**Hash Table**

viernes, 21 de abril de 2023

11:43 p. m.

Hash Tables

The Roadmap Intro: [Hash Table Data Structure | Illustrated Data Structures](https://www.youtube.com/watch?v=jalSiaIi8j4)

Realpython ref: <https://realpython.com/python-hash-table/>

Programiz ref: <https://www.programiz.com/dsa/hash-table>

Edureka ref: <https://www.edureka.co/blog/hash-tables-and-hashmaps-in-python/>

Hash table core features:

* Create an empty hash table
* Insert a key-value pair to the hash table
* Delete a key-value pair from the hash table
* Find a value by key in the hash table
* Update the value associated with an existing key
* Check if the hash table has a given key

Nonessential:

* Create a hash table from a Python dictionary
* Create a shallow copy of an existing hash table
* Return a default value if the corresponding key is not found
* Report the number of key-value pairs stored in the hash table
* Return the keys, values, and key-value pairs
* Make the hash table iterable
* Make the hash table comparable by using the equality test operator
* Show a textual representation of the hash table

Basic Hash Table Structure

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

The Hash Function

One essential concept in this specific data structure is the Hashing Function which is the one responsible to allocate the key to a specific position within this DS.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

It's said that a good HT is determined on how often Hash Collisions happens, and there are two main way to deal with this cases

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

In Separate Chaining, instead of overwriting the Key associated with a value, the corresponding place of the former key is now directed to a link list that stores the values, and this is why in the worst case scenario, the complexity of HT function is O(n)

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Hash Table Operations

The main HT functions are: Setting, Getting, Updating and Deleting Key-Value pairs from the structure and all of them have the same complexity: O(1) in the avg case scenario, but in a worst case scenario, with multiple hash collisions, the complexity goes up to O(n), since it typically goes to a linked list.

Strengths and Weaknesses

**Strengths**

**Fast Average-Case Access:** Hash tables provide fast average-case time complexity for common operations like insertion, deletion, and retrieval. This makes them efficient for tasks such as looking up values by keys.

**Constant-Time Average-Case Performance:** When hash collisions are rare and the hash function is well-distributed, hash table operations have a constant-time average-case performance, which means the time required doesn't depend on the number of elements in the table.

**Flexible Key Types:** Hash tables can often handle a wide range of key types, including integers, strings, and custom objects, as long as a suitable hash function can be defined for them.

**Versatility:** Hash tables can be used for various applications, such as implementing data caches, symbol tables, and associative arrays.

**Dynamic Sizing:** Many hash table implementations automatically resize themselves (rehash) to maintain a low load factor, ensuring that performance remains efficient as the number of elements changes.

**Weaknesses**

**Worst-Case Performance:** In the worst-case scenario, where there are many hash collisions, hash table operations can degrade to O(n) time complexity, essentially becoming a linked list. This worst-case performance can be exploited in certain situations.

**Deterministic Order:** Hash tables do not maintain a deterministic order of elements. If you need to iterate through elements in a specific order, you may need to use additional data structures in conjunction with the hash table.

**Hash Function Sensitivity:** The efficiency of a hash table depends heavily on the quality of the hash function. A poor hash function that generates a lot of collisions can significantly degrade performance.

**Memory Overhead:** Hash tables can have a higher memory overhead compared to some other data structures due to the need to allocate space for potential hash collisions.

**Not Suitable for Range Queries:** Hash tables are not well-suited for range queries (e.g., finding all keys within a specific range), as they don't provide built-in support for ordering elements.

**Limited Sorting:** Hash tables are not designed for sorting elements. If you need to retrieve elements in a sorted order, you'll need to employ a different data structure.

Heap Use Cases

Data Caching: Hash tables are often used to implement caching systems. They store frequently accessed data in memory, such as web page content, database query results, or API responses, to reduce the time and resources needed to fetch the data from a slower source.

Dictionaries and Symbol Tables: Hash tables are a fundamental data structure for implementing dictionaries and symbol tables, where words or identifiers (keys) are associated with their meanings or values. This is a common use case in programming languages and compilers.

Frequency Counting: Hash tables can be used to count the frequency of elements in a collection (e.g., words in a text document or items in a shopping cart). They allow efficient updates and retrieval of element frequencies.

Caching of Expensive Function Calls: When function calls are computationally expensive, hash tables can be used to cache the results of previous function calls based on the function's arguments. This is known as memoization and can significantly speed up algorithms.

Associative Arrays: Hash tables can be used to implement associative arrays or maps, where unique keys are associated with values. This is a fundamental data structure in many programming languages (e.g., JavaScript objects, Python dictionaries).

Database Indexing: In database systems, hash tables can be used for indexing, allowing for efficient data retrieval based on specific criteria, such as primary keys or unique identifiers.

Implementing Sets: Hash tables can be used to implement sets, ensuring that each element is unique. This is helpful for tasks that require checking for membership or removing duplicates from a collection.

Password Storage: Hash tables, combined with cryptographic hash functions, are used to securely store user passwords. Storing hashed passwords instead of plain text helps protect user data.

Graph Algorithms: Hash tables can be employed in graph algorithms to efficiently store and look up adjacency information. They can be used to represent graphs as adjacency lists.

Compiler and Interpreter Symbol Tables: Hash tables are used in compilers and interpreters to store information about variables, functions, and labels, making it possible to quickly look up symbols during compilation or interpretation.

URL Routing in Web Applications: In web development, hash tables can be used for URL routing, mapping URLs to controller actions or specific content, allowing for efficient handling of HTTP requests.

Language Models and Spell Checkers: Hash tables can be used to build language models and spell checkers by efficiently storing and retrieving word frequencies and suggestions for misspelled words.