General Assembly

STUDY GUIDE

HASH TABLES DEEP DIVE

Hashing Recap

Generally, **hashing** refers to the process of taking a key (i.e., a piece of data), scrambling it with a**hash function**, and producing an index that's used to sort the key into a **hash table**.

» A **hash function** is the algorithm that scrambles keys in order to produce indices. A hash function should:

- Always return the same output, given the same input
- Be simple and efficient
- Distribute values evenly throughout the hash table
- Avoid collisions

» A **hash table** is a list-like data structure that's designed to quickly store and retrieve key data records.

- Before a key can be stored, it must be mapped (with a hash function) to an index in the table or to the address of a memory location.

Open and Closed Addressing

The one rule of a hash function is that, for the same input, it must always generate the same output, but two different inputs could happen to have the same output; this is called a **collision**.

» You need to provide a solution to collisions in your implementation.

» There are two main ways to resolve collisions:

- Open addressing (probing)
- Closed addressing (chaining).

Open Addressing (Probing)

If the index generated for a key is already taken,**open addressing** jumps to somewhere else in the table to store your key.
- This process is also called **probing**.
- There are three common types of open addressing:
- **Linear probing** moves one slot to the right until it finds an open index.
- **Quadratic probing** finds an open index by moving one step to the right, then four, then nine, then 16, then 25, etc.
- **Double hashing** uses a secondary hash function to find an open index.

Closed Addressing (Chaining)

The other method for resolving collisions is **closed addressing** (or **chaining**). This method is a more elegant approach to a hash table implementation.
- Each slot in the hash table is built as a bucket that can hold as many keys as you want.
- If the hash function generates the same index for two keys, we just add them to the bucket. - These buckets are implemented as a linked list.

Chaining vs. Probing

Any hash table implementation must include three basic methods:
- **search**
- **insert**
- **remove**

The differences between chaining and probing become really clear when we look at how they each accomplish these methods.

Method	How It's Done With Probing	How It's Done With Chaining
search	hash the key, see if it's at that index, and probe until you find it or find an empty slot.	hash the key, then search the data structure at that index for that key.
insert	hash the key, then put it at the index generated; if that index is taken, probe until you find one that's available.	hash the key, then store it in the data structure at that index.
remove	Essentially a search followed by a deletion, but you must set an indicator that an element was deleted or a probe might stop there when it should keep jumping.	hash a key, then delete the data from the data structure located at that index.

Uses of Hash Tables

Hash tables are great in situations where we need to locate and retrieve a record in a collection of millions or billions of entries, for example:
- Accessing records in a database
- Locating items in a computer's memory
- Spell checkers

Additional Resources

Because hash tables are so widely used, they're an important job interview topic. Be sure to know the different ways of resolving collisions and the trade-offs for each.

Here are some other resources to review:
- A visualization tool for closed hashing using buckets
- Twenty hashing-related questions