Tecnológico de Monterrey

Act 5.2 - Actividad integral de Hash

Reflexión personal

TC1031

Programación de estructuras de datos y algoritmos fundamentales

Profesor: Dr Eduardo A. Rodríguez Tello

Rogelio Guzman Cruzado A01636244

Introducción

En 1953, Hans Peter Luhn inventó la técnica del Hashing mientras trabajaba como científico de investigación en IBM, constantemente se planteaba resolver retos en el campo de la recuperación de información. ordenando químicos que habían sido traducidos a un código, aquí fue cuando decidió poner toda la información que se le daba en un "bucket", prediciendo que esto haría que la búsqueda de información fuese más veloz, y así es como nació la idea base del hashing, la cual ha sido modificada y mejorada desde entonces por distintos científicos, creando variedad de algoritmos de hashing. Ahora los hashmaps son esenciales en varios campos de la ciencia computacional, como la criptografía, los servicios de nubes, y en las investigaciones con cantidades masivas de datos.

recuperado de: https://www.geeksforgeeks.org/importance-of-hashing/

Importancia del Hashing

El hashing proporciona una manera segura y ajustable de recuperar información. En nuestra situación problema estamos trabajando con más de 10 mil datos, Una tabla hash trabaja en promedio en una complejidad temporal de O(1) en la búsqueda de datos. A diferencia de una estructura de datos lineal, como un array o lista, las cuales trabajan en una complejidad promedio de O(n), por lo que si decidimos utilizar alguna de estas, no solo tendríamos que crear varias listas para almacenar tanto la llave como el valor, si no que tendríamos un programa mucho más ineficiente. Parte de la importancia del hashing es la seguridad que provee, debido a que una contraseña que ha sido hasheada no puede ser "modificada, robada, o alterada" (https://www.geeksforgeeks.org/importance-of-hashing/)

recuperado de:
https://betterprogramming.pub/what-are-hash-tables-and-why-are-they-amazing-89cf52246f91

Clase	Método	Complejidad
Graph.h	validate(string ip)	O(n)
Graph.h	getIPSummary()	O(n^2)
Graph.h	loadGraphList()	O(n)
HashTable.h	print()	O(n)
HashTable.h	add(KeyVal, Value)	O(1)
HashTable.h	find()	O(n)
HashTable.h	getDataAt()	O(1)
HashTable.h	addDegree()	O(1)
HashTable.h	totalOverFlowPrint()	O(1)
HashTable.h	printAt(int index)	O(1)

Conclusión

Como podemos ver, la mayoría de nuestros métodos en la implementación trabajan en una complejidad de O(1), probando por qué las Hashtable trabajan en una velocidad promedio de O(1). Pero existe una manera en la que nuestra complejidad se puede ver afectada, y esto sucede cuando existe un número muy elevado de colisiones en nuestra tabla, llevando en el peor caso nuestra complejidad a un O(n), esto sucede principalmente por el manejo de colisiones que se tiene que utilizar. (https://stackoverflow.com/questions/9214353/hash-table-runtime-complexity-insert-search-and-delete)