Hashing

Review:

- Goal: map s items from size m universe to table of size n
- some items get mapped to same place: "collision"
- Problem: any function has bad set mapping m/n items to same bucket
- Solution: build family of functions, choose one that works well

Hash Families:

- Random function has good behavior, but hard to compute efficiently
- Goal: $O(1)$ access time
- So can only look at constant number of cells.
- Each holds value in range $1, \ldots, m$ ($\log m$ bits)
- So, fixed number of cells can only distinguish poly(m) functions
- This bounds size of hash family we can choose from

Recall random function analysis:

- set S of s items
  - what is expected time for i access?
  - $C_{ij} = 1$ if i, j collide
  - Time to find i is $\sum_j C_{ij}$
  - expected value $(s-1)/n \leq 1$ for $s \leq n$ (and optimal for s)

2-universal family:

- how much independence was used above? pairwise (search item versus each other item)
- so: OK if items land *pairwise independent*
- pick p in range $m, \ldots, 2m$ (not random)
- pick random a, b
- map x to $(ax + b \bmod p) \bmod n$
  - pairwise independent, uniform before $\bmod m$
  - So pairwise independent, near-uniform after $\bmod m$
- argument above holds: $O(1)$ expected search time.

- represent with two $O(\log m)$-bit integers: hash family of poly size.
- em max load?
  - 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

perfect hash families

- perfect hash function: no collisions
- for any S of $s \leq n$, perfect h in family
- eg, set of all functions
- but hash choice in table: $m^{O(1)}$ size family.
- exists iff $m = 2^{\Omega(n)}$ (probabilistic method) (hard computationally)
  - random function. Pr(perfect)$= n!/n^n$
  - So take $n^n/n! \approx e^n$ functions. Pr(all bad)$= 1/e$
  - Number of subsets: at most m^n
  - So take $e^n \cdot \ln m^n = ne^n \ln m$ functions. Pr(all bad)$\leq 1/m^n$
  - So with nonzero probability, no set has all bad functions (union)
  - number of functions: $ne^n \ln m = m^{O(1)}$ if $m = 2^{\Omega(n)}$
- Too bad: only fit sets of $\log m$ items
- also, hard computationally

Alternative try: use more space:

- How big can s be for random s to n without collisions?
  - Expected number of collisions is $E[\sum C_{ij}] = \binom{s}{2}(1/n) \approx s^2/2n$
  - So $s = \sqrt{n}$ works with prob. 1/2
- Is this best possible?
  - Birthday problem: $(1 - 1/n) \cdots (1 - s/n) \approx e^{-s^2/2n}$
  - So, when $s = \sqrt{n}$ has $\Omega(1)$ chance of collision
  - 23 for birthdays

Two level hashing solves problem

- Hash s items into $O(s)$ space
- Build quadratic size hash table on contents of each bucket
- bound $\sum b_k^2 = \sum_i [i \in b_k] = \sum C_i + C_{ij} = O(s)$
- expected $O(s)$.
- So try till get
- 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)

Derandomization

- Probability 1/2 top-level function works
- Only m^2 top-level functions
- Try them all!
- Polynomial in m, deterministic algorithm

Treaps

Dictionaries for **ordered** sets

- New Operations.
  - enumerate in order
  - successor-of, predecessor-of (even if not in set)
  - join(S, k, T), split, paste(S, T)

Binary tree.

- child and parent pointers
- endogenous: leaf nodes empty.
- *balanced* if depth $O(\log n)$
- average case.
- worst case

Tree balancing

- rotations
- implementing operations.
- red/black, AVL
- splay trees.
  - drawbacks in geometry:
  - auxiliary structure on nodes in subtree
  - rebuild on rotation

Returning to average case:

- Assign random "arrival orders" to keys
- Build tree **as if** arrived in that order
- Average case applies
- No rotations on searches

Choosing priorities

- define arrival by random priorities
- assume continuous distribution, fix.
- eg, use $2 \log n$ bits, w.h.p. no collisions

Treaps.

- tree has keys in heap order of priorities
- unique tree given priorities—follows from insertion order
- implement insert/delete etc.
- rotations to maintain heap property

Depth $d(x)$ analysis

- Tree is trace of a quicksort
- We proved $O(\log n)$ w.h.p.
- for x rank k, $E[d(x)] = H_k + H_{n-k+1} - 1$
- $S^- = \{y \in S \mid y \leq x\}$
- Q_x = ancestors of x

- Show $E[Q_x^-] = H_k$.
- to show: $y \in Q_x^-$ iff inserted before all z, $y < z \leq x$.
- deduce: item j away has prob $1/j$. Add.
- Suppose $y \in Q_x^-$.
  - The inserted before x
  - Suppose some z between inserted before y
  - Then y in left subtree of z, x in right, so not ancestor
  - Thus, y before every z
- Suppose y first
  - then x follows y on all comparisons (no z splits
  - So ends up in subtree of y

Rotation analysis

- Insert/Delete time
  - define spines
  - equal left spine of right sub plus right spine of left sub
  - proof: when rotate up, on spine increments, other stays fixed.
- R_x length of right spine of left subtree
- $E[R_x] = 1 - 1/k$ if rank k
- To show: $y \in R_x$ iff
  - inserted after x
  - all z, $y < z < x$, arrive after y.
  - if z before y, then y goes left, so not on spine
- deduce: if r elts between, $r!$ of $(r+2)!$ permutations work.
- So probability $1/r^2$.
- Expectation $\sum 1/(1 \cdot 2) + 1/(2 \cdot 3) + \cdots = 1 - 1/k$
- subtle: do analysis only on elements inserted in real-time before x, but now assume they arrive in random order in virtual priorities.