String Matching

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

1. Terminology

Let $\Sigma^*$ denote the set of all strings a finite alphabet $\Sigma$.

**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 $w \sqsubseteq x$) if $x = wy$ for some string $y \in \Sigma^*$. If $w \sqsubseteq x$ then $|w| \leq |x|$.

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

**Example 1** $ab \sqsubseteq abcca$ and $cca \sqsupseteq abcca$.

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

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 $\Sigma$ 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

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

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](image)

$$14152 \equiv (31415 - 3 \cdot 10000) \cdot 10 + 2 \pmod{13}$$

$$\equiv (7 - 3 \cdot 3) \cdot 10 + 2 \pmod{13}$$

$$\equiv 8 \pmod{13}$$
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

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

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

Figure 3: The Rabin-Karp algorithm

Now if $t_s \not\equiv p \pmod d$ then $t_s \not\equiv p$. If $t_s \equiv p \pmod q$ we have a spiritous hit. 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\((T, P, d, q)\);

1. \(n := |T|\)
2. \(m := |P|\)
3. \(h := d^{m-1} \mod q\)
4. \(p := 0\)
5. \(t_0 := 0\)
6. for \(i = 1\) to \(m\) do //preprocessing
   7. \(p := (dp + P[i]) \mod q\)
   8. \(t_0 := (dt_0 + T[i]) \mod q\)
9. for \(s = 0\) to \(n - m\) do //matching
   10. if \((p = t_s)\) then
      11.   if \((P[1 \ldots m] = T[s + 1 \ldots s + m])\) then
           12.      print “pattern occurs with shift” \(s\)
      13.     if \((s < n - m)\) then
            14.       \(t_{s+1} := (d(t_s - T[s + 1]h) + T[s + m + 1]) \mod q\)

The preprocessing time is \(\Theta(m)\).
The matching time is \(\Theta((n - m + 1)m)\) in the worst 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
- \( \Sigma \) is a finite input alphabet
- \( \delta \) is a function \( Q \times \Sigma \rightarrow Q \), 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](image)
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
\[ \sigma(ccaca) = 1, \quad \sigma(ccab) = 2, \quad \sigma(\epsilon) = 0. \]

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
  \[ \delta(q, a) = \sigma(P_qa). \]  \hspace{1cm} (3)

**Algorithm 3** Finite-Automaton-Matcher$(T, \delta, m)$;

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

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

7
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 $\delta()$

**Algorithm 4** Compute-Transition-Function($P, \Sigma$);

1. $m := |P|$
2. for $q = 0$ to $m$
3. for each $a \in \Sigma$
4. $k := \min(m, q + 1)$
5. while ($P_k \neq P_qa$)
6. $k = k - 1$
7. $\delta(q, a) := k$
8. return $\delta$

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

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

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 $P_r \sqsupseteq xa \Rightarrow P_{r-1} \supseteq x$  
$\Rightarrow r - 1 \leq \sigma(x)$, and the lemma follows. $\square$

![Figure 6: An illustration for the proof of Lemma 2](image-url)
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 $\sigma$, if $q = \sigma(x)$ then $P_q \sqsupseteq x$.

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

Since $P_q a \sqsupseteq xa$, $P_r \sqsupseteq xa$, $|P_r| \leq |P_q a|$ \(\Rightarrow\) $P_r \sqsupseteq P_q a$ (Lemma 1).

Therefore, $r \leq \sigma(P_q a)$, i.e. $\sigma(xa) \leq \sigma(P_q a)$.

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

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

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

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

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

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 $P_q \sqsupseteq T_i$.

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