3/18/23 Hashing

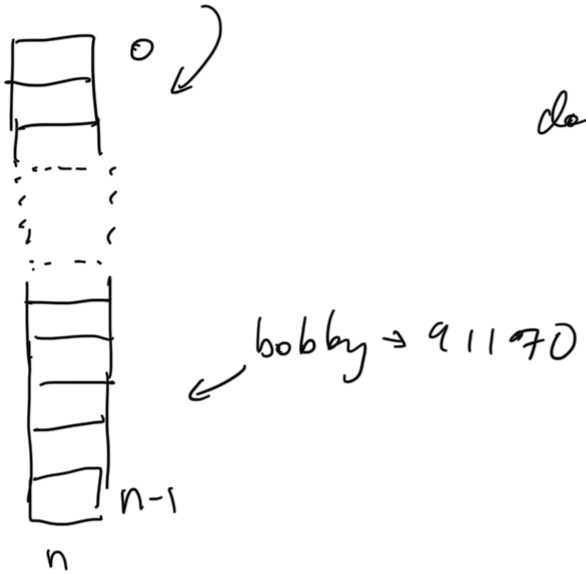
take a "key" (string), make # from it # should be uniformly distributed
- uniformly distributed: equal possibilities, equiprobable

find records by key aka a dictionary:
- regular array searches are $\Theta(n)$ +

a direct address table has its limitations

Trading space for time w/ Hash tables
- use a sparse table, "scatter storage"
- randomizing
O(1) avg search time (kinda)

ex) "bob" → 7



don't want to collide

what happens on collision?
- collision resolution
- direct probe or use other data struct to resolve

need hash funct to generate indicies

- fast
- uniform
- use all input data

Types of hash fncts
- division based
- mult based
- string key - compute # from string

Load factor
- goes to zero, collisions goes down

~23 people 50-50 chance that some pair shares a birthday

Inevitable a collision will happen if # of keys > than # of hash values
- make table big enough for purposes

Linear probing - keep looking for next new spot if collision

Quadratic probing - look for place $(n+1)^2$
- hash table must be prime size
- table must never be more than half full

Double hash functions

can use linked list to do hashing & link any collision to same key

Hashing not good for caching, very slow
- hard to find shit
- hard to delete

Bloom Filter
- data structure meant for checking if an element is present
- only consists of 1s & 0s
- paired w/ data structures including hash tables & linked lists etc
- if find 0, we know element doesn't exist
- no delete
- should have multiple hash functions to reduce collisions & false positives

Cryptographic hash functions
- Message Integrity & Digital signature
- hash document
- hash altered doc
- compare, different hashes
  - can use private key to encrypt
  - public key can verify signature

- deterministic, should get same hash

SHA256
SHA512
SHA3 - sponge construction
- based

Speck
- encrypts & hashes