackoverflow.com/questions/2179965/whats-the-point-of-a-hash-table