COP 3502: 11/12/21

Hash Tables

- inserts
- deletes
- searches

} faster $O(\lg n)$

Goal: Average case $O(1)$ for each
not worry too much about worst case.
(Impossible to make an input force worst case.)

Hash function

Input: Anything (arbitrary string)
Output: fixed integer in some range
0 to n-1, for some given n.
many-to-one (multiple inputs may map to the same output.)

Properties of a good hash function

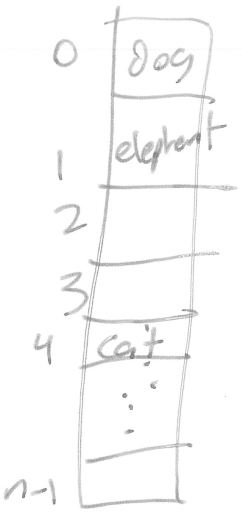
1) each output value is equally likely.

2) Given some output value it should be difficult to construct an input that "creates" that value. [Given y, it's hard to find x s.t. $f(x) = y$.]

3) If we make small changes to x, then (say x and x'), there should many changes btw $f(x)$ and $f(x')$.

How to create a hash table w/a hash function?

Array of size n



1. Insert ("cat")

Calculate f("cat") = 4

2. Place cat in index 4.

f("elephant") = 1

f("dog") = 0

Problem?

What if f("giraffe") = 4 but cat's already there?

COLLISION

4 methods of dealing w/ collisions

1) Overwrite data, lose the old data.
   - ADVANTAGE (EASY)
   - DISADVANTAGE (LOSE STUFF)

2) Linear Probing
   if a slot, k, is full when you try to insert go to slot (k+1) & on and see if it's empty. Continue until you find the first empty slot.
   - ADVANTAGE (DON'T LOSE STUFF)
   - DISADVANTAGE (FINDING ISN'T AS FAST)

When looking for my giraffe, I can't stop looking after index 4 until I find an empty slot, so ~~they~~ this could slow me down a lot.

- Clustering

As there are more collisions, the probability of hitting a cluster increases and linear probing ensures those clusters will grow, if hit.

3) When looking for new places go to these indexes:

k
$(k+1) \bmod p$
$(k+4) \bmod p$
$(k+9) \bmod p$
$(k+16) \bmod p$
$\vdots$
$(k+i^2) \bmod p$
$\vdots$
$\left(k+\left(\frac{p-1}{2}\right)^2\right) \bmod p$

Quadratic Probing

(a) table size prime
(b) less than half full

Prove each # of this list is unique.

Assume the opposite that

$k+i^2 \equiv k+j^2 \pmod p$
$\Rightarrow (i \neq j,\ 0 < i,j < \frac{p}{2})$
$i^2-j^2 \equiv 0 \pmod p$
$(i-j)(i+j) \equiv 0 \pmod p$
$\Rightarrow p \mid [(i-j)(i+j)]$

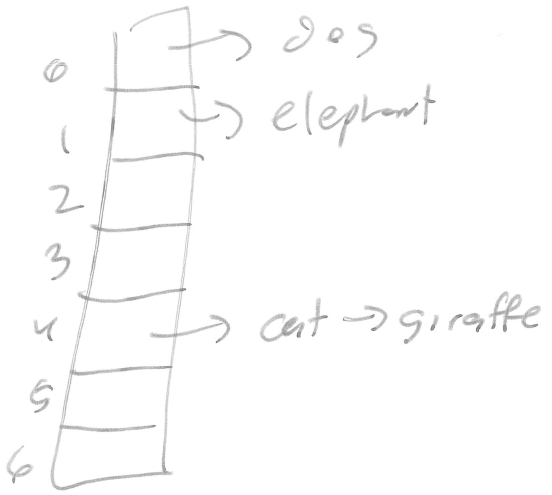
Since p is prime, $p \mid (i-j)$ or $p \mid (i+j)$

$p \mid (i-j)$: FALSE because $i \neq j$ & $|i-j| < \frac{p}{2}$

$p \mid (i+j)$: FALSE $0 < i+j < p$.

For both linear + quadratic probing, double table size when it gets half full

4) Separate Chaining Hashing
At end index have a linked list



f(cat) = 4
f(dog) = 0
f(elephant) = 1
f(giraffe) = 4

Idea is runtimes are ~~probability~~ based on the probability of searching in each slot. Longer ~~lists~~ lists ⇒ worse run times.

Why would someone use Quadratic Probing over this? (Index math in an array is faster then mallocing + ~~too~~ following links in lists.)

Hash Functions
$f("cat") = ('c' + 'a' + 't') \%n$ bad, many values map to the same spot + for most words sum of ascii values clusters in the 200-1500 range.

Improvement $f("cat") = ('c' \times 128^2 + 'a' \times 128^1 + 't' \times 128^0) \%n$
writing a string in some "base"

$$p(\text{search List}) = \frac{\#\text{ words in list}}{\text{Total Words}}$$

$$\text{Avg Run Time (List)} = \frac{\text{list len} + 1}{2}$$

$$\sum_{\text{lists}} \frac{\#\text{ word is list}}{TW} \times \frac{(\text{list len} + 1)}{2}$$

$$\frac{\sum_{i=0}^{100} \text{freq}[i] \times i \times (i+1)/2}{TW}$$

- $\text{freq}[i]$: num lists length i
- $i \times (i+1)/2$: ans for each list of length i
- TW