Comparing strings

- or generally, a big number $x < 2^n$:
- generate a *fingerprint* (like a hash, but not):

  $$ H_p(x) = x \mod p $$

  - where p is a prime from $1 \ldots m$
- this time $co\,\mbox{-}RP$:
  - if $x = y$, then $H_p(x) = H_p(y) (\forall p)$
  - if $x = y$, then sometimes $H_p(x) = H_p(y)$, when:
    - when $(x \mod p) = (y \mod p)$
    - that is p divides $x-y$
- number theory:
  - $\pi(n)$ = number of primes $< n$
    - $\pi(n) \sim \frac{n}{\ln n}$
  - number of prime divisors of $x < 2^n$
    - $< \pi(n)$
- and back:
  - $x, y < 2^n \Rightarrow x-y < 2^n$
  - so then for $< \pi(n)$ primes, the test will fail
  - so with $m := n^2$:

  $$ P[fail] = \frac{\pi(n)}{\pi(n^2)} = \frac{n}{\ln n} \cdot \frac{\ln n^2}{n^2} = \frac{2}{n}$$

Searching strings

- **given**:
  - the text – a string T of length n
  - the pattern – a string P of length m
  - obviously $m < n$
- **task**:
  - find all occurrences of P in T
  - or first occurrence
  - or if one exists
- naive:
  - check every possible starting position
    - $n-m$ positions
    - at worst m steps
  - run time: $O((n-m)\cdot m) = O(nm)$
  - space: $O(1)$
- randomized – Rabin-Karp:
  - like naive, but:
    - compute hash for current window
    - check hash against pattern hash
      - *then* check match
  - **rolling hash**
  - worst case: $O(nm)$
  - expected: $O(n+m)$
  - analysis: via fingerprinting
  - randomized (Las Vegas)
    - or probabilistic (Monte Carlo)?
- deterministic – Knuth-Morris-Pratt:
  1. pre-compute prefixes in P
  2. jump less on a failure
  - example:
    - find: $abcabd$
    - in: $(abc)^nabd$
  - run time: $O(n+m)$
  - space: $O(m)$
- Boyer-Moore:

    1. pre-compute …something in P
    2. skip checking …some alignments
    - run time: $O(n+m)$
    - space: $O(m)$

Multiple patterns

- Rabin-Karp still works:
    - just check if hash in in a set of hashes
    - expected run time: $O(\sum m_i + n + o)$
        - o – # of occurrences
    - space: $O(\sum m_i)$

- deterministic – Aho-Corasick:
    - run time: $O(\sum m_i + n + o)$
        - o – # of occurrences
    - space: $O(\sum m_i)$

Polynomial identity testing

- **given**:
    - polynomial $P(x_1, x_2, \ldots, x_n)$
    - polynomial $Q(x_1, x_2, \ldots, x_n)$
    - implicitly, not explicitly
    - (as a black box, not list of coefficients)
- **task**:
    - decide whether $P \equiv Q$

- method:
    - evaluate for random x_i
    - $co\,\mbox{-}RP$:
        - if equivalent, it will be the same
        - if not, it *mostly* won't
    - why?
        - $P - Q$ is also a polynomial
    - chance of $P(\bar x) = 0$ if $P\not\equiv 0$:
        - $P[fail] \leq \frac{\deg(P)}{|S|}$, choosing $x_i \in S$
        - $P[fail] < \tfrac{1}{2}$ for $S = \{0,\, \pm1,\, \ldots,\, \pm d\}$

- example: Vandermonde identity:

  $$ A := ((x_i^{j-1}))_{n \times n} $$

  $$ det(A) = \prod_{i<j} (x_j - x_i) $$

    - symbolic:
        - determinant – $O(2^n)$ terms
        - product – $O(2^n)$ terms
    - numeric:
        - determinant – $O(n^3)$ time
        - product – $O(n^2)$ time

- example: verifying matrix multiplication:
    - claim: $A \times B = C$, ($n\times n$ matrices)
    - equiv: $(\forall x) A B x = C x$
        - associativity: $(A B) x = A (B x)$
        - polynomials evaluable in $O(n^2)$
    - $P[fail] < \frac{1}{2}$
    - runtime: $O(n^2)$

Bonus: matrix multiplication

- multiply two $2n$-bit numbers:
    - divide and conquer:
        - $X \times Y = (A \cdot 2^n + B) \times (C \cdot 2^n + D)$
    - four multiplications of two n numbers (+adds):
        - $T(2n) = 4T(n) + O(n)$
        - $T(n) = O(n^2)$, nothing gained
    - actually three muls suffice (+adds&subs):

    - $T(2n) = 3T(n) + O(n)$
    - $T(n) = O(n^{\log_2 3})$
  - Karatsuba’s algorithm

- for matrices, the same idea:
  - $T(2n) = 7T(n) + O(n^2)$
  - $T(n) = O(n^{\log_2 7})$
  - Strassen’s algorithm

Admin

- midterm: 21. 4. 19:00 in A