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" \( 0 \leq s \leq n-m \) such that \( T[s+1\ldots s+m]=P \)
- **Example:**
  - \( \Sigma = \{a,b,\ldots,z\} \)
  - \( T[1\ldots 18]="to be or not to be" \)
  - \( P[1\ldots 2]="be" \)
  - Shifts: 3, 16

Simple Algorithm

```plaintext
for s ← 0 to n-m
    Match ← 1
    for j ← 1 to m
        if T[s+j] ≠ P[j] then
            Match ← 0
            exit loop
    if Match=1 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

Karp-Rabin Algorithm

• A very elegant use of an idea that we have encountered before, namely... HASHING!

• Idea:
  – Hash all substrings $T[1...m], T[2...m+1], T[3...m+2], \ldots$
  – 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.,
    \[ t_s = T[s+1]2^0 + T[s+2]2^1 + \ldots + T[s+m]2^{m-1} \]
    \[ T = 111010010111 \; ; \; P = 101 = 5 \]
    (note that the most significant digit is the rightmost one)
  – Find a fast way of computing $t_{s+1}$ given $t_s$
    \[ t_s = 111 ; t_s = 110 = 3 \; ; \; t_s = 101 = 5 \]
    \[ t_s = 101 = 5 ; t_s = 010 = 2 \; ; \; t_s = 001 = 4 \]
  – Output all $s$ such that $t_s$ is equal to the number $p$ represented by $P$
Warning

- In this lecture, \( p \) is for “pattern”, not for “prime”.
- All primes are denoted by \( q \)

The great formula

- How to transform \( t_s = T[s+1]2^0 + T[s+2]2^1 + \ldots + T[s+m]2^{m-1} \)
  into \( t_{s+1} = T[s+2]2^0 + T[s+3]2^1 + \ldots + T[s+m+1]2^{m-1} \).

  e.g. \( T = 111010010111 \) – need to transform \( 111 \) to \( 110 \) to \( 101 \)

- 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 \( t_0, t_1, \ldots, t_{m-1} \)
  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

Recall

- For any integers $a, b, q$
  - $(ab) \mod q = ((a \mod q)(b \mod q)) \mod q$
  - $(a+b) \mod q = ((a \mod q) + (b \mod q)) \mod q$

The great formula (revised)

- How to transform $t' = \left( T[s+1]2^0 + T[s+2]2^1 + T[s+3]2^2 + \ldots + T[s+m]2^{m-1} \right) \mod q$
  into $t'_v = \left( T[s+2]2^0 + T[s+3]2^1 + \ldots + T[s+m]2^{m-2} + T[s+m+1]2^{m-1} \right) \mod q$

  e.g. $T=111010010111$--need to transform 111 => 110 => 101

- Four steps:
  - Subtract $T[s+1]2^0$ (either 0 or 1)
  - Divide by $2$ (i.e., shift the bits by one position)
  - Add $T[s+m+1]2^{m-1} \mod q$
  - Compute $\mod q$ of the result

- Therefore: $t'_v = \left( \{t'_v - T[s+1]2^0\}/2 + T[s+m+1]2^{m-1} \right) \mod q$
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 \( t'_s \):
  \[ t'_{s+1} = (t'_s - T[s+1]2^0) * 2^{-1} + T[s+m+1]2^{m-1} \mod q \]

• If \( q \) is not large, i.e., has \( O(\log n) \) bits, we can compute all \( t'_s \) (and \( p' \)) in \( O(n) \) time

• Recall \( t'_s = t_s \mod q \).

• Only if \( t'_s = p \mod q \) we check if \( T[s+1 \ldots s+m-1] = p \) (might be a false positive)

Algorithm

• Let \([\prod] \) be a set of \( 2nm \) primes, each having \( O(\log n) \) bits

• Choose \( q \) uniformly at random from \([\prod]\)

• Compute \( t'_0, t'_1, \ldots, \) and \( p' \)

• If \( t'_s = p' \) check if \( T[s+1 \ldots s+m-1] = p \) (might be a false positive.)

We will show that with high probability we have no false positive

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
  \[ t_s \mod q = p \mod q \]
  that is,
  \[ (t_s - p) \mod q = 0 \mod q \]
  \( t_s - p = Kq \) for some integer \( K \), and \( q \) is a divisor of \( t_s - p \)

• Such prime has to divide \( x_s = t_s - p \)

• Recall \( x_s \leq 2^m \)

• Represent \( x_s = q_1^{e_1} q_2^{e_2} \ldots q_k^{e_k} \), \( q_i \) prime, \( e_i \geq 1 \)

• Since \( 2 \leq q_i \), we have \( 2^k \leq x_s \leq 2^m \rightarrow k \leq m \)

• There are \( \leq m \) primes dividing \( x_s \)
**Analysis**

- Call a prime $q$ a **bad prime** for $x_i$ if $q$ divides $t_i - p$. that is, $t_i \mod q = p \mod q$ (false positive)

**Lemma**: $q$ is a bad prime with probability $\leq \frac{1}{2}$

- Let $\prod$ be a set of $2nm$ primes, each having $O(\log n)$ bits
- We choose $q$ uniformly at random from $\prod$, and compute $t_0, t_1, \ldots, t_{n-m}$ and $p$
- Pretend that we are crossing out from $\prod$ all the bad primes.
- Cross out the divisors of $t_s - q$, for $s=0,1,2,\ldots,n-m$
- At most $mn$ primes are crossed out.
- So at least $mn$ are left in $\prod$
- We picked $q$ at random, so with probability $\geq \frac{1}{2}$ it is *not* a bad prime. QED

**Example** $n=2, m=4, |\prod|=16$

- $x_1 = 15, x_2 = 12, x_3 = 26, x_4 = 49$
- $\prod = \{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53\}$

**Conclusion**

- With probability $\geq \frac{1}{2}$ we *don’t have any* false positives
- Also, the expected number of false positive is small, so the expected running time is $O(n)$.

**“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 PRIMES(N) is the set of primes smaller than N, then asymptotically
  \[ \frac{|\text{PRIMES}(N)|}{N} \sim \frac{1}{\log N} \]
- If N large enough, then
  \[ |\text{PRIMES}(N)| \geq \frac{N}{2\log N} \]

Prime density continued

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