COP3502 - 11/8/23

(0) P4 so people submitted same file. Called qsort. Don't do that!

(1) Videos on Line - Optional (Helpful)

(2) Hash Tables!

insert
delete
search

Can we do better than $O(\log n)$?

On avg, yes!

hash function

input: any size
output: fixed size (a number from 0 to n-1)
multiple inputs may map to the same output.
goal w/good hash function: if $x \neq y$, the probability $H(x) = H(y)$ is $\approx \frac{1}{n}$.

example of bad hash function

$f("cat") = 2 + 0 + 19 = 21$ o/o n ← So keeps in range.

Sum of letter values } Oops!

$f("act") = 21$

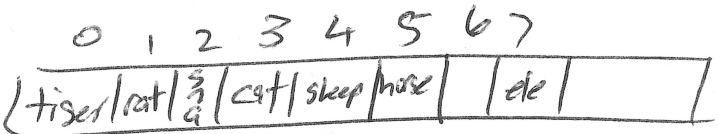
How to deal w/ collision

1) Don't - erase the old thing!
(LOSSY) but quick.

2) linear probing - if slot i is full, goto slot i+1, i+2, ... n-1, 0, 1, 2 ....
keep looking

3) quadratic probing

4) separate chaining hashing



insert(cat) = 3
insert(ele) = 7
insert(tiger) = 0
insert(sheep) = 3
insert(snake) = 2
insert(rat) = 1
insert(horse) = 0
search(cheetah) = 0 → go to index 6 (1st empty index)

Insert works BUT if lots slot are taken insert → O(n)

clumping

Search(& horse)

→ AMEND, if sequence of deletes just have to go through the whole thing!

Quadratic Probing

n = PRIME

i, i+1, i+4, i+9, i+16, ... mod n] reduces clumping clustering

① ③ ⑤ ⑦

In code

x += 1
x += 3
x += 5

} if I modify where I'm looking I don't add a perfect SQ

Reasonable

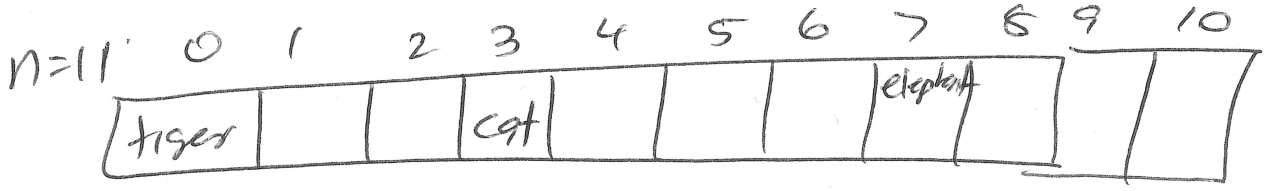
```
int f(char* word, int n) {
    int res = 0, len = strlen(word);
    for (int i = 0; i<len; i++)
        res = (26*res + word[i] - 'a') % n;
    return res;
}
```

cat = $2\times26^2 + 0\times26^1 + 19\times26^0$

↳ treats string as base 26.

We assume we have access to a decent hash function w/ the desired characteristics.

Hash Table will be an array of size n.



insert(x) → 1) calculate H(x) → valid index
2) put x in index H(x)

H("cat") = 3

Search(x) → 1) calculate H(x)
2) look in array @ index H(x).

H("elephant") = 7

H("tiger") = 0

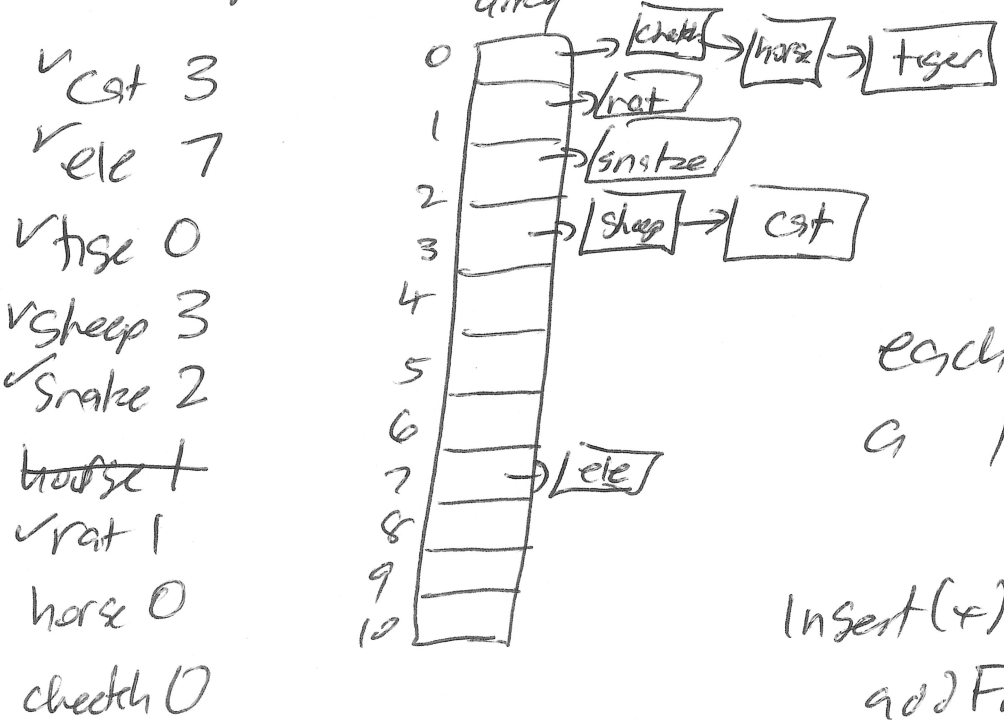
Fine - UNTIL a second item maps to a previously used array index.

COLLISION
$x \neq y$
but $H(x) = H(y)$

* If delete not allowed, search is easier.

LAST SOL

separate chaining hashing



each array slot IS a linked list

```
Insert(x)
    addFront(array[H(x)], x)
```

```
Search(x)
    return search(array[H(x)], k)
```

O(length longest list)

↳ ~ lg lg n - really good!

make table as big as # of elements you expect.