COL 351: Analysis and Design of Algorithms

Lecture 22

Pattern Matching

Given: String T = $(t_{n-1}, \ldots, t_1, t_0)$ and a pattern X = $(x_{k-1}, \ldots, x_1, x_0)$, both binary.

Find: If there exists a sub-string of T that is identical to X.

Pattern

X = $(x_{k-1}, \ldots, x_1, x_0)$

$$N_X = 2^{k-1}x_{k-1} + \cdots + 2^1 x_1 + 2^0 x_0$$

(decimal form of X)

$X = 11001$

$N_X = 16 + 8 + 1 = 25$

T = $(t_{n-1}, \ldots, t_1, t_0)$

$$N_T(j) = 2^{k-1}t_{j+k-1} + \cdots + 2^1 t_{j+1} + 2^0 t_j$$

(decimal form of $(t_{j+k-1}, \cdots, t_{j+1}, t_j)$)

Algorithm

Flag= False

For $j = 0$ to $(n - k)$:

    If $N_X = N_T(j)$ **then**

        Flag = True

Return Flag

Time = $O(nk)$

Algorithm

p = random prime in range $[2, n^4]$.

Hash Function $H : z \to z \mod p$

Flag= False

For $j = 0$ to $(n - k)$:

  If $H(N_X) = H(N_T(j))$ **then**

    Flag = True

Return Flag

Show:

- Answer returned is correct with probability $(1 - 1/n)$.
- Implementation in $O(n)$ time.

Computing random prime in range $[2, n^4]$

Prime Number Theorem: Number of primes in the range $[2,L]$ is $\Theta\left(\frac{L}{\log L}\right)$.

- Probability $\left(\begin{array}{l}\text{a random number in} \\ \text{range } [2, L] \text{ is prime}\end{array}\right) \geq \frac{c}{\log L}$

AKS Primality Test (By Agarwal, Kayal, Saxena)

- Checks if a number n is prime or not in $O(\log^c(n))$ time, for some fixed $c \geq 1$.

Observations

Claim 1: For any integer $z \leqslant 2^k$, the number of distinct prime factors of z is at most k.

$$z = \underbrace{p_1^{i_1}}_{\geq 2} \cdot \underbrace{p_2^{i_2}}_{\geq 2} \cdot \cdots \cdot \underbrace{p_\alpha^{i_\alpha}}_{\geq 2} \leq 2^k$$

$$\Rightarrow \alpha \leq k$$

Claim 2: For any j, the number of distinct prime factors of $(N_T(j) - N_X)$ is at most n.

$$N_T(j) \leq 2^n$$

$$N_X \leq 2^k \leq 2^n$$

$$\therefore N_T(j) - N_X \leq 2^n$$

$\Rightarrow$ Distinct Prime factors $\leq n$

Claim 3: For any $j \in [0, n-k]$ with $N_T(j) \neq N_X$.

$$\underbrace{\textbf{Prob}\Big(p \text{ divides } (N_T(j) - N_X)\Big)}_{\text{Probability of error at location } j} \leqslant \frac{1}{n^2}.$$

$$\frac{\text{No of Prime factors of } N_T(j) - N_X}{\text{No of choices for "}p\text{"}} \leq \frac{n}{\Theta(n^4/\log n^4)} \leq \frac{1}{n^2}$$

Claim 4: If X is not a substring of T. Then

$$\textbf{Prob}\Big(\exists\, j, \text{ such that } H(N_T(j)) = H(N_X)\Big) \leqslant \frac{1}{n}.$$

By union bound,

$$\text{Prob}(\text{Error in entire algo}) \leq \sum_{j=0}^{n-k} \text{Prob}(\text{Error at location } j) \leq \frac{n-k+1}{n^2} \leq \frac{1}{n}$$

How to recursively compute $N_T(j)$?

$$N_T(j) \;=\; \text{Decimal form of } (t_{j+k-1}, \cdots, t_{j+1}, t_j)$$

Removed

$$N_T(j+1) \;=\; \text{Decimal form of } (t_{j+k}, t_{j+k-1}, \cdots, t_{j+1})$$

Added

$$N_T(j+1) = \frac{N_T(j) - t_j}{2} \; + \; 2^{k-1}\, t_{j+k}$$

Recursively Computing Hash of $N_T(j)$

$$N_T(j+1) \;=\; \frac{N_T(j) - t_j}{2} \;+\; 2^{k-1}\; t_{j+k}$$

$$H(N_T(j+1)) \;=\; N_T(j+1)$$

$$= \Big((N_T(j) - t_j) \cdot 2^{-1} \;-\; 2^{k-1} \cdot t_{j+k} \Big) \bmod p$$

$$= \Big(\big(\underbrace{N_T(j) \bmod p}_{H(N_T(j))} - \underbrace{t_j \bmod p}_{O(1)\text{ time}} \big) \cdot \underbrace{\boxed{2^{p-2} \bmod p}}_{A} \;-\; \underbrace{\boxed{2^{k-1} \bmod p}}_{B} \cdot \overbrace{t_{j+k} \bmod p}^{O(1)\text{ time}} \Big) \bmod p$$

> <u>Claim</u> : $H(N_T(j+1))$ can be obtained from $H(N_T(j))$ in $O(1)$ space, $O(1)$ time if we know A, B.

Efficiently Computing A and B

$A = 2^{p-2} \mod p$ $\qquad\qquad$ $B = 2^{k-1} \mod p$

$$2^{z} \pmod p = \begin{cases} \left(2^{\lfloor z/2 \rfloor} \pmod p\right)^2 \bmod p & \text{if } z \text{ is even} \\ \left(2^{\lfloor z/2 \rfloor} \pmod p\right)^2 \cdot 2 \bmod p & \text{if } z \text{ is odd} \end{cases}$$

$$\boxed{\text{Claim: If we know } 2^{\lfloor z/2 \rfloor} \pmod p \text{ then we can compute } 2^{z} \pmod p \text{ in } O(1) \text{ space, time.}}$$

Computing A – $O(\log p) = O(\log n)$ Time, $O(1)$ Space

Computing B – $O(\log k) = O(\log n)$ Time, $O(1)$ Space

Base Case

$$H(N_T(0)) = \text{Decimal}(\underset{\substack{\uparrow \\ 2^{k-1}}}{t_{k-1}} \; \cdots\cdots \; \underset{\substack{\uparrow \\ 2^{1}}}{t_1} \;\; \underset{\substack{\uparrow \\ 2^{0}}}{t_0}) \bmod p$$

We can compute $2^{i+1} \bmod p$ from $2^i \bmod p$ in $O(1)$ space, $O(1)$ query time.

Claim: We can compute $H(N_T(0))$ and $H(N_x)$ in $O(k)$ time and $O(1)$ space.