Chapter 1

Hash Tables

A hash table, also known as a hash map, is a data structure that implements an associative array abstract data type, a structure that can map keys to values. It uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

1.1 Structure of a Hash Table

A hash table is composed of an array (the 'table') and a hash function. The array has a predetermined size, and each location (or 'bucket') in the array can hold an item (or several items if collisions occur, as will be discussed later). The hash function is a function that takes a key as input and returns an integer, which is then used as an index into the array.

1.2 Inserting and Retrieving Data

When inserting a key-value pair into the hash table, the hash function is applied to the key to compute the index for the array. The corresponding value is then stored at that index.

When retrieving the value associated with a key, the hash function is applied to the key to compute the array index, and the value is retrieved from that index.

1.3 Handling Collisions

A collision occurs when two different keys hash to the same index. There are several methods for handling collisions:

- **Chaining (or Separate Chaining)**: In this method, each array element contains a linked list of all the key-value pairs that hash to the same index. When a collision occurs, a new key-value pair is added to the end of the list.
- **Open Addressing (or Linear Probing)**: In this method, if a collision occurs, we move to the next available slot in the array and store the key-value pair there. When looking up a key, we keep checking slots until we find the key or reach an empty slot.

1.4 Time Complexity

In an ideal scenario where hash collisions do not occur, hash tables achieve constant time complexity $O(1)$ for search, insert, and delete operations. However, due to hash collisions, the worst-case time complexity can become linear $O(n)$, where n is the number of keys inserted into the table.

Using good hash functions and collision resolution strategies can minimize this issue and allow us to take advantage of the hash table's efficient average-case performance.

Appendix A

Answers to Exercises

Index