String Matching

1. Problem statement
2. A naive approach
3. The Rabin-Karp algorithm
4. String matching with finite automata

1. Terminology

Let Σ^* denote the set of all strings a finite alphabet Σ.

concatenation: For strings x and y, the concatenation is the string xy and has length $|x| + |y|$.

prefix A string w is a prefix of x (denotation $w \sqsubset x$) if $x = wy$ for some string $y \in \Sigma^*$. If $w \sqsubset x$ then $|w| \leq |x|$.

suffix A string w is a suffix of x (denotation $w \sqsupset x$) if $x = yw$ for some string $y \in \Sigma^*$. If $w \sqsupset x$ then $|w| \leq |x|$.

Example 1 `ab` $\sqsubset$ `abcca` *and* `cca` $\sqsupset$ `abcca`.

Lemma 1 *Suppose that x, y, and z are strings such that $x \sqsupset z$ and $y \sqsupset z$. If $|x| \leq |y|$ then $x \sqsupset y$. If $|x| \geq |y|$ then $y \sqsupset x$. If $|x| = |y|$ then $x = y$.*

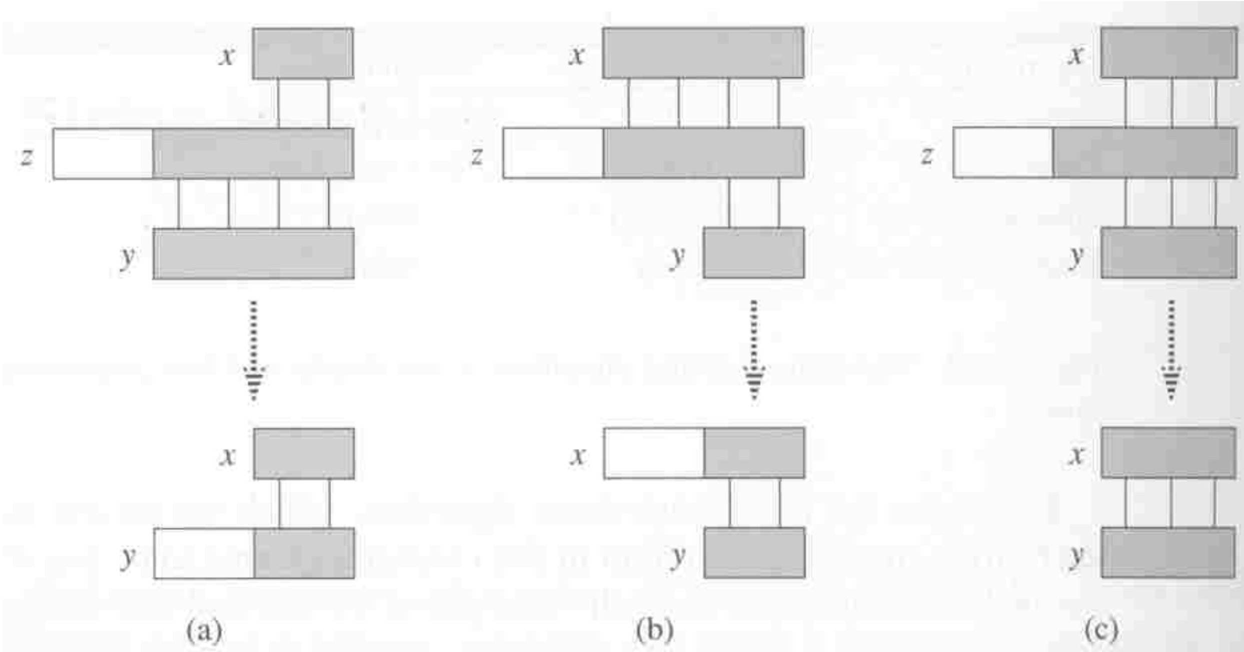


Figure 1: A graphical proof of Lemma 1

2. A naive approach

The following algorithm looks for all occurrences of a pattern $P[1..m]$ in the string $T[1..n]$ and reports all s for which there is a match, i.e.

$$P[1 \ldots m] = T[s+1 \ldots s+m]$$

Algorithm 1 NAIVE-STRING-MATCHER(T, P);

$n = |T|$
$m = |P|$
for $s = 0$ to $n - m$ **do**
  if $(P[1 \ldots m] = T[s+1 \ldots s+m])$ **then**
    print "Pattern occurs with shift" s

The running time of this algorithm is $\Theta((n - m + 1)m)$.

3. The Rabin-Karp algorithm

We consider each character of Σ as a digit in radix-d notation, where $d = |\Sigma|$.

Given a pattern $P[1 \ldots m]$, we let p denote its corresponding decimal value, which can be computed in $\Theta(m)$ time using Horner's rule:

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

Similarly, denote by t_s the decimal value of the length-m substring $T[s+1 \ldots s+m]$, for $s = 1, 2, \ldots, n-m$.

Clearly, $t_s = p$ if and only if $T[s+1 \ldots s+m] = P[1 \ldots m]$. The value t_0 can be computed in time $\Theta(m)$.

To compute the values $t_1, t_2, \ldots, t_{n-m}$ in time $\Theta(n-m)$, note that

$$t_{s+1} = d(t_s - d^{m-1}T[s+1]) + T[s+m+1]. \qquad (1)$$

Assuming that d^{m-1} is precomputed, t_{s+1} can be computed from t_s in a constant time.

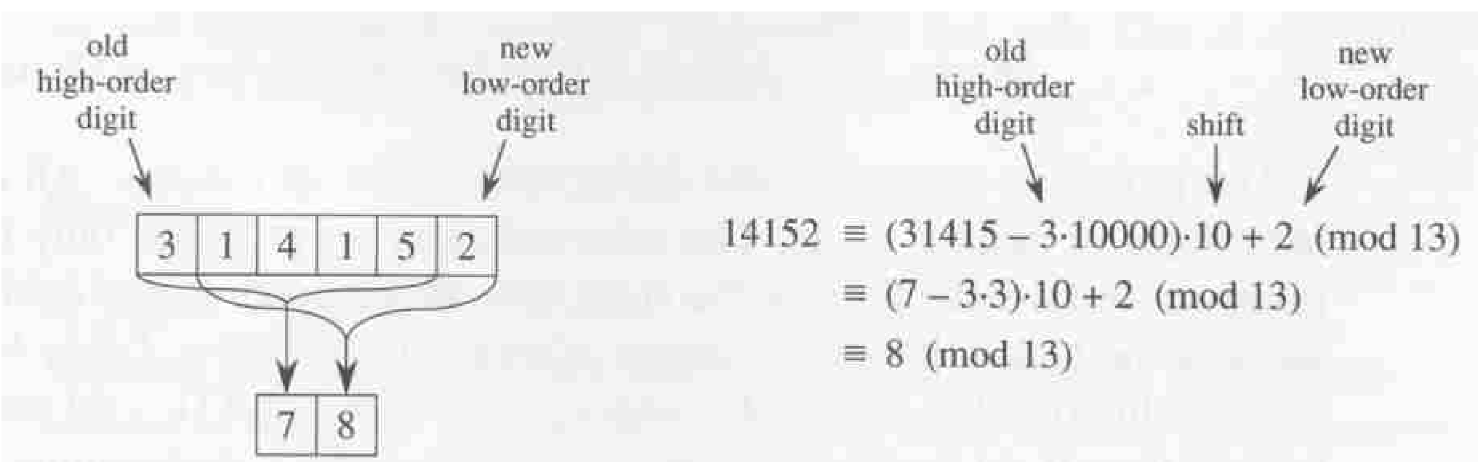


Figure 2: Recomputing the value for a window in a constant time

The only disadvantage of the above method is that the values p and t_s become very large.

To make the approach practical, we consider these numbers modulo q, where q is maximum number such that qd fits within one computer word. Then (1) becomes

$$t_{s+1} = (d(t_s - h \cdot T[s+1]) + T[s+m+1]) \bmod q \qquad (2)$$

where $h \equiv d^{m-1} \pmod q$.

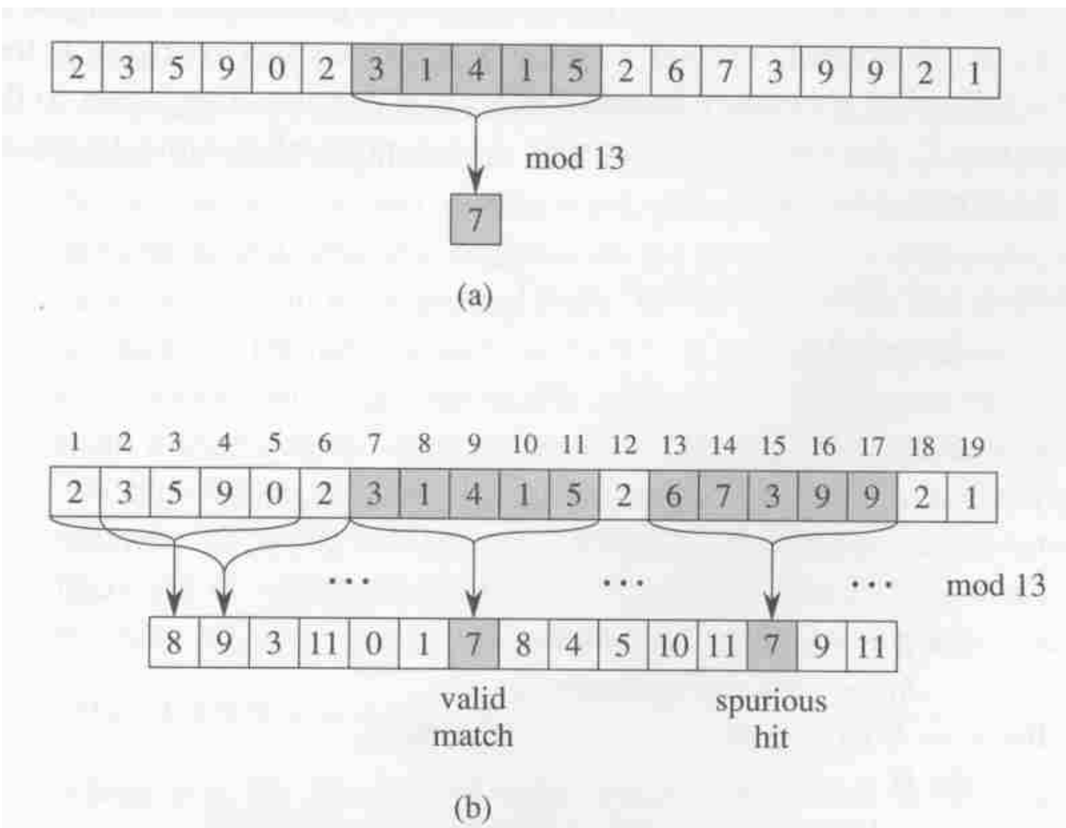


Figure 3: The Rabin-Karp algorithm

Now if $t_s \not\equiv p \pmod d$ then $t_s \neq p$. If $t_s \equiv p \pmod q$ we have a <u>spurious hit</u>. In this case the strings $P[1 \ldots m]$ and $T[s+1 \ldots s+m]$ have to be compared character-by-character as in the naive approach.

Algorithm 2 Rabin-Karp-Matcher(T, P, d, q);

$n := |T|$
$m := |P|$
$h := d^{m-1} \bmod q$
$p := 0$
$t_0 := 0$
for $i = 1$ **to** m **do** //preprocessing
$p := (dp + P[i]) \bmod q$
$t_0 = (dt_0 + T[i]) \bmod q$
for $s = 0$ **to** $n - m$ **do** //matching
if $(p = t_s)$ **then**
if $(P[1 \ldots m] = T[s + 1 \ldots s + m])$ **then**
print "pattern occurs with shift" s
if $(s < n - m)$ **then**
$t_{s+1} := (d(t_s - T[s + 1]h) + T[s + m + 1]) \bmod q$

The preprocessing time is $\Theta(m)$.
The matching time is $\Theta((n - m + 1)m)$ in the worth case.

In many application a few valid shifts are expected. Then the Rabin-Karp algorithm runs significantly faster than the naive one.

4. String matching with finite automata

A finite automaton M is a 5-tuple $(Q, q_0, A, \Sigma, \delta)$, where

- Q is a finite set of states
- q_0 is the start state
- $A \subseteq Q$ is a set of accepted states
- Σ is a finite input alphabet
- δ is a function $Q \times \Sigma \mapsto Q$, called the transition function of M.

If the automaton is in state q and reads a symbol a, it moves to state $\delta(q, a)$. If $\delta(q, a) \in A$, the string ending with a is called accepted.

Example 2 *The following automaton accepts those strings in the alphabet $\Sigma = \{a, b\}$, which end with an odd number of a's.*

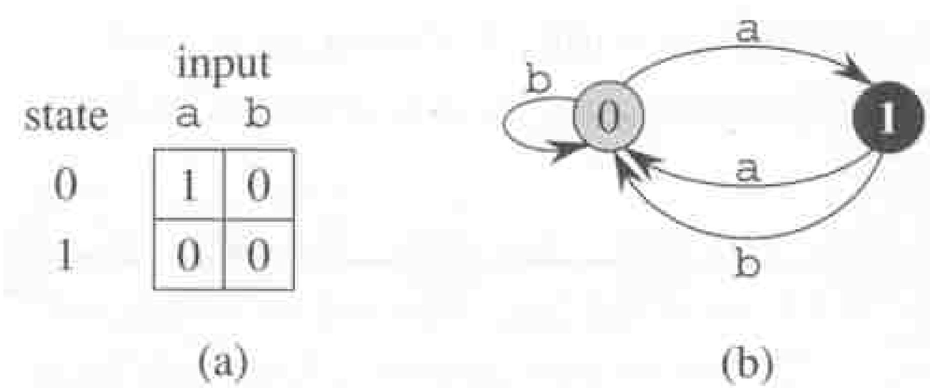


Figure 4: A simple two-state automaton

Given a pattern string $P[1 \ldots m]$, we define $P_k = P[1 \ldots k]$ and introduce the suffix function $\sigma : \Sigma^* \mapsto \{0, 1, \ldots, m\}$ of P as

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

Example 3 *If $P = ab$, we have*

$$\sigma(ccaca) = 1, \qquad \sigma(ccab) = 2, \qquad \sigma(\epsilon) = 0.$$

In general, $\sigma(x) = m$ for $|P| = m$ if and only if $P \sqsupset x$, and if $x \sqsupset y$ then $\sigma(x) \leq \sigma(y)$.

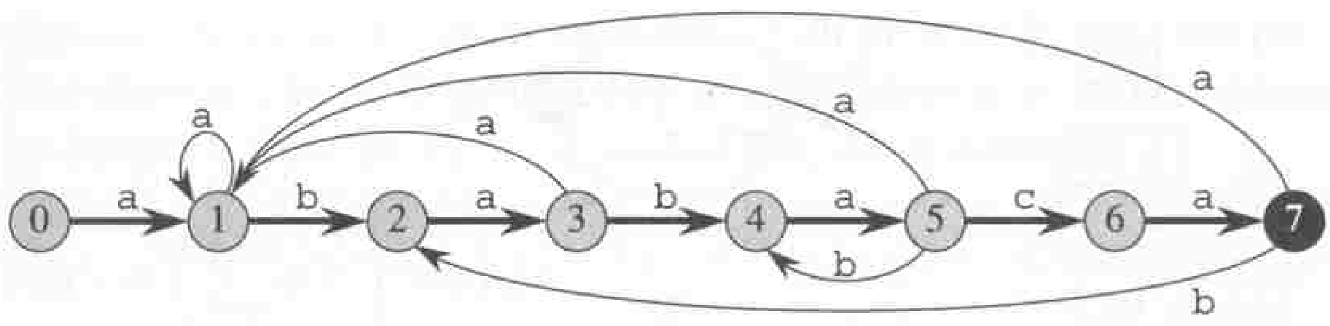
We define the string-matching automaton corresponding to a given pattern $P[1 \ldots m]$ as follows:

- Set $Q = \{0, 1, 2, \ldots, m\}$ and $q_0 = 0$.
- For any $q \in Q$ and $a \in \Sigma$ set
$$\delta(q, a) = \sigma(P_q a). \tag{3}$$

Algorithm 3 Finite-Automaton-Matcher(T, δ, m);

1. $n := |T|$
2. $q := 0$
3. **for** $i = 1$ **to** n **do**
4. $\quad q := \delta(q, T[i])$
5. $\quad$ **if** $(q = m)$ **then**
6. $\qquad$ print "pattern occurs with shift" $i - m$

The running time is $\Theta(n)$ + preprocessing for constructing $\delta()$.



(a)

state	input 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	

(b)

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

(c)

Figure 5: The string-matching automaton for $P = ababaca$

The transition table is constructed so that

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

The machine is designed so that after scanning the first i characters of the string T it is in the state $q = \sigma(T_i)$.

Computing the transition function $\delta()$

$\underline{\textbf{Algorithm } 4}$ Compute-Transition-Function(P, Σ);

1. $m := |P|$
2. **for** $q = 0$ **to** m
3. **for** each $a \in \Sigma$
4. $k := \min(m, q+1)$
5. **while** $(P_k \not\sqsupset P_q a)$
6. $k = k - 1$
7. $\delta(q, a) := k$
8. **return** δ

This algorithm computes $\delta(q, a)$ according to its definition (3)

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

The running time of this method is $\Theta(m^3|\Sigma|)$, however, there exist faster implementations with the running time $\Theta(m|\Sigma|)$.

Therefore, the search for P can be done with $\Theta(m|\Sigma|)$ preprocessing time and $\Theta(n)$ matching time.

To prove the correctness of the above algorithm we will need two lemmas.

Lemma 2 *For any string $x \in \Sigma^*$ and character $a \in \Sigma$, we have $\sigma(xa) \leq \sigma(x) + 1$.*

Proof.

Let $r = \sigma(xa)$. If $r = 0$, then $r = \sigma(xa) = 0 \leq \sigma(x) + 1$ is trivially satisfied, since $0 \leq \sigma(x)$.

If $r > 0$, then $P_r \sqsupset xa \Rightarrow P_{r-1} \sqsupset x$
$\Rightarrow r - 1 \leq \sigma(x)$, and the lemma follows. $\square$

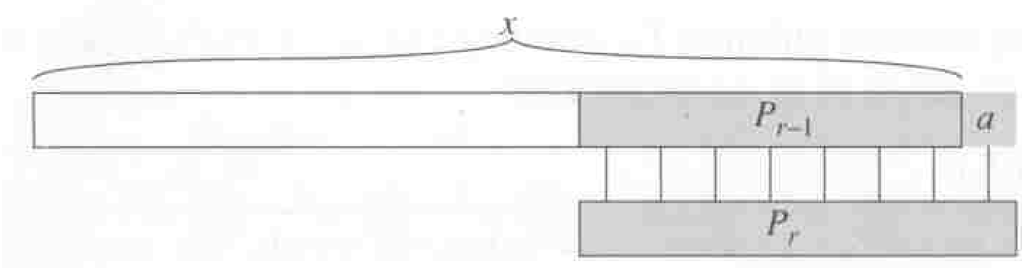


Figure 6: An illustration for the proof of Lemma 2

Lemma 3 *For any $x \in \Sigma^*$ and $a \in \Sigma$, if $q = \sigma(x)$ then $\sigma(xa) = \sigma(P_q a)$.*

Proof.

By the definition of σ, if $q = \sigma(x)$ then $P_q \sqsupset x$.

By Lemma 2, for $r = \sigma(xa)$ we have $r \leq q + 1$.

Since $P_q a \sqsupset xa$, $P_r \sqsupset xa$, $|P_r| \leq |P_q a| \Rightarrow P_r \sqsupset P_q a$ (Lemma 1). Therefore, $r \leq \sigma(P_q a)$, i.e. $\sigma(xa) \leq \sigma(P_q a)$.

On the other hand, $P_q a \sqsupset xa \Rightarrow \sigma(P_q a) \leq \sigma(xa)$. $\square$

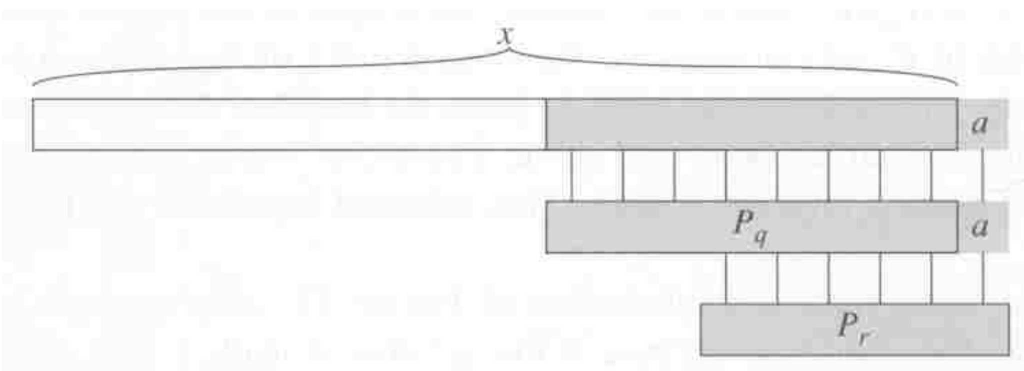


Figure 7: An illustration for the proof of Lemma 3

Theorem 1 *If $\phi(T)$ is the final-state function of a string-matching automaton for a fixed pattern P and given text T, then*

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

Proof. We use induction on i. For $i = 0$ we have $T_0 = \epsilon$, so $\phi(T_0) = \sigma(T_0) = 0$, and the theorem is true.

Assuming $\phi(T_i) = \sigma(T_i)$, we show $\phi(T_{i+1}) = \sigma(T_{i+1})$.

For this denote $q = \phi(T_i)$ and $a = T[i+1]$. One has

$$\begin{aligned}
\phi(T_{i+1}) &= \phi(T_i a) && \text{(since } T_{i+1} = T_i a\text{)} \\
&= \delta(\phi(T_i), a) && \text{(definition of } \phi\text{)} \\
&= \delta(q, a) && \text{(since } \phi(T_i) = q\text{)} \\
&= \sigma(P_q a) && \text{(definition (3) of } \delta\text{)} \\
&= \sigma(T_i a) && \text{(Lemma 3 for } x = T_i \text{ and induction} \\
& && \text{ here } q = \phi(T_i) = \sigma(T_i)\text{)} \\
&= \sigma(T_{i+1}) && \text{(since } T_i a = T_{i+1}\text{)}
\end{aligned}$$

The above theorem implies that if the automaton M enters state q in line 4 of the algorithm, then q is the largest value such that $P_q \sqsupset T_i$.

Thus, $q = m$ in line 5 iff an occurrence of P is found.