Faculdade de Design,
Tecnologia e Comunicação
Universidade Europeia

Hash Tables

Data Structures

Fernando Marson

fernando.marson@universidadeeuropeia.pt

Introduction

- A hash table is a data structure that implements an associative array, which allows for the **storage of key-value pairs**.

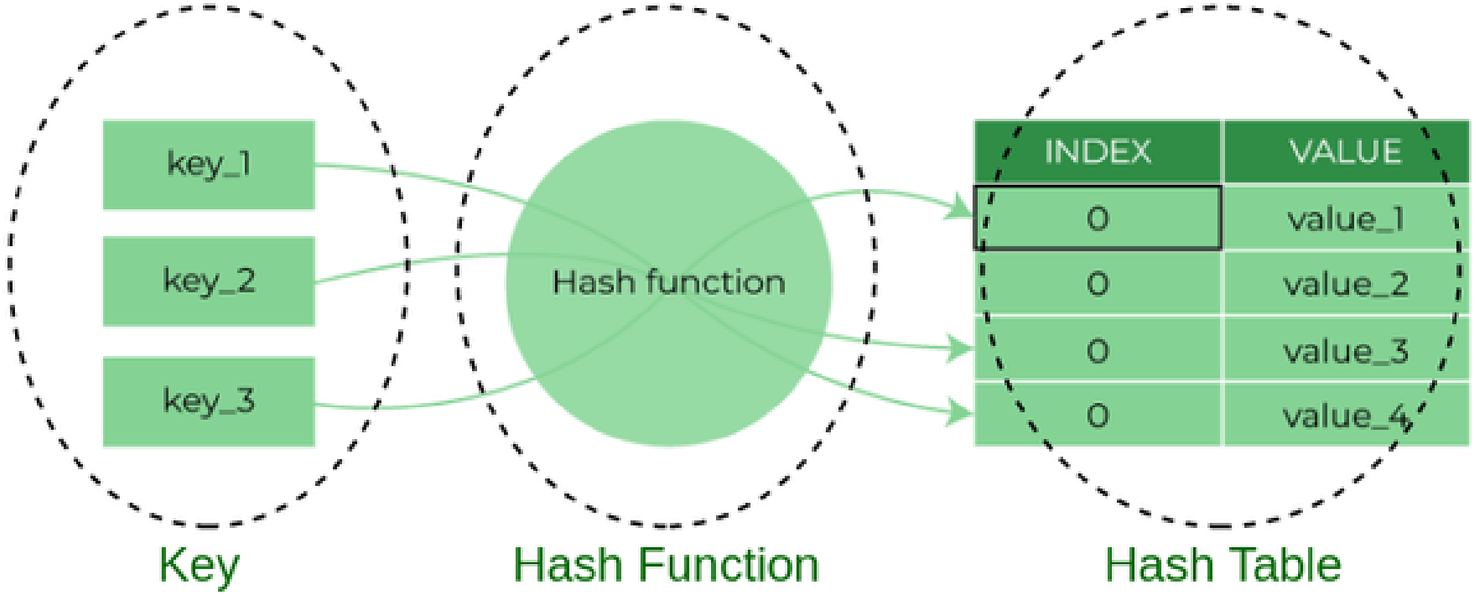
- In this context, a **key** is a **unique identifier** used to **access a corresponding value**, enabling fast data retrieval.

- The primary strength of hash tables lies in their ability to provide average-case **constant time complexity**, **O(1)**, for operations such as **insertion**, **deletion**, and **search**.

Introduction

- At the core of a hash table is the concept of a **hash function**.
- A hash function takes **a key as input** and **computes an index** in the underlying array where the corresponding value will be stored.
- This mapping process **transforms the key into an integer**, which then determines the **specific location in the array**.

Components



- **Key**: Data to be stored or retrieved.
- **Hash Function**: Transforms the key into a unique index.
- **Hash Table**: An array that stores data at unique indices for efficient access.

Benefits

- Hash tables **combine the benefits of arrays** (constant time access) with the **flexibility of associative arrays** (**key-value pairs**), making them an essential data structure in various applications, including **databases** and **caching systems**.

Hashing

https://youtu.be/VeYKEMY2F9k

Hash Function

- A hash function is a crucial component of hash tables, responsible for transforming input data (**keys**) into a fixed-size integer value, known as a **hash code** or **hash value**.
- This **integer value** serves as an **index** in the underlying array of the **hash table**, allowing for efficient data retrieval and storage.

```
public int SimpleHash(string key) {
    int hash = 0;
    foreach (char c in key) {
        hash += (int) c;
    }
    return hash % tableSize; // Assume tableSize is the size of the array }
```

Hash Function

- What happens when two key applied to same hash function produce the same value?

Collisions in Hash Tables

- A collision in a hash table occurs **when two different keys produce the same hash value** and, consequently, **map to the same index in the underlying array**.

- Since hash tables rely on a hash function to determine an index for storage and retrieval, **collisions are an inherent challenge** due to the **finite size of the array** and the **potentially vast range of input keys**.

Collisions in Hash Tables

- Chaining
- Open Addressing
- Rehashing

Collisions in Hash Tables

Chaining

- In this method, each index in the hash table array contains a linked list (or another collection) of all entries that hash to that index. When a collision occurs, the new key-value pair is simply added to the list at that index.

- **Advantages**: This method is simple to implement and can handle an arbitrary number of collisions without needing to resize the table.

Hashing
Separate Chaining

https://youtu.be/LRtKQdsJC3o

Collisions in Hash Tables

Open Addressing

- In this approach, when a collision occurs, the algorithm searches for the next available index in the array using a probing sequence. Common probing methods include linear probing, quadratic probing, and double hashing.
- **Linear Probing** Example: If a collision occurs at index i, the algorithm checks indices i+1, i+2, etc., until it finds an empty slot.

Hashing
Linear Probing

https://youtu.be/98Y0UDZ9vvs

Hashing
Quadratic Probing

https://youtu.be/0CFJAkpnhBg

Collisions in Hash Tables

Rehashing

- If the **load factor** (the ratio of stored entries to the total number of slots) **exceeds a certain threshold**, it may be beneficial **to resize the hash table and rehash all existing entries**.
- It reduces collisions by **providing more space**.
- **Implementation**: When resizing, create a new array with a larger size and re-insert all existing key-value pairs using the **new hash function**.

In-game applications

- **Inventory Management**: Hash tables can efficiently manage player inventories by associating unique item IDs (keys) with item attributes (values). This allows for quick lookups, additions, and removals of items, enabling seamless inventory management in games.

- **Game State Storage**: In complex games, tracking various states (e.g., player progress, achievements, or level completion) is essential. Hash tables can store these states using unique identifiers as keys, allowing for rapid access and updates during gameplay.

In-game applications

- **Caching Resources**: Hash tables can serve as a caching mechanism for game assets such as textures, sounds, and models. By using asset names or paths as keys, developers can quickly retrieve loaded resources, reducing loading times and improving performance.

- **Character Attributes and Skills**: For role-playing games (RPGs) or strategy games, hash tables can store character attributes (like strength, agility, etc.) or skills associated with unique character IDs. This allows for fast access to a character's properties during gameplay.

In-game applications

- **Event Management:** In games with numerous interactive elements, hash tables can manage events and callbacks.
- By mapping **event names** (**keys**) to their corresponding handler functions (**values**), developers can efficiently trigger specific actions based on user input or game conditions.