String matching with finite automata

Many string-matching algorithms build a finite automaton that scans the text string T for all occurrences of the pattern P. This section presents a method for building such an automaton. These string-matching automata are very efficient: they examine each text character *exactly once*, taking constant time per text character. The time used--after the automaton is built--is therefore $\Theta(n)$. The time to build the automaton, however, can be large if Σ is large. Section 34.4 describes a clever way around this problem.

We begin this section with the definition of a finite automaton. We then examine a special string-matching automaton and show how it can be used to find occurrences of a pattern in a text. This discussion includes details on how to simulate the behavior of a string-matching automaton on a given text. Finally, we shall show how to construct the string-matching automaton for a given input pattern.

Finite automata

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

- Q is a finite set of **states**,
- $q_0 \in Q$ is the **start state**,
- $A \subseteq Q$ is a distinguished set of **accepting states**,
- Σ is a finite **input alphabet**,
- δ is a function from $Q \times \Sigma$ into Q, called the **transition function** of M.

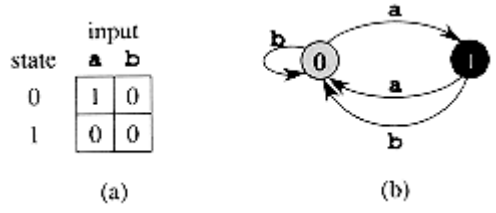


Figure 34.5 A simple two-state finite automaton with state set Q = {0, 1}, start state q_0 = 0, and input alphabet Σ = {a, b}. (a) A tabular representation of the transition function δ. (b) An equivalent state-transition diagram. State 1 is the only accepting state (shown blackened). Directed edges represent transitions. For example, the edge from state 1 to state 0 labeled b indicates $\delta(1, b) = 0$. This automaton accepts those strings that end in an odd number of a's. More precisely, a string x is accepted if and only if x = yz, where y = ε or y ends with

a b, and $z = a^k$, where k is odd. For example, the sequence of states this automaton enters for input abaaa (including the start state) is ⟨0, 1, 0, 1, 0, 1⟩, and so it accepts this input. For input abbaa, the sequence of states is ⟨0, 1, 0, 0, 1, 0⟩, and so it rejects this input.

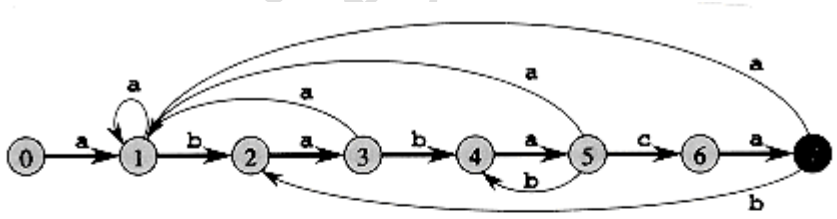
The finite automaton begins in state q_0 and reads the characters of its input string one at a time. If the automaton is in state q and reads input character a, it moves ("makes a transition") from state q to state $\delta(q, a)$. Whenever its current state q is a member of A, the machine M is said to have ***accepted*** the string read so far. An input that is not accepted is said to be ***rejected***. Figure 34.5 illustrates these definitions with a simple two-state automaton.

A finite automaton M induces a function ø, called the ***final-state function***, from Σ^* to Q such that $ø(w)$ is the state M ends up in after scanning the string w. Thus, M accepts a string w if and only if $ø(w) \in A$. The function ø is defined by the recursive relation

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

String-matching automata

There is a string-matching automaton for every pattern P; this automaton must be constructed from the pattern in a preprocessing step before it can be used to search the text string. Figure 34.6 illustrates this construction for the pattern P = ababaca. From now on, we shall assume that P is a given fixed pattern string; for brevity, we shall not indicate the dependence upon P in our notation.



(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 34.6 (a) A state-transition diagram for the string-matching automaton that accepts all strings ending in the string ababaca. State 0 is the start state, and state 7 (shown blackened) is the only accepting state. A directed edge from state i to state j labeled a represents $\delta(i,a) = j$. The right-going edges forming the "spine" of the automaton, shown heavy in the figure, correspond to successful matches between pattern and input characters. The left-going edges correspond to failing matches. Some edges corresponding to failing matches are not shown; by convention, if a state i has no outgoing edge labeled a for some $a \in \Sigma$, then $\delta(i,a) = 0$. (b) The corresponding transition function δ, and the pattern string P = ababaca. The entries corresponding to successful matches between pattern and input characters are shown shaded. (c) The operation of the automaton on the text T = abababacaba. Under each text character T[i] is given the state $\phi(T_i)$ the automaton is in after processing the prefix T_i. One occurrence of the pattern is found, ending in position 9.

In order to specify the string-matching automaton corresponding to a given pattern $P[1 . . m]$, we first define an auxiliary function Σ, called the ***suffix function*** corresponding to P. The function Σ is a mapping from Σ^* to $\{0, 1, . . . , m\}$ such that $\Sigma(x)$ is the length of the longest prefix of P that is a suffix of x:

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

The suffix function Σ is well defined since the empty string $P_0 = \varepsilon$ is a suffix of every string. As examples, for the pattern $P = ab$, we have $\Sigma(\varepsilon) = 0$, $\Sigma(ccaca) = 1$, and $\Sigma(ccab) = 2$. For a pattern P of length m, we have $\Sigma(x) = m$ if and only if $P \sqsupset x$. It follows from the definition of the suffix function that if $x \sqsupset y$, then $\Sigma(x) \leq \Sigma(y)$.

We define the string-matching automaton corresponding to a given pattern P[1 . . m] as follows.

- The state set Q is $\{0, 1, . . . , m\}$. The start state q_0 is state 0, and state m is the only accepting state.

- The transition function δ is defined by the following equation, for any state q and character a:

$$\delta(q, a) = \Sigma(P_q a) .$$

(34.3)

Here is an intuitive rationale for defining $\delta(q, a) = \Sigma(P_q a)$. The machine maintains as an invariant of its operation that

$$\Phi(T_i) = \Sigma(T_i) ;$$

(34.4)

this result is proved as Theorem 34.4 below. In words, this means that after scanning the first i characters of the text string T, the machine is in state $\Phi(T_i) = q$, where $q = \Sigma(T_i)$ is the length of the longest suffix of T_i that is also a prefix of the pattern P. If the next character scanned is $T[i + 1] = a$, then the machine should make a transition to state $\Sigma(T_i + 1) = \Sigma(T_i a)$. The proof of the theorem shows that $\Sigma(T_i a) = \Sigma(P_q a)$. That is, to compute the length of the longest suffix of $T_i a$ that is a prefix of P, we can compute the longest suffix of $P_q a$ that is a prefix of P. At each state, the machine only needs to know the length of the longest prefix of P that is a suffix of what has been read so far. Therefore, setting $\delta(q, a) = \Sigma(P_q a)$ maintains the desired invariant (34.4). This informal argument will be made rigorous shortly.

In the string-matching automaton of Figure 34.6, for example, we have $\delta(5, \texttt{b}) = 4$. This follows from the fact that if the automaton reads a b in state $q = 5$, then $P_q\texttt{b} = \texttt{ababab}$, and the longest prefix of P that is also a suffix of $\texttt{ababab}$ is $P_4 = \texttt{abab}$.

To clarify the operation of a string-matching automaton, we now give a simple, efficient program for simulating the behavior of such an automaton (represented by its transition function δ) in finding occurrences of a pattern P of length m in an input text $T[1 \ldots n]$. As for any string-matching automaton for a pattern of length m, the state set Q is $\{0,1, \ldots, m\}$, the start state is 0, and the only accepting state is state m.

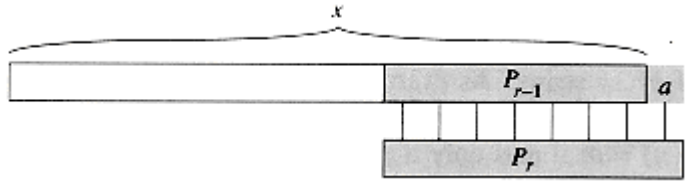


Figure 34.7 An illustration for the proof of Lemma 34.2. The figure shows that $r \le \Sigma(x) + 1$, where $r = \Sigma(xa)$.

```
FINITE-AUTOMATON-MATCHER(T, δ, m)
1  n ← length[T]
2  q ← 0
3  for i ← 1 to n
4        do q ← δ (q, T[i])
5           if q = m
6              then s ← i - m
7                   print "Pattern occurs with shift" s
```

The simple loop structure of FINITE-AUTOMATON-MATCHER implies that its running time on a text string of length n is $O(n)$. This running time, however, does not include the time required to compute the transition function δ. We address this

problem later, after proving that the procedure FINITE-AUTOMATON-MATCHER operates correctly.

Consider the operation of the automaton on an input text $T[1 . . n]$. We shall prove that the automaton is in state $\Sigma(T_j)$ after scanning character $T[i]$. Since $\Sigma(T_i) = m$ if and only if $P \sqsupset T_i$, the machine is in the accepting state m if and only if the pattern P has just been scanned. To prove this result, we make use of the following two lemmas about the suffix function Σ.

Lemma 34.2

For any string x and character a, we have $\Sigma(xa) \leq \Sigma(x) + 1$.

Proof Referring to Figure 34.7, let $r = \Sigma(xa)$. If $r = 0$, then the conclusion $r \leq \Sigma(x) + 1$ is trivially satisfied, by the nonnegativity of $\Sigma(x)$. So assume that $r > 0$. Now, $P_r \sqsupset xa$, by the definition of Σ. Thus, $P_{r-1} \sqsupset x$, by dropping the a from the end of P_r and from the end of xa. Therefore, $r - 1 \leq \Sigma(x)$, since $\Sigma(x)$ is largest k such that $P_k \sqsupset x$.

Lemma 34.3

For any string x and character a, if $q = \Sigma(x)$, then $\Sigma(xa) = \Sigma(P_q a)$.

Proof From the definition of Σ, we have $P_q \sqsupset x$. As Figure 34.8 shows, $P_q a \sqsupset xa$. If we let $r = \Sigma(xa)$, then $r \leq q + 1$ by Lemma 34.2. Since $P_q a \sqsupset xa$, $P_r \sqsupset xa$, and $|Pr| \leq |Pqa|$, Lemma 34.1 implies that $P_r \sqsupset P_q a$. Therefore, $r \leq \Sigma(P_q a)$, that is, $\Sigma(xa) \leq \Sigma(P_q a)$. But we also have $\Sigma(P_q a) \leq \Sigma(xa)$, since $P_q a \sqsupset xa$. Thus, $\Sigma(xa) = \Sigma(P_q a)$.

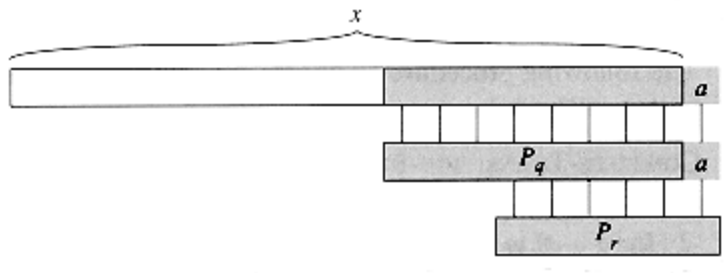


Figure 34.8 An illustration for the proof of Lemma 34.3. The figure shows that $r = \Sigma(P_q a)$, where $q = \Sigma(x)$ and $r = \Sigma(xa)$.

We are now ready to prove our main theorem characterizing the behavior of a string-matching automaton on a given input text. As noted above, this theorem

shows that the automaton is merely keeping track, at each step, of the longest prefix of the pattern that is a suffix of what has been read so far.

Theorem 34.4

If ϕ is the final-state function of a string-matching automaton for a given pattern P and $T[1 \ldots n]$ is an input text for the automaton, then

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

Proof The proof is by induction on i. For $i = 0$, the theorem is trivially true, since $T_0 = \varepsilon$. Thus, $\phi(T_0) = \sigma(T_0) = 0$.

Now, we assume that $\phi(T_i) = \sigma(T_i)$ and prove that $\phi(T_{i+1}) = \sigma(T_{i+1})$.
Let q denote $\phi(T_i)$, and let a denote $T[i + 1]$. Then,

$$\begin{aligned}
\phi(T_{i+1}) &= \phi(T_i a) && \text{(by the definitions of } T_{i+1} \text{ and } a) \\
&= \delta(\phi(T_i), a) && \text{(by the definition of } \phi) \\
&= \delta(q, a) && \text{(by the definition of } q) \\
&= \sigma(P_q a) && \text{(by the definition (34.3) of } \delta) \\
&= \sigma(T_i a) && \text{(by Lemma 34.3 and induction)} \\
&= \sigma(T_{i+1}) && \text{(by the definition of } T_{i+1}) .
\end{aligned}$$

By induction, the theorem is proved.

By Theorem 34.4, if the machine enters state q on line 4, then q is the largest value such that $P_q \sqsupset T_i$. Thus, we have $q = m$ on line 5 if and only if an occurrence of the pattern P has just been scanned. We conclude that FINITE-AUTOMATON-MATCHER operates correctly.

Computing the transition function

The following procedure computes the transition function δ *from a given pattern* P*[1 . .* m*]*.

```
COMPUTE-TRANSITION-FUNCTION(P, Σ)
1 m ← length[P]
2 for q ← 0 to m
3     do for each character a ∈ *
4            do k ← min (m + 1, q + 2)
5               repeat k ← k - 1
```

```
6              until P_k ⊐ P_q a

7          δ(q, a) ← k
8  return δ
```

This procedure computes $\delta(q, a)$ *in a straightforward manner according to its definition. The nested loops beginning on lines 2 and 3 consider all states* q *and characters* a, *and lines 4-7 set* $\delta(q, a)$ to be the largest k such that $P_k \sqsupset P_q a$. The code starts with the largest conceivable value of k, which is $\min(m, q + 1)$, and decreases k until $P_k \sqsupset P_q a$.

The running time of COMPUTE-TRANSITION-FUNCTION is $O(m^3|\Sigma|)$, because the outer loops contribute a factor of $m|\Sigma|$, the inner **repeat** loop can run at most $m + 1$ times, and the test $P_k \sqsupset P_q a$ on line 6 can require comparing up to m characters. Much faster procedures exist; the time required to compute δ from P can be improved to $O(m|\Sigma|)$ by utilizing some cleverly computed information about the pattern P (see Exercise 34.4-6). With this improved procedure for computing δ, the total running time to find all occurrences of a length-m pattern in a length-n text over an alphabet Σ is $O(n + m|\Sigma|)$.