String Matching

Thanks to
Piotr Indyk

String Matching

• Input: Two strings \( T[1\ldots n] \) and \( P[1\ldots m] \), containing symbols from alphabet \( \Sigma \)
• Goal: find all “shifts” \( 1 \leq s \leq n-m \) such that \( T[s+1\ldots s+m] = P \)
• Example:
  – \( \Sigma = \{a, b, \ldots, z\} \)
  – \( T[1\ldots 18] = \text{“to be or not to be”} \)
  – \( P[1\ldots 2] = \text{“be”} \)
  – Shifts: 3, 16

Simple Algorithm

\[
\text{for } s \leftarrow 0 \text{ to } n-m \\
\text{Match} \leftarrow 1 \\
\text{for } j \leftarrow 1 \text{ to } m \\
\quad \text{if } T[s+j] \neq P[j] \text{ then} \\
\quad \quad \text{Match} \leftarrow 0 \\
\quad \text{exit loop} \\
\text{if } \text{Match} = 1 \text{ then output } s
\]

Results

• Running time of the simple algorithm:
  – Worst-case: \( O(nm) \)
  – Average-case (random text): \( O(n) \)
• Is it possible to achieve \( O(n) \) for any input?
  – Knuth-Morris-Pratt’77: deterministic
  – Karp-Rabin’81: randomized

© 2001 by Charles E. Leiserson
Karp-Rabin Algorithm

- A very elegant use of an idea that we have encountered before, namely... 
  HASHING!
- Idea:
  - Hash all substrings $T[1...m], T[2...m+1], T[3...m+2]$, etc.
  - Hash the pattern $P[1...m]$
  - Report the substrings that hash to the same value as $P$
- Problem: how to hash $n-m$ substrings, each of length $m$, in $O(n)$ time?

Implementation

- Attempt I:
  - Assume $\Sigma = \{0, 1\}$
  - Think about each $T[s+1...s+m]$ as a number in binary representation, i.e.,
    $t_s = T[s+1]2^0 + T[s+2]2^1 + ... + T[s+m]2^{m-1}$
  - Find a fast way of computing $t_{s+1}$ given $t_s$
  - Output all $s$ such that $t_s$ is equal to the number $p$ represented by $P$

The great formula

- How to transform
  $t_s = T[s+1]2^0 + T[s+2]2^1 + ... + T[s+m]2^{m-1}$
  into
  $t_{s+1} = T[s+2]2^0 + T[s+3]2^1 + ... + T[s+m]2^{m-2} + T[s+m+1]2^{m-1}$
- Three steps:
  - Subtract $T[s+1]2^0$
  - Divide by 2 (i.e., shift the bits by one position)
  - Add $T[s+m+1]2^{m-1}$
- Therefore: $t_{s+1} = (t_s - T[s+1]2^0)/2 + T[s+m+1]2^{m-1}$

Algorithm

- Can compute $t_{s+1}$ from $t_s$ using 3 arithmetic operations
- Therefore, we can compute all $t_0, t_1, ..., t_{n-m}$ using $O(n)$ arithmetic operations
- We can compute a number corresponding to $P$ using $O(m)$ arithmetic operations
- Are we done?
Problem

• To get $O(n)$ time, we would need to perform each arithmetic operation in $O(1)$ time
• However, the arguments are $m$-bit long!
• It is unreasonable to assume that operations on such big numbers can be done in $O(1)$ time
• We need to reduce the number range to something more manageable

Hashing

• We will instead compute
  \[ t'_s = T[s+1]2^0 + T[s+2]2^1 + \ldots + T[s+m]2^{m-1} \mod q \]
  where $q$ is an “appropriate” prime number
• One can still compute $t'_{s+1}$ from $t'_s$:
  \[ t'_{s+1} = (t'_s - T[s+1]2^0 \cdot 2^{-1} + T[s+m+1]2^{m-1}) \mod q \]
• If $q$ is not large, i.e., has $O(\log n)$ bits, we can compute all $t'_s$ (and $p'$) in $O(n)$ time

Problem

• Unfortunately, we can have false positives, i.e., $T \neq p$ but $t'_s = p$
• Need to use a random $q$
• We will show that the probability of a false positive is small $\rightarrow$ randomized algorithm

False positives

• Consider any $t_s \neq p$. We know that both numbers are in the range $\{0 \ldots 2^m-1\}$
• How many primes $q$ are there such that $t_s \mod q = p \mod q \equiv (t_s-p) = 0 \mod q$?
• Such prime has to divide $x = t_s-p$
• Recall $x \leq 2^m$
• Represent $x = p_1^{e_1}p_2^{e_2} \ldots p_k^{e_k}$, $p_i$ prime, $e_i \geq 1$
• Since $2 \leq p_1$, we have $2^k \leq x \leq 2^m \rightarrow k \leq m$
• There are $\leq m$ primes dividing $x$
Algorithm

- Let $\prod$ be a set of $2nm$ primes, each having $O(\log n)$ bits
- Choose $q$ uniformly at random from $\prod$
- Compute $t'_0, t'_1, \ldots, t'_q$
- For each $s$, the probability that $t'_s = p'$ while $T_s \neq P$ is at most $m/2nm = 1/2n$
- The probability of any false positive is at most $(n-m)/2n \leq 1/2$

"Details"

- How do we know that such $\prod$ exists?
- How do we choose a random prime from $\prod$ in $O(n)$ time?

Prime density

- Primes are "dense". I.e., if $\text{PRIMES}(N)$ is the set of primes smaller than $N$, then asymptotically
  $|\text{PRIMES}(N)|/N \sim 1/\log N$
- If $N$ large enough, then
  $|\text{PRIMES}(N)| \geq N/(2\log N)$

Prime density continued

- If we set $N = 9mn \log n$, and $N$ large enough, then
  $|\text{PRIMES}(N)| \geq N/(2\log N) \geq 2mn$
- All elements of $\text{PRIMES}(N)$ are $\log N = O(\log n)$ bits long
Prime selection

• Still need to find a random element of PRIMES(N)
• Solution:
  – Choose a random element from \{1 \ldots N\}
  – Check if it is prime
  – If not, repeat

Prime selection analysis

• A random element \( q \) from \{1\ldots N\} is prime with probability \( \sim 1/\log N \)
• We can check if \( q \) is prime in time polynomial in \( \log N \) (trust me 😊)
• Therefore, we can generate random prime \( q \) in \( o(n) \) time
• The rest of the algorithm takes \( O(n) \) time

© 2001 by Charles E. Leiserson