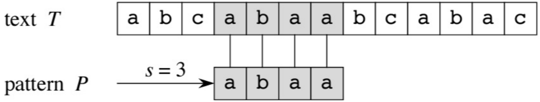
String Matching

Jie Wang

University of Massachusetts Lowell
Department of Computer Science

Problem Description

Input: A pattern $P[1..m]$ and text $T[1..n]$ over the same alphabet Σ.
Output: All **shifts** s such that $P[i] = T[s+i]$, where $i = 1, 2, \ldots, m$.

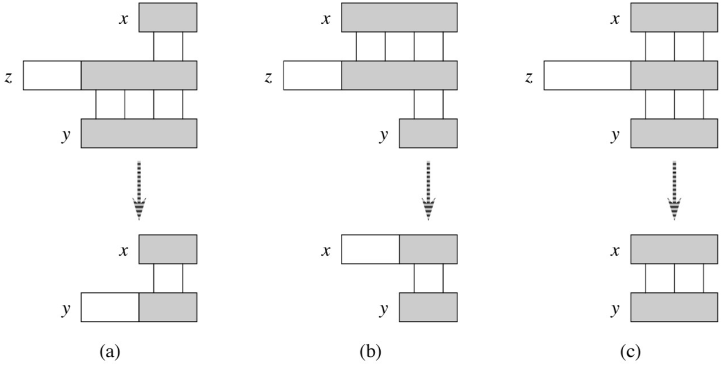


Four Algorithms

Algorithm	Preprocessing time	Matching time
Naive	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\,\lvert\Sigma\rvert)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$

Lemma 32.1 Suppose that x and y are both suffices of z. If $|x| \leq |y|$, then x is a suffix of y. Otherwise, y is a suffix of x. Moreover, $x = y$ iff $|x| = |y|$.

Proof.



The Naive Algorithm

- Brute force

```
NAIVE-STRING-MATCHER(T, P)
n = T.length
m = P.length
for s = 0 to n − m
    if P[1..m] == T[s + 1..s + m]
        print "Pattern occurs with shift" s
```

- Runtime: $\Theta((n - m + 1)m)$.

The Rabin-Karp Algorithm

- Perform well in practice, but the worst-case complexity is still $O(n-m+1)m$.
- The average-case complexity is better.
- Idea: Let $d = |\Sigma|$. Represent each symbol as a digit in the radix-d notation with the set of digits $\{0, 1 \ldots, d-1\}$.
- Each string can be represented as a number. Let p denote the number representing $P[1..m] = P[1]P[2]\cdots P[m]$, where

$$p = P[1]d^{m-1} + P[2]d^{m-2} + \cdots + P[m-1]d + P[m],$$

  and t_s the number representing $T[s+1..s+m]$.
- Check for $s = 1, 2, \ldots, n-m$ if $t_s = p$.

Runtime of Rabin-Karp

Computing p can be done in $\Theta(m)$ time using the Horner's rule:

$$p = P[m] + d(P[m-1] + d(P[m-2] + \cdots + d(P[2] + dP[1])\cdots)).$$

Computing all t_s can be done in $\Theta(n-m+1)$ time, for we can compute t_{s+1} from t_s in $\Theta(1)$ time as follows:

$$\begin{aligned}
t_s =& d^{m-1}T[s+1] + d^{m-2}T[s+2] + \cdots + dT[m+s-1] + T[m+s], \\
t_{s+1} =& d^{m-1}T[s+2] + d^{m-2}T[s+3] + \cdots + dT[m+s] + T[m+s+1] \\
=& d(d^{m-2}T[s+2] + \cdots + dT[m+s-1] + T[m+s]) + T[m+s+1] \\
=& d(d^{m-1}T[s+1] + d^{m-2}T[s+2] + \cdots + dT[m+s-1] + T[m+s] \\
& \quad - d^{m-1}T[s+1]) + T[m+s+1] \\
=& d(t_s - d^{m-1}T[s+1]) + T[m+s+1].
\end{aligned}$$

Problem: Values too Large

- Unfortunately, the values of p and t_s may be too large to work with conveniently.
- Solution: Compute p and t_s modulo a suitable modulus q.
- Choose q such that dq just fits within one computer word.
- Then

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

  where $h \equiv d^{m-1} \pmod{q}$.
- However, $t_s \equiv p \pmod{q}$ does not imply $t_s = p$. But $t_s \not\equiv p \pmod{q}$ implies $t_s \neq p$.
- Use the test $t_s \equiv p \pmod{q}$ to rule out invalid shifts s, then check further if $t_s = p$.
- Worst-case runtime: $\Theta((n - m + 1)m)$.

Expected Runtime

- Assume that there are v valid shifts.
- The probability that $p \bmod q = t_s \bmod q$ but s is invalid is $1/q$. (This result is nontrivial)
- Expected runtime: $O(n) + O(m(v + n/q))$.
- Become linear when $v = O(1)$ and $q \geq m$.

Pseudocode

```
RABIN-KARP-MATCHER(T, P, d, q)
n = T.length
m = P.length
h = d^(m-1) mod q
p = 0
t_0 = 0
for i = 1 to m                       // preprocessing
    p = (dp + P[i]) mod q
    t_0 = (d t_0 + T[i]) mod q
for s = 0 to n - m                   // matching
    if p == t_s
        if P[1..m] == T[s+1..s+m]
            print "Pattern occurs with shift" s
    if s < n - m
        t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) mod q
```

String-Matching Automata

- Let $\sigma(x) = \max\{k \mid P[1..k] \text{ is a suffix of } x\}$.
  - $\sigma(x)$ is the length of the longest prefix of P that is a suffix of x.
  - If x is a suffix of y, then $\sigma(x) \leq \sigma(y)$.
- Given a pattern $P[1..m]$, construct a string-matching automaton as follows:
  - Finite set of states: $Q = \{0, 1, \ldots, m-1\}$, where 0 is the initial state q_0 and m the final state.
  - The transition function δ is defined by

$$\delta(q, a) = \sigma(P[1..q]a),$$

    where $P[1..0]$ is the empty string ϵ.

Finite-State Function Φ

Φ is a function from a string to a state:

$$\Phi(\epsilon) = q_0,$$
$$\Phi(wa) = \delta(\Phi(w), a) \text{ for } w \in \Sigma^*, a \in \Sigma.$$

- Maintain the following invariant in the automaton while reading the text T:

$$\Phi(T[1..i]) = \sigma(T[1..i]).$$

  That is, maintain the state number to be the length of the longest prefix of P that is also a suffix of $T[1..i]$.

Suffix-function Recursion Lemma

Lemma 32.3 If $q = \sigma(T[1..i])$, then $\sigma(T[1..i]a) = \sigma(P[1..q]a)$.
Proof.

- We cannot have $\sigma(T[1..i]a) > q + 1$, for this would imply that $\sigma(T[1..i]) > q$, contradicting to the assumption.
- If $P[1..q+1]$ is a suffix of $T[1..i]a$, then $\sigma(T[1..i]a) = q + 1$.
- Since $q = \sigma(T[1..i])$, $P[1..q]$ is a prefix of $T[1..i]$. Since $\sigma(T[1..i]a) \leq q + 1$, we have $\sigma(T[1..i]a) = \sigma(P[1..q]a)$.

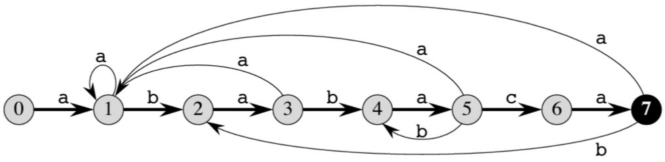
Correctness

Theorem 32.4 $\Phi(T[1..i]) = \sigma(T[1..i])$, for $i = 0, 1, \ldots, n$.
Proof. Induction on i.

- Basis: Since $T[1..0] = \epsilon$, $\Phi(T[1..0] = 0 = \sigma(T[1..0])$.
- Inductive hypothesis: Assume that $\Phi(T[1..i]) = \sigma(T[1..i])$.
- Induction step: Show that $\Phi(T[1..i+1]) = \sigma(T[1..i+1])$.
  Let $\Phi(T[1..i]) = q$. By induction hypothesis, $\sigma(T[1..i]) = q$, and hence $\sigma(T[1..i]a) = \sigma(P[1..q]a)$ for any $a \in \Sigma$.
  Let $a = T[i+1]$. We have

$$
\begin{aligned}
\Phi(T[1..i+1]) &= \Phi(T[1..i]a) && \text{(by the definition of } a\text{)} \\
&= \delta(\Phi(T[1..i]), a) && \text{(by definition of } \Phi\text{)} \\
&= \delta(q, a) && \text{(by the definition of } q\text{)} \\
&= \sigma(P[1..q]a) && \text{(by the definition of } \delta\text{)} \\
&= \sigma(T[1..i]a) && \\
&= \sigma(T[1..i+1]) && \text{(by the definition of } T[1..i+1]\text{)}
\end{aligned}
$$

Example: $P = ababaca$



(a)

	input			
state	a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

Pseudo Code

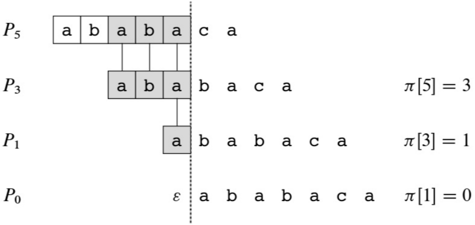
```
FINITE-AUTOMATON-MATCHER(T, δ, m)
n = T.length
q = 0
for i = 1 to n
    q = δ(q, T[i])
    if q == m
        print "Pattern occurs with shift" i − m
```

- Preprocessing time (constructing a string-matching automaton): $\Theta(m\Sigma)$.
- Runtime: $\Theta(n)$.

- An efficient implementation of string-matching automata.
- Let $\pi(q) = \max\{k \mid k < q \text{ and } P[1..k] \text{ is a suffix of } P[1..q]\ \}$.
- $P[1..k]$ is a suffix of $P[1..q]$ if $P[1..k] = P[q-k+1..q]$. (In another word, $\pi(q)$ is the longest prefix of P that is a proper suffix of $P[1..q]$.)

i	1	2	3	4	5	6	7
$P[i]$	a	b	a	b	a	c	a
$\pi[i]$	0	0	1	2	3	0	1

(a)



(b)

KMP Matcher

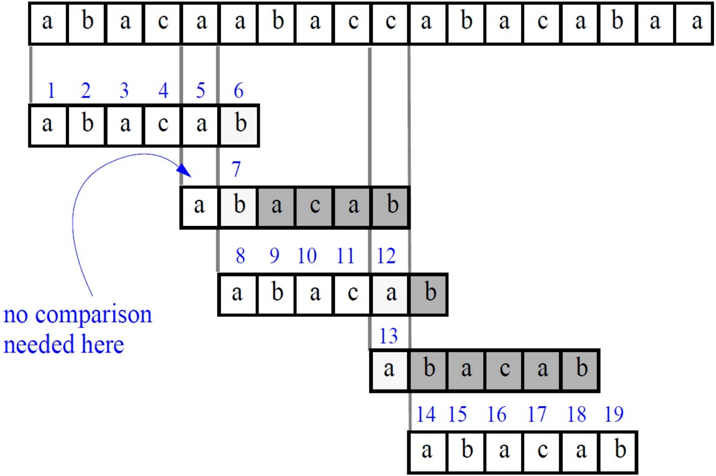
```
KMP-MATCHER(T, P)
n = T.length
m = P.length
π = COMPUTE-PREFIX-FUNCTION(P)
q = 0                                    // number of characters matched
for i = 1 to n                           // scan the text from left to right
    while q > 0 and P[q + 1] ≠ T[i]
        q = π[q]                         // next character does not match
    if P[q + 1] == T[i]
        q = q + 1                        // next character matches
    if q == m                            // is all of P matched?
        print "Pattern occurs with shift" i − m
        q = π[q]                         // look for the next match
```

Prefix Function π

COMPUTE-PREFIX-FUNCTION(P)

```
m = P.length
let π[1..m] be a new array
π[1] = 0
k = 0
for q = 2 to m
    while k > 0 and P[k + 1] ≠ P[q]
        k = π[k]
    if P[k + 1] == P[q]
        k = k + 1
    π[q] = k
return π
```

A KMP Example



Time Complexity

- Computing the prefix function: $\Theta(m)$.
  Within the **for** loop, count the the number of changes to k.
  Since $\pi[k] < k$ and k is incremented $m - 1$ times, k can be decreased at most $m - 1$ times.
- KMP Matcher: $\Theta(n)$ ($\Theta(n + m)$ including the computation of π).
  Within the **for** loop, count the number of changes to q.
  Since q is incremented $\Theta(n)$ times and $\pi[q] < q$, q can be decreased at most $\Theta(n)$ times.