Lecture 27: Hash Functions

We need a Universal Hash Function. Here you go: assume that all inputs are base p, for some prime p. p will be the size of our Hash Table. Now, English words are base 26, but we can translate them to base p quite easily. So an English word will be $w = w_1 w_2 ... w_x, 0 \leq w_i < p$.

Suppose that the longest english word has length x. Then we will choose a random number (base p) $a = a_1 a_2 ... a_x$. We choose this random number ONCE when we create our hash table, and then keep that random number until we delete the hash table. The hash function is then this:

$h(w) = (\sum_i a_i w_i) \bmod p$.

Collision Resolution

Another way to handle collisions is *probing*, wherein we just have an array of either key/value pairs (no linked lists). There are many variants.

In *linear probing*, if $h(k) = i$ and $A[i]$ is taken, we try $A[i+1]$ and then $A[i+2]$ and so on. If you reach the end of the array, you loop back to the beginning.

That is, $h(k, i) = (h(k) + i)\%m$, where i is the number of failed inserts, and m is the size of our hash table.

Question 1 *What problems can you see arising when using linear probing?*

Quadratic Probing

As before, if $h(k) = i$ and $A[i]$ is taken, try somewhere else. Except now, $A[i+1]$ is followed by an attempt at $A[i+4]$, then $A[i+9]$ and so on. As before, loop back to the beginning if you hit the end of the array.

That is, $h(k, i) = (h(k) + i^2)\%m$.

Two different items mapped to i and $i+1$ respectively will follow very different paths. We don't get *primary clumping* as we did for Linear Probing.

Secondary clumping will only occur amongst all of the items which were originally mapped to index i. Note that chaining has this form of clumping as well.

Question 2 *Using the hash function $h(k) = k\%10$, determine the contents of the hash table after inserting $1, 11, 2, 21, 12, 31, 41$.*

The load factor of a hash table is the number of stored elements divided by the number of indices.

When you use chaining, you generally try to keep the load factor below 1.0. If you load factor becomes too large, you should resize the array and rehash the contents (and possibly use a new hash function).

When you use probing, you **must** make sure the load factor of the hash table is reasonable. If the hash table is completely full, you simply cannot add new elements. As you approach completely

full, operations take ridiculously long. Usually you'll want to keep the load factor below 0.5.

In fact, if your load factor is above 0.5, you cannot guarantee that quadratic probing will find an empty bucket, even if the hash table size is prime.

Question 3 *Using the hash function* $h(k) = k\%7$*, determine what happens after inserting* $14, 8, 21, 2, 7$.

If the table size is not prime, the problem is even worse.

Question 4 *Using the hash function* $h(k) = k\%9$*, determine what happens after inserting* $36, 27, 18, 9, 0$.

If your hash table has a prime size, then the first $\frac{m}{2}$ probes are guaranteed to go to distinct locations, meaning that you will find a location if the load factor is no more than 0.5.

Double Hashing

This avoids both primary and secondary clumping. You have a second hash function h'(k).

If $h(k) = i$ is taken, then try, in order, $i + h'(k)$, $i + 2h'(k)$, $i + 3h'(k)$, ...

That is, $h(k, i) = (h(k) + i \cdot h'(k))\%m$.

This is better because even amongst all the items mapped to i, they will follow very different paths.

A good choice of secondary hash function can ensure you always find a free slot as long as the load factor is smaller than 1 (but you still want to keep the load factor below 0.5 for performance issues).

An example of a good secondary hash functions is $h'(k) = p - (k\%p)$, where p is a prime smaller than m.

Removal

Question 5 *There is a serious problem if we want to delete an item from a hash table that uses probing. What is it?*

The "simplest" solution is to add a "deleted" flag to an item in the hash table that has been removed. When searching, if you hit a deleted item, you keep searching.

Now the load factor should consider both items and deleted items, since search will get unreasonably long otherwise.

After you have too many deleted items, you will want to rehash your entire hash table contents, so as to maintain a reasonable runtime (you don't necessarily have to resize the hash table though). This takes a long time!

In an amortized sense, it's okay, since you have to do a bunch of deletes before you have to re-hash.