Hashing

Dictionaries

- Operations.
  - makeset, insert, delete, find

Model

- keys are integers in $M = \{1, \ldots, m\}$
- (so assume machine word size, or “unit time,” is $\log m$)
- can store in array of size $M$
- using power: arithmetic, indirect addressing
- compare to comparison and pointer based sorting, binary trees
- problem: space.

Hashing:

- find function $h$ mapping $M$ into table of size $n \ll m$
- Note some items get mapped to same place: “collision”
- use linked list etc.
- search, insert cost equals size of linked list
- goal: keep linked lists small: few collisions

Hash families:

- problem: for any hash function, some bad input (if $n$ items, then $m/n$ items to same bucket)
- This true even if hash is e.g. SHA1
- Solution: build family of functions, choose one that works well

Set of all functions?

- Idea: choose “function” that stores items in sorted order without collisions
- problem: to evaluate function, must examine all data
- evaluation time $\Omega(\log n)$. 

“description size” \( \Omega(n \log m) \),

- Better goal: choose function that can be evaluated in constant time without looking at data (except query key)

How about a random function?

- set \( S \) of \( s \) items
- If \( s = n \), balls in bins
  - \( O((\log n)/(\log \log n)) \) collisions w.h.p.
  - And matches that somewhere
  - but we care more about average collisions over many operations
  - \( C_{ij} = 1 \) if \( i, j \) collide
  - Time to find \( i \) is \( \sum_j C_{ij} \)
  - expected value \( (n - 1)/n \leq 1 \)

- more generally expected search time for item (present or not): \( O(s/n) = O(1) \) if \( s = n \)

Problem:

- \( n^m \) functions (specify one of \( n \) places for each of \( n \) items)
  - too much space to specify \( (m \log n) \),
  - hard to evaluate
- for \( O(1) \) search time, need to identify function in \( O(1) \) time.
  - so function description must fit in \( O(1) \) machine words
    - Assuming \( \log m \) bit words
    - So, fixed number of cells can only distinguish \( \text{poly}(m) \) functions

- This bounds size of hash family we can choose from

Our analysis:

- sloppier constants
- but more intuitive than book

2-universal family: [Carter-Wegman]

- Key insight: don’t need entirely random function
- All we care about is which pairs of items collide
- so: OK if items land pairwise independent
• pick $p$ in range $m, \ldots, 2m$ (not random)
• pick random $a, b$
• map $x$ to $(ax + b \mod p) \mod n$
  – pairwise independent, uniform before $\mod n$
  – So pairwise independent, near-uniform after $\mod n$
  – at most 2 “uniform buckets” to same place
• argument above holds: $O(1)$ expected search time.
• represent with two $O(\log m)$-bit integers: hash family of poly size.
• max load may be large is $\sqrt{n}$, but who cares?
  – expected load in a bin is 1
  – so $O(\sqrt{n})$ with prob. 1-1/n (chebyshev).
  – this bounds expected max-load
  – some item may have bad load, but unlikely to be the requested one
  – can show the max load is probably achieved for some 2-universal families

perfect hash families

Ideally, would hash with no collisions

• Explore case of fixed set of $n$ items (read only)
• perfect hash function: no collisions
• Even fully random function of $n$ to $n$ has collisions

Alternative try: use more space:

• How big can $s$ be for random $s$ to $n$ without collisions?
  – Expected number of collisions is $E[\sum C_{ij}] = \binom{s}{2}(1/n) \approx s^2/2n$
  – Markov Inequality: $s = \sqrt{n}$ works with prob. 1/2
  – Nonzero probability, so, 2-universal hashes can work in quadratic space.
• Is this best possible?
  – Birthday problem: $(1 - 1/n) \cdots (1 - s/n) \approx e^{-(1/n + 2/n + \cdots + s/n)} \approx e^{-s^2/2n}$
  – So, when $s = \sqrt{n}$ has $\Omega(1)$ chance of collision
  – 23 for birthdays
  – even for fully independent
Finding one

- We know one exists—how find it?
- Try till succeed
- Each time, succeed with probability 1/2
- Expected number of tries to succeed is 2
- Probability need $k$ tries is $2^{-k}$

Two level hashing for linear space

- Hash $s$ items into $O(s)$ space 2-universally
- Build quadratic size hash table on contents of each bucket
- bound $\sum b_k^2 = \sum_k (\sum_i [i \in b_k])^2 = \sum C_i + C_{ij}$
- expected value $O(s)$.
- So try till get (markov)
- Then build collision-free quadratic tables inside
- Try till get
- Polynomial time in $s$, Las-vegas algorithm
- Easy: $6s$ cells
- Hard: $s + o(s)$ cells (bit fiddling)

Define las vegas, compare to monte carlo.

Derandomization

- Probability 1/2 top-level function works
- Only $m^2$ top-level functions
- Try them all!
- Polynomial in $m$ (not $n$), deterministic algorithm