More Hashing

- Remember given a hash table with n slots and m items the load factor is defined as:
$$\alpha = \frac{\text{items} = m}{\text{slots} = n}$$
The book has n and m reversed.
- The average number of probes to find an element is going to be
$$\Theta(1 + \alpha)$$
- Which means that α can be quite large.

It has to be Random!

- Remember the analysis in the last lecture depends on keys being inserted at random.
- Or more precisely (given a hash function h)
$$h(K)$$
to be a uniformly distributed random variable.
- There are lots of good hash functions, a good to pick is:
$$h(K) = K \mod N$$
- It is good to pick the table size as a prime number.
- If the keys are randomly distributed then $h(K)$ will be.

What happens if things are not random?

- Imagine a malicious demon who wants your program to fail.
- What he does is look at your hash function h and pick keys $K_1, \ldots, K_m$ such that:
$$h(K_1) = h(K_2) = \cdots = h(K_m)$$
- So that all things are placed in the same slot.
- Resulting in worst case behaviour $\Theta(m)$
- Worse your strings might not be randomly distributed.

How to outwit your demons

- Common idea in algorithm design.
  - Often used in Quicksort, or in binary search trees.
- To improve the average case behaviour randomise things before you start.
- To approaches:
  - Randomize the data
  - Randomize the hash function.

Universal Hash Classes of Hash Functions

- Let $\mathcal{H}$ be a finite class of hash functions such that for all keys $k \neq l$ then the number of hash functions $h \in \mathcal{H}$ such that

  $$h(k) = h(l)$$

  is a most

  $$\frac{|\mathcal{H}|}{n}$$

  (where n is the number of slots in the hash table).

What does this mean?

- Given a randomly chosen hash function from $\mathcal{H}$
- Then the probability of collision between distinct keys k and l is no more than $1/m$.
- This is independent of the dataset.
- The fact that the class of hash functions is universal is enough.
- This gives the expected length of a list to be at most $1 + \alpha$.

A good class of universal hash functions

- Pick a prime number p such that every key is in the range $0, \ldots, p-1$ pick $0 \leq a \leq p-1$ and $1 \leq b \leq p-1$.
- Define

  $$h_{a,b} = ((ak + b) \mod p) \mod n$$

  (n = number of slots).
- The set

  $$\mathcal{H} = \{h_{a,b} | 0 \leq a \leq p-1, 1 \leq b \leq p-1\}$$

  is a class of universal hash functions.
- Added bonus n need not be prime.

Open Hashing

- Implementing chaining requires lots of pointer.
- Deleting an element from a list can be costly.
- Open Hashing uses the table to store values. We hop around the table until we find the element in the table or hit an unused slot.
- All slots are marked with `empty`,`used` (or `deleted`).

Open Hashing – Hoping around

- Given a table of size n define a sequence of hash functions

$$h_0, h_1, \ldots, h_{n-1}$$

- Often h_{i+1} has a simple relationship to h_i.
- Instering a key k, if h_o is not empty then insert in h_1, if h_1 is not empty then insert in h_2 and so on until h_{n-1}.
- Searching for a key k, if we don't find it at h_1 try h_2 and so on, if we get to h_{n-1} or an empty slot then the element is not in the hash table.
- Deleting is more difficult, since we have to distinguish between empty and deleted slots, then the search has to be modified to hop over deleted slots.

Linear Probing

- Given a hash function h' define

$$h_i(k) = (h'(k) + i) \mod n$$

- Which means that $h_0(k) = h'(k)$, $h_1(k) = h'(k)+1$, $h_2(k) = h'(k)+2$ and so on when we get the end of the table we wrap around to the beginning.

Analysis of Open Hashing

- If the load factor is less than 1 (otherwise the hash table is full) the expected number of probes in an unsucessful search is at most

$$\frac{1}{1-\alpha}$$

- For example if the hash table is 90% full then the average number of probles is at most $1/(1-0.9) = 10$.

Open hashing versus chaining

- Open hashing, less memory overhead than chaining.
- If the load factor is small consider open hashing
- Chaining, slightly more overhead
- But chaining degrades well with increased load factor.

In reality it doesn't really matter (unless you are tight for memory) what you use. Any kind of hash function as long as the lookup overhead is small works well in practice.