Table Size with Probing

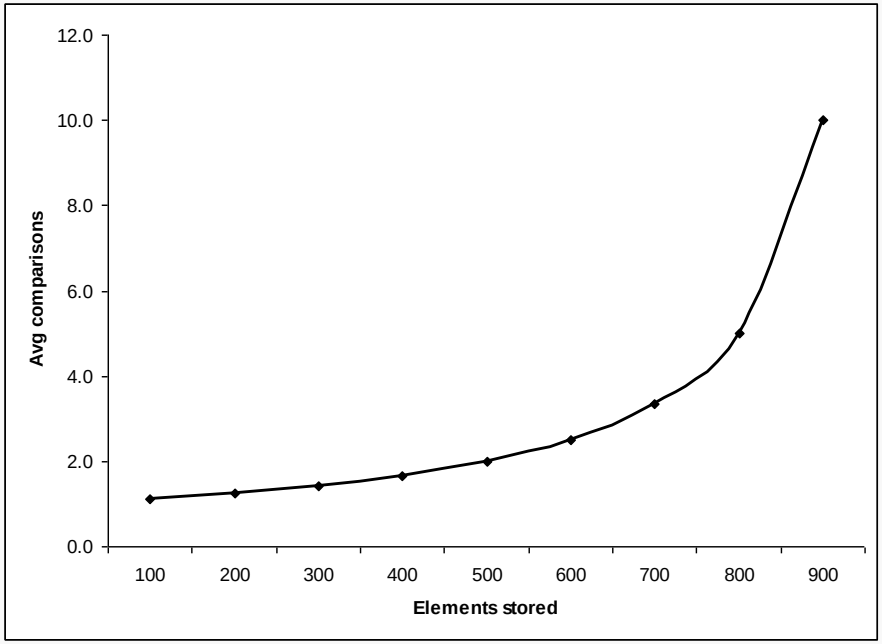
If a probe strategy is used to resolve collisions then empirical evidence indicates that search performance is likely to degrade significantly if more than 70% of the table slots are filled.

In particular, if the table has N slots and stores K elements then the average number of comparisons performed on a search is:

$$\theta\left(\frac{1}{1-K/N}\right)$$

load factor

If we take N = 1000 and plot this, we get:



Re-hashing

If a probe strategy is used to resolve collisions then we may choose to resize the table itself if its load factor gets too large.

However, this not only requires copying all the data elements from the old array into the new one, but also re-hashing all the key values (or at least recomputing the remainder).

Even so, there's little alternative if you adopt a probing strategy to cope with collisions and your table performance degrades…

Table Size with Chaining

If chaining is used to resolve collisions then the table never really fills up.

It can be shown that the average number of comparisons on a search is: $\theta(1+K/N)$

QTP: If we use a table of 1000 slots, how does this compare with the earlier example?

QTP: Is that a fair comparison? Why?

Other Considerations

Regardless of how we handle collisions, we would like the number of them to be small.

Clearly the table size has something to do with that.

When does a collision occur?

When we have two different key values such that: $H(K_1) - H(K_2) \equiv \text{mod}(N)$

Now that's equivalent to there exists a positive integer q such that: $H(K_2) = H(K_1) + qN$

Another way of looking at this is that collisions occur when the hash function produces values that are congruents mod N, and the number of possible congruents mod N is strictly determined:

$$\frac{-H(K_1)}{N} \leq q \leq \frac{INT\_MAX - H(K_1)}{N}$$

So, the larger N is the fewer possible congruents there are…

Prime Table Size

Should the table size be prime?

If chaining is used, it doesn't seem to matter, because whether we mod by a prime number has no effect on the likelihood of getting congruent slot numbers. However…

If a probe strategy is used then a prime table size may improve the odds the probe strategy will eventually examine all table slots can be increased (e.g., if quadratic probing is used).

But… if probing needs to examine a large number of table slots we have already lost the performance cost that motivated using a hash table in the first place…

Still, there are some hash functions for which a prime table size may improve results…

Prime Table Size

Consider a string hash function that takes character strings, and applies a shift-and-add approach to compute an integer.

Say we employ an r-bit shift, then for a k-byte string we're computing an integer of the form:

$$HashValue = b_1 \cdot (2^r)^{k-1} + b_2 \cdot (2^r)^{k-2} + b_3 \cdot (2^r)^{k-3} + \mathrm{L} \; + b_{k-1} \cdot (2^r)^1 + b_k$$

But if the table size was a power of 2, say 2^r, then modding by that would effectively ignore the contributions of all but the low byte… which would almost certainly not produce a good scattering of keys throughout the table.

Since it's natural to work with powers of 2 in such algorithms, selecting a prime table size would eliminate this effect.