String Matching

Σ alphabet

T, P strings over Σ

$n = |T|, \quad m = |P|$

P occurs in T with shift s

$\Updownarrow$

$T[s+1..s+m] = P[1..m]$

s is a valid shift when P occurs in T with shift s

String matching problem for pattern P

input: text T

output: all valid shifts / occurences of P

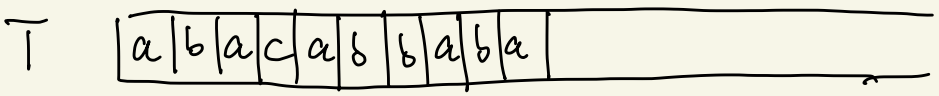
Notation

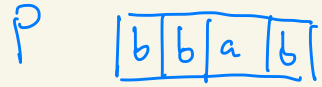
Σ^* = set of all finite length strings over Σ

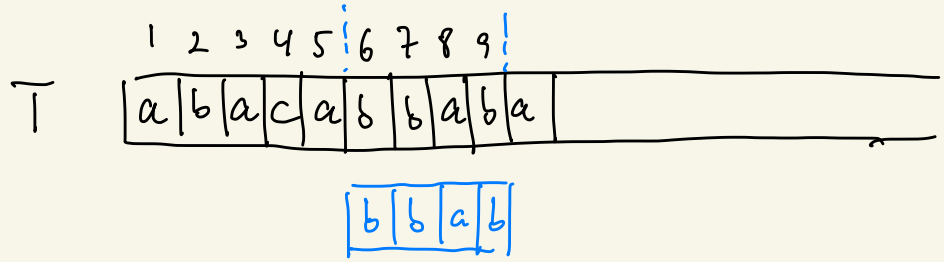
ε empty string

$w \sqsubset x$ w is a prefix of x $\iff$ $x = wy$ for some $y \in \Sigma^*$

$w \sqsupset x$ w is a suffix of x $\iff$ $x = yw$ ---







match

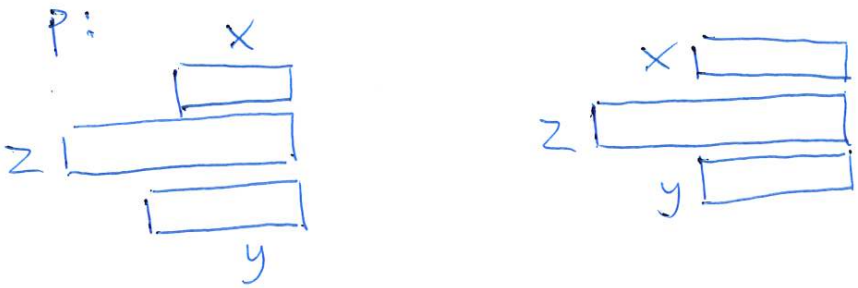
$s = 5$ valid shift as $T[6..9] = P$

Lemma 32.1 (overlapping-suffix lemma)

Let $x, y, z \in \Sigma^*$ such that

$x \sqsupset z$ and $y \sqsupset z$

- if $|x| \leq |y|$ then $x \sqsupset y$
- if $|y| \leq |x|$ then $y \sqsupset x$
- if $|x| = |y|$ then $x = y$



Notation

$T[i..j] = T[i]T[i+1]\cdots T[j]$

$P[1..k]$ is denoted P_k $\quad k \geq 0$

do for $T[1..k]$

So

P occurs in T with shift s

$\Updownarrow$

$P \sqsupset T_{s+m} = T[1..s+m]$

For a given shift s we can test whether s is a valid shift in time $\Theta(t+1)$ where t is the largest value s.t $P_t \sqsupset T_{s+t}$ $t \leq |P|$

Naive algorithm: try all possible shifts $s = 0, 1, 2, \ldots n-m$

Preprocessing: none

Running time $O((n-m+1)m)$ worst case

tight for $T = a^n$ $P = a^m$

Problem: the algorithm completely ignores info collected while trying shifts

The Rabin-Karp algorithm

Idea: think of P and T as numbers

and compare P ~~$\neq$~~ T_{s+m} by comparing $T[s+1..s+m]$

their numbers

$d = |\Sigma|$ so numbers base d

For simplicity assume $d = 10$ $\Sigma = \{0,1,2 \cdots 9\}$
but works for all finite alphabets

Ex $T[s+1..s+5] = 37133$

$P[1..5] = 37233$

$P_5 \neq T_{s+5}$ since $37133 \neq 37233$

Notation

p = decimal value of $P[1..m]$

t_s = decimal value of $T[s+1..s+m]$

So match $\Leftrightarrow p = t_s$

assumption: size of the numbers p and t_s can be handled (in constant time)
so $p \stackrel{?}{=} t_s$ takes $O(1)$

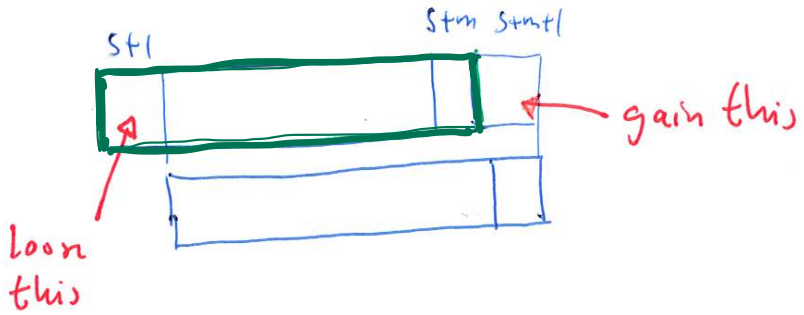
Calculation of p: use Horner's rule → see example

$$p = P[m] + 10\left(P[m-1] + 10\left(P[m-2] + \cdots + 10P[1]\right)\cdots\right)$$

similarly t_0 calculated from $T[1..m]$
in $\Theta(m)$

Calculating t_{s+1} from t_s:

$$t_{s+1} = 10\left(t_s - 10^{m-1}T[s+1]\right) + T[s+m+1]$$



So $O(1)$ to calculate next t_{s+1}
when 10^{m-1} is precalculated
(takes $O(m)$, can be done in $O(\log m)$)

Example of calculation using Horners rule

$$47632 = 4\cdot 10^4 + 7\cdot 10^3 + 6\cdot 10^2 + 3\cdot 10 + 2\cdot 10^0$$

$$= 2\cdot 10^0 + 3\cdot 10^1 + 6\cdot 10^2 + 7\cdot 10^3 + 4\cdot 10^4$$

$$= 2 + 10\left(3 + 10\left(6 + 10\left(7 + 10\cdot 4\right)\right)\right)$$

R-K-algorithm:

- Compute p in time $\Theta(m)$
- compute t_0 — " — $\Theta(1)$ report match if $p = t_0$
- compute t_1 — " — $\Theta(1)$ report match if $p = t_1$

⋮

- compute t_{n-m} in time $\Theta(1)$ report match if $p = t_{n-m}$

total time $\Theta(m)$ preprocessing

$\Theta(n-m+1)$ matching time

Too good to be true?

Yes!

Unrealistic that we can compare p and t_s in constant time when m is not small

Solution Compute p and t_s modulo q
for suitable value of q

Can calculate $p \bmod q$ in $\Theta(m)$ (by Horners rule)
$t_0 \bmod q$ in $\Theta(m)$ — " —
$t_1, \ldots t_{n-m}$ in total $\Theta(n-m+1)$

good value of q?

as large as possible s.t $10q$ fits in one computer word.

Then we can do all calculations with single precision arithmetic

Case of general alphabet $\Sigma = \{0, 1, 2, \ldots d-1\}$

Choose q s.t dq fits in one computer word

$$t_{s+1} = \left(d\left(t_s - T[s+1] \cdot h\right) + T[s+m+1]\right) \bmod q$$

when $h = d^{m-1} \bmod q$

new problem:

Spurious hits $t_s \equiv p \bmod q$ but $t_s \neq p$

Such hits must be eliminated

⇓

need to test all shifts with $t_s \equiv p \bmod q$ $\Theta(m)$ per test

Hence worst case matching time is

$\Theta((n-m+1)m)$ as the naive algorithm.

happens for $T = a^n$ $p = a^m$

Expected # hits

assumption $\bmod q$ ↓ acts like ≈ random map $\Sigma^* \to \mathbb{Z}_q$

(Compare with hashing)

If all the q values mod q are equally likely

then $\Pr(t_s \equiv p \bmod q) = \frac{1}{q}$

So expected # spurious hits is $\frac{O(n)}{q} = O(\frac{n}{q})$

Expected matching time is thus

$$O(n) + O\left(m\left(v + n/q\right)\right)$$

where v is the number of valid (correct) shifts

when v is $O(1)$ and $q \geq m$

this is $O(n)$

so total time (preprocessing + matching)

is $O(n+m) = O(n)$ as $n \geq m$

String matching with DFA's

DFA 5-tuple $M=(Q, q_0, A, \Sigma, \delta)$ Recall from DM565

- Q finite set of states
- q_0 initial (start) state
- $A \subseteq Q$ the accepting states
- Σ finite input alphabet
- $\delta : Q \times \Sigma \to Q$ transition function of M

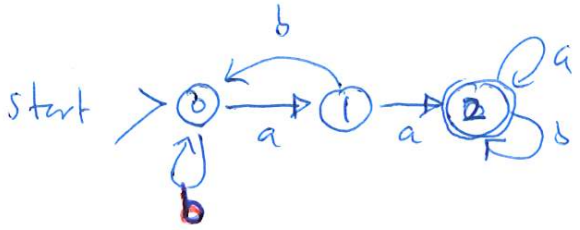
DFA ~ labelled digraph (labels on arcs)

each state is a vertex

each vertex has precisely

$|\Sigma|$ arcs leaving it

(may be loops)



M accepts $w \in \Sigma^*$

$\Updownarrow$

following arcs labelled by w we end in a state (vertex) q with $q \in A$

M rejects w if it does not accept it

M induces a function ϕ the final state function which is the state in which M resides after reading w

Hence M accepts $w \Leftrightarrow \phi(w) \in A$

$\phi(\varepsilon) = q_0$

$\phi(wa) = \delta(\phi(w), a) \quad w \in \Sigma^* \; a \in \Sigma$

String matching automata

P pattern $\longrightarrow$ $M = M(P)$

Goal M will accept each prefix T_{strm} of a text T for which the last $m = |P|$ characters of T_{strm} equal P.

Show that we can build M independently of T, such that same $M = M(P)$ works for all input texts.

Notation

Suffix function for pattern P:

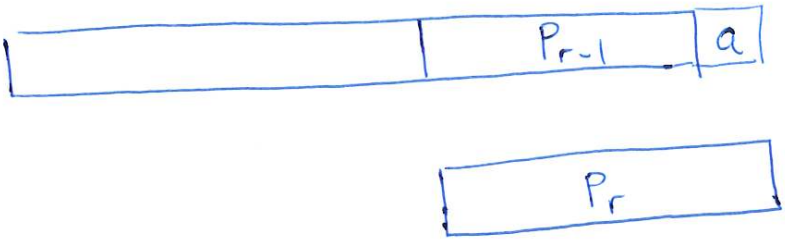
$\sigma : \Sigma^* \longrightarrow \{0, 1, 2, \ldots, m\}$

$$\sigma(x) = \max \{k \mid P_k \sqsupset x\}$$

σ is well defined as $\varepsilon \sqsupset x \quad \forall x \in \Sigma^*$

Lemma 32.2

$$\forall x \in \Sigma^*, a \in \Sigma \qquad \sigma(xa) \leq \sigma(x) + 1$$



If $r = \sigma(xa)$ then the last $r-1$ characters of x equal P_{r-1} so $\sigma(x) \geq r-1$

Define string matching automaton $M = M(P)$ $P = P[1..m]$ as follows

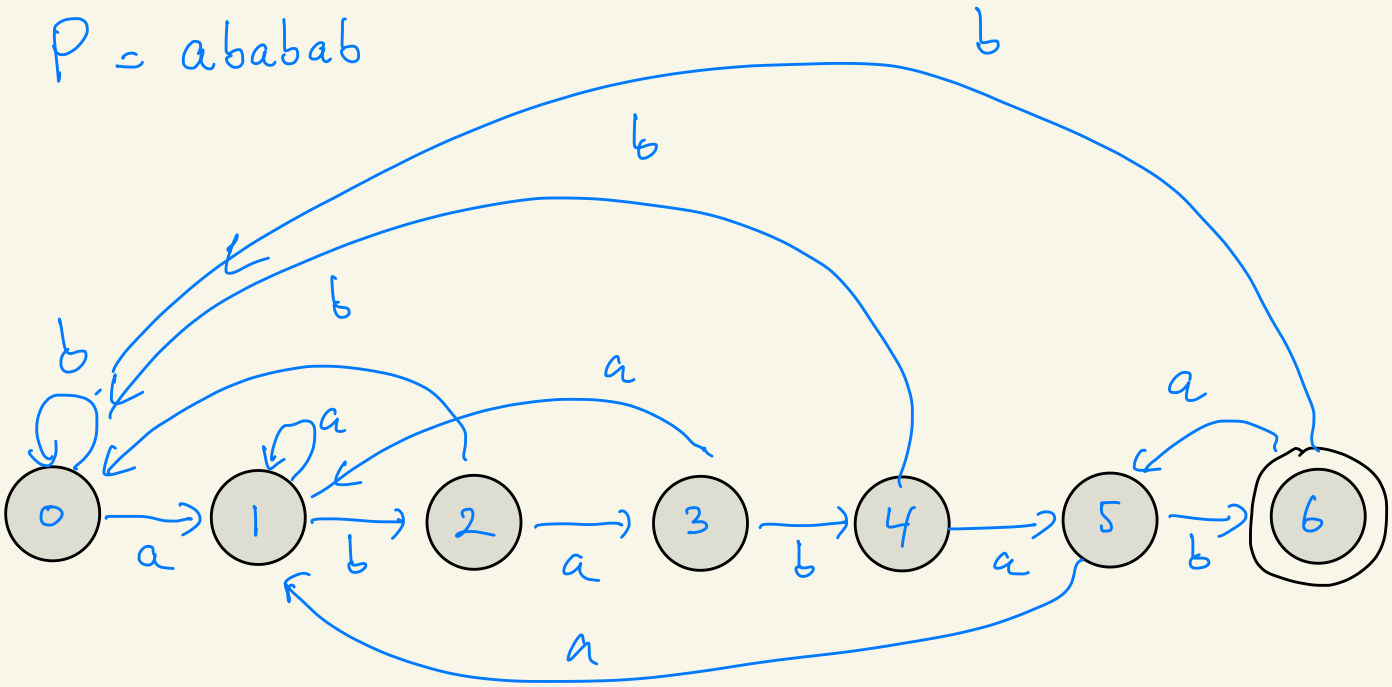
$Q = \{0, 1, 2, \ldots, m\}$ $q_0 = 0$ $A = \{m\}$

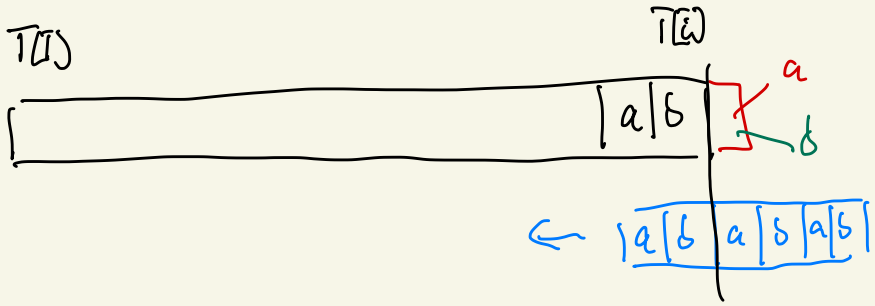
$\delta(q, a) = \sigma(P_q a)$

if the automata is currently

δ is chosen so that in state q the current string T_i satisfies that $P_q \sqsupset T_i$ and $q = \sigma(T_i)$

P = ababab





Let $q = \Phi(T_i)$ be the state we are in after reading T_i

Assume P_q is the longest prefix of P which is a suffix of T_i

Let $a = T[i+1]$

We want M to make a transition to the state $q' = \sigma(T_i a)$.

Lemma 32.3

$$\forall x \in \Sigma^*, a \in \Sigma : q = \sigma(x) \Rightarrow \sigma(xa) = \sigma(P_q a)$$

P: By definition of σ $P_q \sqsupset x$ so $P_q a \sqsupset xa$

Let $r = \sigma(xa)$ so $P_r \sqsupset xa$

Lemma 32.2 $\Rightarrow$ $r \leq q+1$ as $q = \sigma(x)$

so $|P_r| = r \leq q+1 = |P_q a|$

and thus $P_r \sqsupset P_q a$ so $r \leq \sigma(P_q a)$

but $\sigma(P_q a) \leq \sigma(xa)$ as $P_q a \sqsupset xa$

$\sigma(xa) = r$

so $\sigma(xa) = \sigma(P_q a)$

Theorem 32.4

If ϕ is the final state function of $M = M(P)$ and $T[1..n]$ is the input to M, then

$$\phi(T_i) = \sigma(T_i) \quad \text{for } i = 0, 1, 2 \ldots n$$

P: induction on i

$i = 0$ $\quad T_0 = \varepsilon$ and $\phi(T_0) = 0 = \sigma(T_0)$

induktion step assume $\phi(T_i) = \sigma(T_i)$

let $q = \phi(T_i)$ $\quad a = T[i+1]$

$$\begin{aligned}
\phi(T_{i+1}) &= \bar{\phi}(T_i a) && \text{as } a = T[i+1] \\
&= \delta(\phi(T_i), a) && \text{by def of } \phi \\
&= \delta(q, a) && \text{as } q = \phi(T_i) \\
&= \sigma(P_q a) && \text{by def of } \delta \\
&= \sigma(T_i a) && \text{by Lemma 32.3 + induction} \\
&= \sigma(T_{i+1}) &&
\end{aligned}$$

□

Invariant ⇒ correct function of M:

when M enters state q after reading T_i
q is the largest value s.t $P_q \sqsupset T_i$ thus
$q = m \iff P \sqsupset T_i$ so match

Computing the function δ

Naive way: when in state q

Let $k = \min\{m+1, q+2\} - 1$

try all values $k, k-1, \ldots, 0$

until $P_k \sqsupset P_q a$

$O(m^2)$ work per state and $a \in \Sigma$

total $O(m^3|\Sigma|)$

This can be improved to $O(m|\Sigma|)$

So we can report all occurences of P in T in $\Theta(m)$ matching time and $O(m|\Sigma|)$ preprocessing time.

More clever algorithm: KMP

not pervom!