Lecture #4 — collection of texts

- Basic problem: search P as a substring of T

- Old applic = Editors / Spell check — Now = { Antivir ⇒ Σ = Ascii (virus) ; search engines (words)

FINGERPRINT Useful tool in many contexts

$$h(s) = \sum_{i=1}^{|s|} 2^{|s|-i} \, s[i]$$ just as a binary representation

of course $s' = s'' \iff h(s') = h(s'')$

- Where is the problem ? We cannot manipe pattern longer than 4/8 bytes.

→ We need to squeeze $h(s')$ in a <u>regular</u> way; without introducing many conflicts so that we can "count" them.

idea reset modular arithmetic : $h_p(s) = h(s) \bmod p$

① p is chosen as a prime (many properties)

② use of Horner rule to manipe small intermediary results

③ incremental computation is possible

m-long substring

$$h_p(T_{r+1}) = 2 \times h(T_r) - 2^m T[r] + T[r+m] \qquad O(1) \text{ time}$$

T_{r+1} ↖ $T[r+1, r+m]$

FACT Computing the $N-m+1$ fingerprints of m-long substrings takes $O(N)$ time, which is independent of m.

Prop 1 # primes smaller than n are $\left[\frac{n}{\ln n}, \frac{1.26\, n}{\ln n}\right]$ (clean)

Prop 2. If $n \geq 29$, the product of all primes $\leq n$ is larger than 2^n

Cor 3. If $n \geq 29$ and $x \leq 2^n$ then x has fewer than $\pi(n)$ prime divisors (distinct).

TEO. $|P| = n$, $|T| = m$ with $n \cdot m \geq 29$

p prime $\leq I$, where I will be set next

$\mathbb{P}(\text{false match}) \leq \dfrac{\pi(n \cdot m)}{\pi(I)}$

r false match $\Leftrightarrow$ $P \neq T_r$ but $h_p(P) = h_p(T_r)$ $\Rightarrow$ p divides $h(P) - h(T_r)$

R all positions where $P \neq T_r$ $\Leftrightarrow$ $\exists$ false match $\Leftrightarrow$ p divides $\prod_{r \in R} [h(P) - h(T_r)]$

$\Rightarrow$ p is one of the prime divisors of that number

$\Rightarrow$ How large is it? $|h(P) - h(T_r)| \leq 2^m$

hence $|\prod \ldots| \leq 2^{n \cdot m}$

$\Rightarrow$ # prime divisors (distinct) $\leq \pi(n \cdot m)$

$\mathbb{P}(\text{false match}) = \mathbb{P}(p \text{ is one of those divisors}) \leq \dfrac{\pi(n \cdot m)}{\pi(I)}$ ← p selected $\leq I$

what about k primes? All k primes will give a false match $\Rightarrow \left[\dfrac{\pi(n \cdot m)}{\pi(I)}\right]^k$

a more refined argument observes that if r is a false match then the k primes divide $|H(P) - H(T_r)| \leq 2^{nm}$, and they are ones in $\pi'(nm)$

$\mathbb{P}(\ ) \leq nm \cdot \left[\dfrac{\pi(nm)}{\pi(I)}\right]^k \Rightarrow$ $I = n^2 \cdot m \log^* m / \ln m$

$\dfrac{nm}{\ln nm} \Big/ \dfrac{n^2 m \ln m}{2 \ln n + \ln m + \ln\ln m} \leq \dfrac{2 \ln n}{n \ln m}$

what about I? We must fit every arithmetic calculation in one RAM word

Typically $I = m \cdot n^2$ $\Rightarrow$ $\dfrac{\pi(n \cdot m)}{\pi(n^2 m)} \leq \dfrac{2.53}{n} = O\left(\dfrac{1}{n}\right)$

Since Ops $\leq 4(\log n + \log m)$

Also $\Rightarrow$ # exp false matches is $O(1)$ $\Rightarrow$ checking them takes $O(m)$

So we get a Las Vegas algorithm

what about changing p at each false match? $\mathbb{P}(t \text{ errors}) \leq \left(\dfrac{2.53}{n}\right)^t$

What about a dictionary of searched patterns ?

⇓

all same length

① Use a Trie built on the patterns

-) more flexible ⇒ diff. lengths

-) ~~[illegible]~~ • T × ln (D)

② Put all patterns in a hash table according to KR-fingerprint

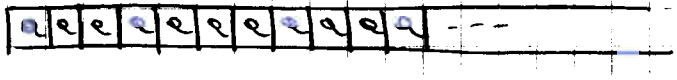
$$\mathbb{P}(\text{false match}) \leq d \cdot \left[\frac{\pi(n \cdot m)}{\pi(I)}\right]^k$$

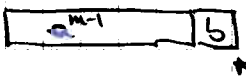
UNION BOUND

play with this to recover the exact bound and cope with d = # patterns.

KMP

* No randomization
* circumvent the brute-force : $O(n \cdot m)$

T = 

P = 

need to check over and over n times to establish P not in T at each position.

Brute-force

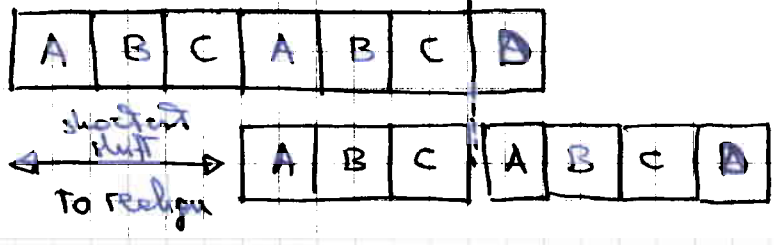
$n \cdot m$

Avoid Rescanning ⇒ next function.

T = A B C A B C H A B C D

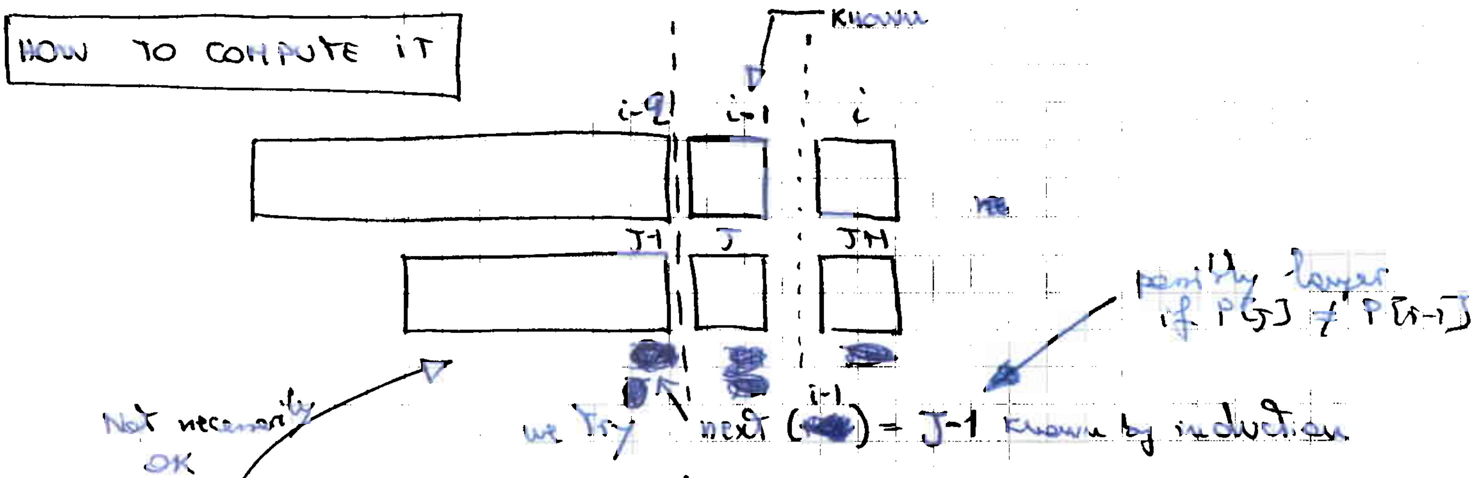
i

- Preprocess the pattern



next (i) = longest prefix of P[1, i-1] which is a suffix of that string.

HOW TO COMPUTE IT



we try next (i-1) = J-1 known by induction

base { next (1) = -1
next (2) = 0

next (i) = { J if P[i-1] = P[J] && J-1 = next (i-1)
ITERATE ON J else

```
Compute_next (P)
   next (1) = -1; next (2) = 0;
   for (i = 3; i ≤ m; i++)
         J = next (i-1) + 1;
         while ((T[i-1] ≠ T[J]) && J > 0)
               J = next (J) + 1;
         next (i) = J;
```

We must try all shifts:

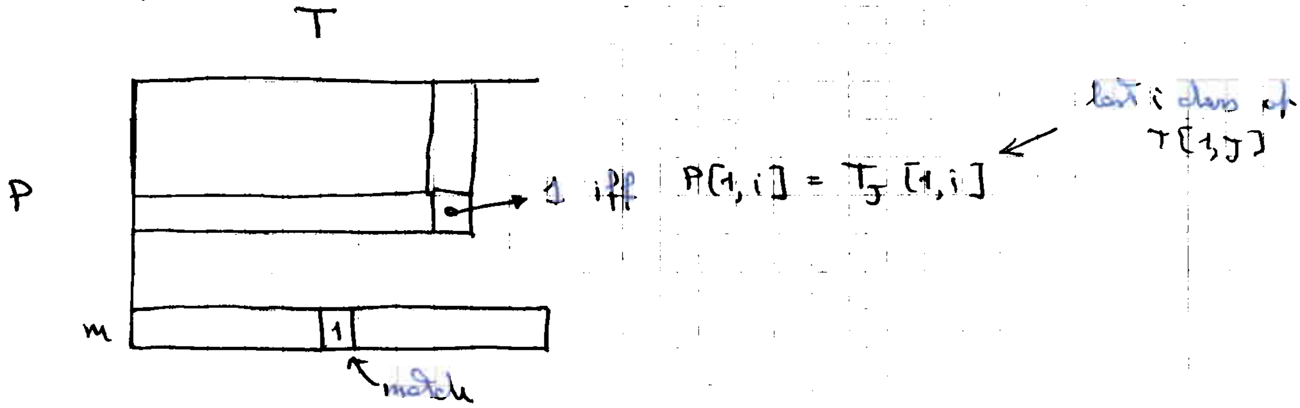
P = a b a b d a b a b c c a b a b d a b a b | a | ☒ (positions 1 2 3 4 5 6 7 8 9 10 … 18 19 20 21 22)

a b a b d a b a b	c	next (21) = 9	J = 10, c ≠ a
a b a b	d	next (10) = 4	d ≠ a, J = 5
a b	a	next (5) = 2	a = a, J = 3

next (22) = ?

⟹ next (22) = 3

AGREP — Turn everything into shift + AND ops.



IDEA ① Compute M column-by-column rightward.

② $U[i][\sigma] = 1$ iff $P[i] = \sigma$

characteristic matrix for P

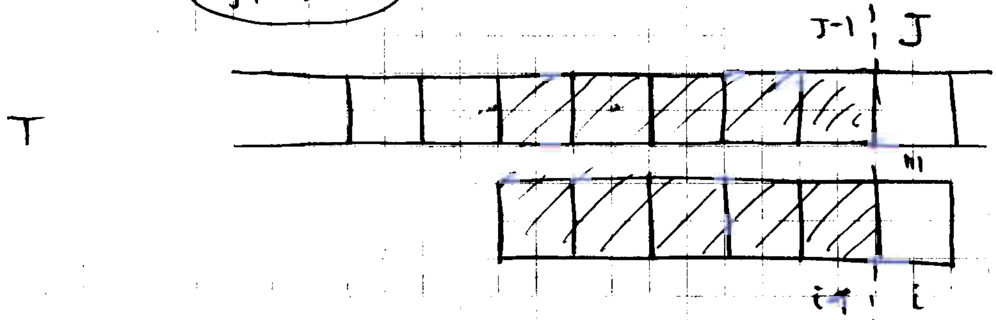
P	A	B	C	D
A	1	0	0	0
B	0	1	0	0
A	1	0	0	0
A	1	0	0	0
D	0	0	0	1

(columns: Σ)

③ Bit Shift (v) = [1 | V_{m-1}] ← drop last bit (1: manage special case)

$M[i][J] = 1$ iff $M[i-1, J-1] = 1$ && $P[i] = T[J]$ ($\sigma = T[J]$)

$P[1, i]$ ≡ $T_J[1, i]$; $P[1, i-1]$ ≡ $T_{J-1}[1, i-1]$; $P[i] = T[J]$ ≡ $U[i][T[J]] = 1$



This holds for all ⓘ ⊕ I do not want to make checks

$\text{BitShift}(M[\,][J-1])$ & $U[\,][T[J]]$

First bit = 1 because this way we correctly manage the first bit of a column, for which we have no carry from above, it is = 1 iff $U[1][T[J]]$ (first char of P matches T[J])

	b	x	a	b	a	a	c
a					1	1	0
b					0	0	0
a					1	0	0
a					0	1	0
d					0	0	0

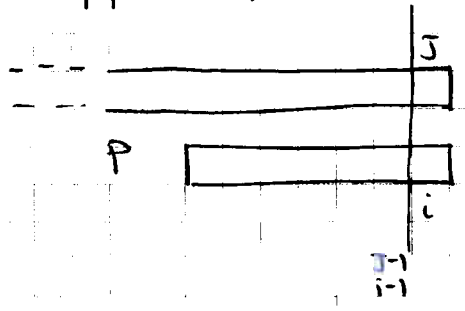
$j-1$ j

$M[\,][\sigma] = \text{BitShift}(M[\,][\sigma])$ & $U[\,][a]$

$$\text{BitShift}\left(\begin{bmatrix}1\\0\\1\\0\\0\end{bmatrix}\right) \;\&\; \begin{bmatrix}1\\0\\1\\1\\0\end{bmatrix} = \begin{bmatrix}1\\1\\0\\1\\0\end{bmatrix} \;\&\; \begin{bmatrix}1\\0\\1\\1\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\\1\\0\end{bmatrix}$$

How to extend to k mismatches — k matrices

$M^k[i][j] = 1$ iff $P[1,i]$ matches $T_j[1,i]$ with at most k mismatches



RECURSIVE ?

Two cases :

① $P[i] = T[j]$ & $\leq k$ mismatches are in $P[1,i-1]$, $T_{j-1}[1,i-1]$

② $P[i] \neq T[j]$ & $\leq k-1$ mismatches are in ↗

no cost

$$M^k[\,][j] = \left(\text{BitShift}\left(M^k[\,][j-1]\right) \;\&\; U[\,][T[j]]\right)$$

$$\| \underbrace{\left(\text{BitShift}\left(M^{k-1}[\,][j-1]\right)\right)}_{\text{no cost about } P[i] \neq T[j]}$$

	B	X	A	B	A	A	C
A					1	1	0
B					0	0	0
A					1	0	0
A					0	1	0
D					0	0	0

M^0 ↑ J

	B	X	A	B	A	A	C
A					1	1	1
B					0	1	1
A					1	0	0
A					0	1	0
D					0	0	1

M^1 ↑ J

$$M^1[\,][J] = \left(\text{BitShift}\left(\begin{bmatrix}1\\0\\1\\0\\0\end{bmatrix}\right) \;\&\; \begin{bmatrix}1\\0\\1\\1\\0\end{bmatrix}\right) \;||\; \text{BitShift}\left(\begin{bmatrix}1\\0\\1\\0\\0\end{bmatrix}\right) =$$

$M^1[\,][J-1]$ $\quad$ $U[A]$ $\quad$ $M^0[\,][J-1]$

Complexity $\quad n * K = \frac{m}{w} \overset{\text{Trivially}}{=} O(n \cdot K)$

$$M^1[\,][J+1] = \underbrace{\left(\text{BitShift}\left(\begin{bmatrix}1\\1\\0\\1\\0\end{bmatrix}\right) \;\&\; \begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix}\right)}_{\text{case of match @ } J \text{ and has errors before is not possible}} \;||\; \underbrace{\text{BitShift}\left(\begin{bmatrix}1\\0\\0\\1\\0\end{bmatrix}\right)}_{\begin{bmatrix}1\\1\\0\\0\\1\end{bmatrix}}$$

How do we manage sets of chars ?

→ Union of $U[\sigma]$, $\forall \sigma$ in the set.