Nov 2: MIdterm

Nov 4th Hash Tables:
Hash takes in a bunch of stuff and outputs something small
What do we want to make, we want a dictionary.

- Search:
- Insert:
- Delete:

```
search(k)
    return T[k]
insert(x)
    T[x->key] = x
delete(x)
    T[x->key] = NULL
```

K is in U
where U is all keys
what do we want U to be? ints: fuck no way too many,

h; U(1) -> some num {0,...,m-1}
|u| cardinality how many things are in the set. if |U| = m and 1:1 onto, we have a perfect hash function. In practice you don't do this, because you won't know all the K's ahead of time.

Usually U is very large, and K is much smaller.
|u|/|T| is the number of things that'll fall into each slot.
collisions h(k1) = h(k2)
SHA-1(text) -> 160 bits
take in 10^6
The adversary is someone who can engineer a collision.

|T|<<|U|
usually
|T| << |K|
Gonna have collisions all over the place. potential collisions is |K|/m
Hash collisions resolved through chaining, (linked lists!)

Exact order: (2+å/2 - å/2n) alpha is how full the hash table is
n≈m
n=O(m) some finite constant times m
å = n/m = O(1)

While there is a finite constant times more inserts than spots then on average the runtime is still O(1). Hash's are only order 1 when they are dominant when the linked lists begin to dominate then it becomes order O(n).
O((n/c)^2) is O(n^2) unless n/c is small say 2 or 5 in which case it's still kinda O(1).
Move to front Rule, helps out the linked list part of the hash.
You can also replace the linked list with a self balancing binary tree. In which case worstcase is still O(nlogn)

I FINALLY GET TO LEARN HOW TO MAKE A HASH FUNCTION YAYAYAYAY!!!!!

K is a member of set ∑20 keys include " " a,b,c...z,aa,ab….. 20zs"
∑ = ascii 0-127
h(K) -> Natural numbers
K e {0,1}
K = "sam"
n = |S| + 128x|a| + 128^2x|m|
This is the only way to get this number in base 128.
This is a BDN a big darn number
h(k) = n mod m

oh btw m should be prime. otherwise things become multiples of another.
You also don't want m associated with a pattern in the key distribution.

The problem with this is that it's subject to clustering.

Sequence of primes. Look for his paper.

h(k) = floor(m(KAmod1))
O<A<1
This will give you a number between 0 and m-1
A = 1 - the golden ratio 1- ((sqrt(5)+1) /2)

The drawback is you have to use floating point arithmetic, it's a lot faster than it used to be but it's still a lil difficult and annoying if you fuck up. Especially because K and A are BDN

Uniform distribution everything is equally likely, a unloaded dice. You want this for h. The multiplication method does a better job of this than division because golden ratio.

Universal hashing: h1,…. ,hj, whenever I startup the computer I set r = 1-j so he won't know j so he won't know h so they won't be able to fuck with us at least for a while.

```
insert(k)
    i = 0
    repeat
    y = h(k,i)
    if T[i] = Null
        T[i] = k
    else
        i++
```

h: u x {0,1,...m-1} -> {0,1,...m-1}
m! permutations
h is a permutation of m
h1(k) -> hm(k) will hit every slot in the table if they are unique permutations.

h(k,i) = (h'(k)+i)modn (h' is original hash function)
h(k,0) = h'
h(k,1) = h'(k) +1 modn
linear probing

h(k,i) = h'(k) + c1*i + c2 * i^2
if m = 2^n c1=c2=.5 works for free.
if m is prime lots of stuff works

h1(K)+ih2(k)
h2(k) must be relatively prime to m, you get this for free if m is prime, or if m = 2^n make h2 return only odd numbers.

Crypto hashes, it's an art. What you want is something that is hard to invert, something that is hard to figure out.
In crypto you ask: who is your adversary.