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 $Z_p$ 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:

- $Q \neq 0$. So pick some (say) $x_1$ that affects $Q$
- write $Q = \sum_{i \leq k} x_i Q_i(x_2, \ldots, x_n)$ 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 $q(x) = \sum x_i Q(r_2, \ldots, r_n)$ 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 $\mathbb{Z}_p$, can do all math mod $p$
  - Need $p$ exceeding coefficients, degree
  - $p$ need not be random
- Solution 2:
  - Work in $\mathbb{Z}$
  - 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 \( n!r^n \),
- 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

**Hashing**

We’ve been looking at collisions for a pair, now collisions for a group.

**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 \( n \ll m \)
- hash function is fingerprint.
- 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

Our analysis:

• sloppier constants
• but more intuitive than book

Hash families:

• problem: for any hash function, some bad input
• solution: choose randomly from a hash family

First family: all functions

• 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:

• too much space \( (m \log n) \), hard to evaluate

• note: 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 \( m^{O(1)} \) functions.

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 \( m, \ldots, 2m \)

• 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 $\mod m$

• 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