Admin

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)
- 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 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]

- 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 $(ax + b \mod p) \mod n$
  – pairwise independent, uniform before $\mod m$
  – So pairwise independent, near-uniform after $\mod m$
  – 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.

• $\text{max}$ 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
  – can show the max load is probably achieved for some 2-universal families

**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. $\Pr(\text{perfect}) = n!/n^n$
  – So take $n^n/n! \approx e^n$ functions. $\Pr(\text{all bad}) = 1/e$
  – Number of subsets: at most $m^n$
  – So take $e^n \cdot \ln m^n = ne^n \ln m$ functions. $\Pr(\text{all bad}) \leq 1/m^n$
  – 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

• note one word contains $n$-bits—one per item

• 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 (markov)
  - 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

Two level hashing solves problem
  - 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)

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

Fingerprinting
Basic idea: compare two things from a big universe \( U \)
  - generally takes \( \log U \), could be huge.
  - Better: randomly map \( U \) to smaller \( V \), compare elements of \( V \).
  - Probability(same) = 1/|V|
• intuition: \( \log V \) bits to compare, error prob. \( 1/|V| \)

We work with \textit{fields}

• add, subtract, mult, divide
• 0 and 1 elements
• eg reals, rats, (not ints)
• talk about \( \mathbb{Z}_p \)
• which field often won’t matter.

Verifying matrix multiplications:

• Claim \( AB = C \)
• check by mul: \( n^3 \), or \( n^{2.376} \) with deep math
• Freivald’s \( O(n^2) \).
• Good to apply at end of complex algorithm (check answer)

Freivald’s technique:

• choose random \( r \in \{0, 1\}^n \)
• check \( ABr = Cr \)
• time \( O(n^2) \)
• if \( AB = C \), fine.
• What if \( AB \neq C \)?
  – trouble if \( (AB - C)r = 0 \) but \( D = AB - C \neq 0 \)
  – find some nonzero row \( (d_1, \ldots, d_n) \)
  – \( \text{wlog } d_1 \neq 0 \)
  – trouble if \( \sum d_ir_i = 0 \)
  – \( \text{ie } r_1 = (\sum_{i>1} d_ir_i)/d_1 \)
  – principle of deferred decisions: choose all \( i \geq 2 \) first
  – then have exactly one error value for \( r_1 \)
  – prob. pick it is at most \( 1/2 \)

How improve detection prob?

– \( k \) trials makes \( 1/2^k \) failure.
– Or choosing \( r \in [1, s] \) makes \( 1/s \).
• Doesn’t just do matrix mul.
  – check any matrix identity claim
  – useful when matrices are “implicit” (e.g. $AB$)
• We are mapping matrices ($n^2$ entries) to vectors ($n$ entries).

**String matching**

Checksums:
• Alice and Bob have bit strings of length $n$
• Think of $n$ bit integers $a$, $b$
• take a prime number $p$, compare $a \mod p$ and $b \mod p$ with $\log p$ bits.
• trouble if $a = b \pmod p$. How avoid? How likely?
  – $c = a - b$ is $n$-bit integer.
  – so at most $n$ prime factors.
  – How many prime factors less than $k$? $\Theta(k/\ln k)$
  – so take $2n^2 \log n$ limit
  – number of primes about $n^2$
  – So on random one, $1/n$ error prob.
  – $O(\log n)$ bits to send.
  – implement by add/sub, no mul or div!

How find prime?
  – Well, a randomly chosen number is prime with prob. $1/\ln n$,
  – so just try a few.
  – How know its prime? Simple randomized test (later)

Pattern matching in strings
• $m$-bit pattern
• $n$-bit string
• work mod prime $p$ of size at most $t$
• prob. error at particular point most $m/(t/\log t)$
• so pick big $t$, union bound
• implement by add/sub, no mul or div!
Fingerprints by Polynomials

Good for fingerprinting “composable” data objects.

• check if $P(x)Q(x) = R(x)$
• $P$ and $Q$ of degree $n$ (means $R$ of degree at most $2n$)
• mult in $O(n \log n)$ using FFT
• evaluation at fixed point in $O(n)$ time
• Random test:
  – $S \subseteq F$
  – pick random $r \in S$
  – evaluate $P(r)Q(r) - R(r)$
  – suppose this poly not 0
  – then degree $2n$, so at most $2n$ roots
  – thus, prob (discover nonroot) $|S|/2n$
  – so, eg, sufficient to pick random int in $[0, 4n]$
  – Note: no prime needed (but needed for $\mathbb{Z}_p$ sometimes)

• Again, major benefit if polynomial implicitly specified.

String checksum:

• treat as degree $n$ polynomial
• eval a random $O(\log n)$ bit input,
• prob. get 0 small

Multivariate:

• $n$ variables
• degree of term: sum of vars degrees
• total degree $d$: max degree of term.
• Schwartz-Zippel: fix $S \subseteq F$ and let each $r_i$ random in $S$
  \[
  \Pr[Q(r_i) = 0 \mid Q \neq 0] \leq d/|S|
  \]
  Note: no dependence on number of vars!

Proof:
• induction. Base done.

• $Q \neq 0$. So pick some (say) $x_1$ that affects $Q$

• write $Q = \sum_{i \leq k} x_i^i Q_i(x_2, \ldots, x_n)$ with $Q_k() \neq 0$ by choice of $k$

• $Q_k$ has total degree at most $d - k$

• By induction, prob $Q_k$ evals to 0 is at most $(d - k)/|S|$

• suppose it didn’t. Then $q(x) = \sum x_i^i Q(r_2, \ldots, r_n)$ is a nonzero univariate poly.

• by base, prob. eval to 0 is $k/|S|$

• add: get $d/|S|$

• why can we add?

$$\Pr[E_1] = \Pr[E_1 \cap \overline{E_2}] + \Pr[E_1 \cap E_2] \leq \Pr[E_1 | \overline{E_2}] + \Pr[E_2]$$

Small problem:

• degree $n$ poly can generate huge values from small inputs.

• Solution 1:
  – If poly is over $Z_p$, can do all math mod $p$
  – Need $p$ exceeding coefficients, degree
  – $p$ need not be random

• Solution 2:
  – Work in $Z$
  – but all computation mod random $q$ (as in string matching)

**Perfect matching**

• Define

• Edmonds matrix: variable $x_{ij}$ if edge $(u_i, v_j)$

• determinant nonzero if PM

• poly nonzero *symbolically*.
  – so apply Schwartz-Zippel
  – Degree is $n$
  – So number $r \in (1, \ldots, n^2)$ yields 0 with prob. $1/n$
Det may be huge!

- We picked random input $r$, knew evaled to nonzero but maybe huge number
- How big? About $n^r$,
- So only $O(n \log n + n \log r)$ prime divisors
- (or, a string of that many bits)
- So compute mod $p$, where $p$ is $O((n \log n + n \log r)^2)$
- only need $O(\log n + \log \log r)$ bits