Hashing

The goal of a hash function is to minimize collisions. But in practice, some collisions are unavoidable, thus implementing a hash must include some collision resolution policy. This can be separated into two classes of techniques: open hashing (separate chaining) and closed hashing (open addressing).

Open Hashing

Open hashing is when collisions are stored outside of the table. The simplest implementation stores all values that hash to a particular index in a linked list at that index. We can sort the linked lists upon insertion to improve access time. Open hashing is only ideal when the hash table is stored in main memory.

Closed Hashing

Each record R with key k_R has a **home position** given by $h(k_R)$. If R is to be inserted, yet another record already occupies $h(k_R)$, then in a closed hashing scheme, R must be stored elsewhere in the table. Naturally, we must also use the same process when accessing the record to get to the same index.

Bucket Hash

A hash table of M slots is divided into B buckets of M/B slots. The hash function will assign each record to the first slow within a bucket, unless that slow is already occupied in which case, the bucket slots are sequentially searched until an open one is found. If a bucket becomes completely full, all records to be inserted into that bucket will now go to an overflow bucket of sufficient capacity at the end of the table, regardless of key value.

Linear Probing

Collision resolution can be viewed as generating a list of potential hash table slots. This sequence is known as the probing sequence and the function which generates it is the probe function. A common function is

$$p(K, i) = i.$$

That is a function which when given key K and an integer i representing the position in the probe sequence, returns i. If the position at $p(K, i)$ is filled, we simply try $p(K, i+1)$. This sequential search for an open slot is called **linear probing**. This method can be problematic when it leads to **primary clustering**, where certain indices become more likely to be probed as more insertions occur.

Better CRMs

- Pseudo-random Probe: The probe function $p(K, i) = \texttt{Perm}[i-1]$, yields the probe sequence $(h(K) + r_i) \bmod M$, where r_i is the ith index for the random permutation of the numbers 1 through $M-1$, in `Perm`, which is a fixed global array.
- Quadratic Probe: The probe function $p(K, i) = c_1 i^2 + c_2 i + c_3$ for integers c_j.

These methods avoid the problem of primary clustering, but since they're entirely dependent on the home position, they'll produce the same probe sequence for all keys that hash to the same number. This causes **secondary clustering**.

- Double hashing: Avoid problem of secondary clustering by incorporating the key into the probe function. For example, $p(K, i) = i \star h_2(K)$, where h_2 is another hashing function. Combining a double hash with either the pseudo-random or the quadratic probe yields a better CRM.

Deletion

When deleting we must be careful not to disrupt future searching. Therefore, once a key to be deleted is found it is marked with a **tombstone**. During searches, the tombstone is just skipped, but during insertion, the tombstone allows the record to be overwritten.

Closed Hashing Analysis

Define the **load factor** to be $\alpha = \frac{N}{M}$ where N is the number of records currently in the table. The expected number of probes is approximated by

$$\frac{1}{1-\alpha} = \frac{M}{M-N}.$$

The true cost of for insertion or unsuccessful searching is given by

$$\frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right),$$

and for deletion or successful searching is

$$\frac{1}{2}\left(1 + \frac{1}{(1-\alpha)}\right).$$