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**.
- These string matching automata are very efficient: they examine each text character ***<u>exactly once</u>***, taking constant time per character.
- The matching time used is therefore <u>$\Theta(n)$</u>**.**
- However, the time to build the automaton(preprocessing) can be large if **Σ** is large.

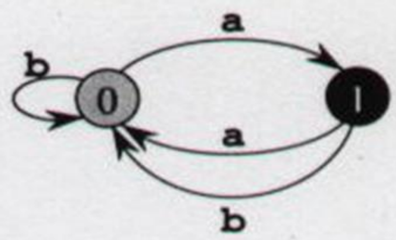
Finite State Machines (FSM)

- FSM(Finite State Machine = Finite Automata) is a computing machine that takes
  - A string as an input
  - Outputs Yes/No answer
    - That is, the machine “accepts” or “rejects” the string

Finite Automata

state	input a	input b
0	1	0
1	0	0

(a)



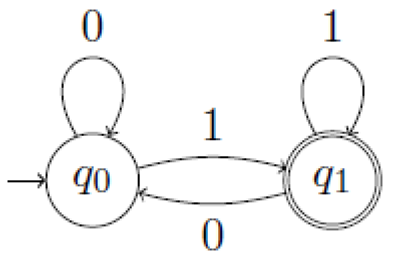
(b)

Figure 34.5 A simple two-state finite automaton with state set $Q = \{0, 1\}$, start state $q_0 = 0$, and input alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. **(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 $\delta(1, \mathtt{b}) = 0$. 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 $y = \varepsilon$ or y ends with a b, and $z = \mathtt{a}^k$, where k is odd. For example, the sequence of states this automaton enters for input abaaa (including the start state) is $\langle 0, 1, 0, 1, 0, 1 \rangle$, and so it accepts this input. For input abbaa, the sequence of states is $\langle 0, 1, 0, 0, 1, 0 \rangle$, and so it rejects this input.

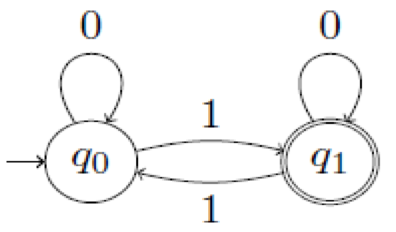
Strategy: Build automaton for pattern, then examine each text character once.

FSM Exercise

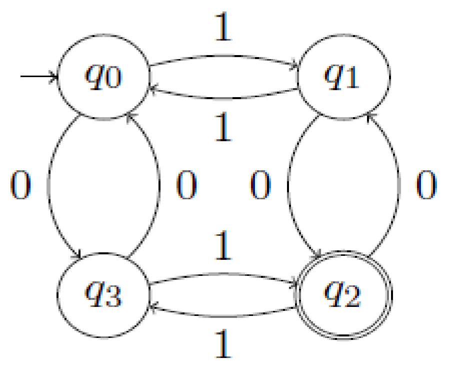
- Automaton accepts strings ending in 1



- Automaton accepts strings having an odd number of 1s



FSM Exercise

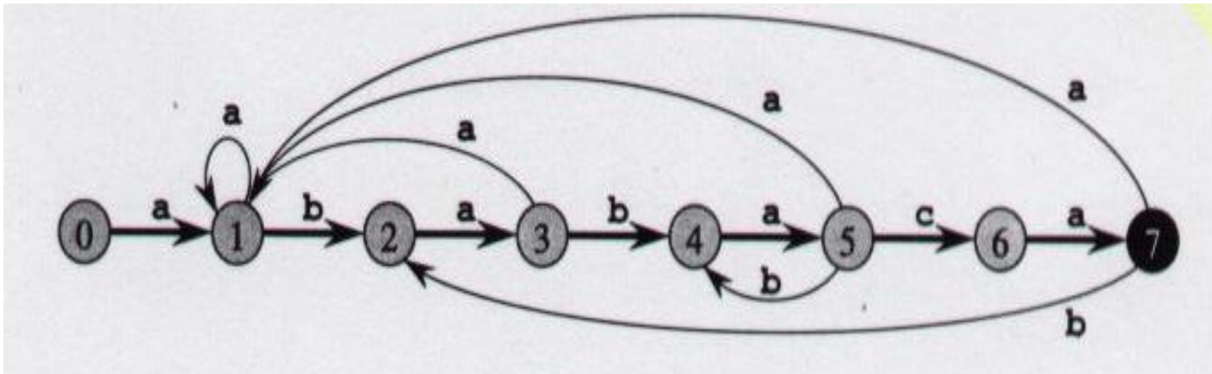


Automaton accepts strings having an odd number of 1s and odd number of 0s

Why Study FSM's

- Useful Algorithm Design Technique
  - String matching problem
  - Lexical Analysis ("tokenization")
  - Control Systems
- Modeling a problem with FSM is
  - Simple
  - Elegant

Example: Pattern = P = **ababaca**



(a)

(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** is the only accepting state. A directed edge from state **i** to state **j** labeled ***a*** represents **δ(i, a) = j**. This 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** $\in$ **Σ**, then **δ(i, a) = 0**.

Example: Pattern = P = **ababaca**

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	

(b)

(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.

Example: Pattern = P = **ababaca**

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)

(c) The operation of the automaton on the text **T = ababababacaba**. Under each text character **T[i]** appears 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**.

String Matching with Finite Automata

- Idea
  - build a finite automaton to scan T for all occurrences of P
  - examine each character exactly once and in constant time
  - matching time $\Theta(n)$, but preprocessing time can be large
- A **finite automaton** M is a 5-tuple $(Q, q0, A, \Sigma, \delta)$
  - Q is a finite set of **states**
  - $q0 \in Q$ 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 **transition function** of M

String Matching with Finite Automata

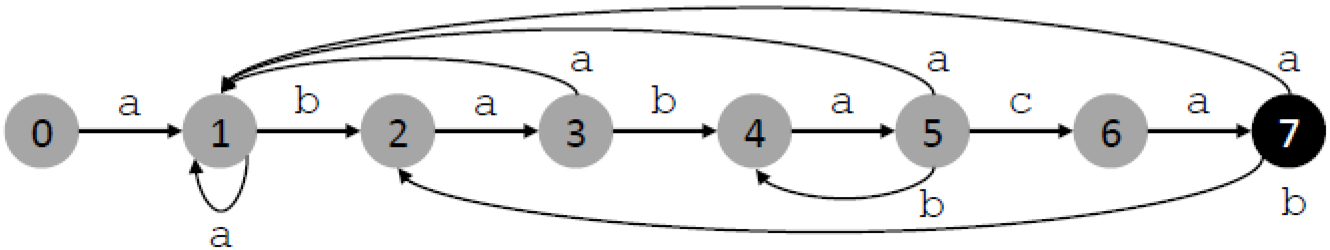
- Finite automaton
  - begins in state q_0, reads one input character a at a time
  - **transitions** from state q into state $\delta(q,a)$
  - **accepts** the string read so far if current state $q \in A$
  - **reject** the string read so far if current state $q \notin A$
- A finite automaton induces a **final-state function** ϕ
  - $\phi:\Sigma^* \rightarrow Q$, such that $q=\phi(w)$ is the state M is in after scanning the string w
  - M accepts a string w if and only if $\phi(w) \in A$
  - recursive definition of ϕ
    $\phi(\epsilon)=q_0$
    $\phi(wa)=\delta(\phi(w), a)$ for $w \in \Sigma^*$, $a \in \Sigma$

String-Matching Automata

- For every pattern $P[1..m]$, we need to construct a string-matching automaton in preprocessing
  - the state set Q is $\{0,1,\ldots,m\}$, where start state q_0 is state 0 and state m is the only accepting state
  - the transition function is defined as $\delta(q, a)=\sigma(P_q a)$ for any state q and character a
- Suffix function σ for a given pattern $P[1..m]$
  - $\sigma:\Sigma\rightarrow\{0,1,\ldots,m\}$ such that $\sigma(x)=\max\{k:Pk\sqsupset x\}$ is the length of the longest prefix of P that is a suffix of x
  - for a pattern P of length m, $\sigma(x)=m$ if and only if $P\sqsupset x$
  - if $x\sqsupset y$, then $\sigma(x)\leq\sigma(y)$

Example

- Assume pattern P= ababaca



  - 8 states and a "spine" of forward transitions
  - $\delta(1, \mathrm{a}) = 1$, since $P_1\mathrm{a} = \mathrm{a}\mathbf{a}$ and $\sigma(P_1\mathrm{a}) = 1$
  - $\delta(3, \mathrm{a}) = 1$, since $P_3\mathrm{a} = \mathrm{aba}\mathbf{a}$ and $\sigma(P_3\mathrm{a}) = 1$
  - $\delta(5, \mathrm{a}) = 1$ since $P_5\mathrm{a} = \mathrm{ababa}\mathbf{a}$ and $\sigma(P_5\mathrm{a}) = 1$
  - $\delta(5, \mathrm{b}) = 4$, since $P_5\mathrm{b} = \mathrm{ab}\mathbf{abab}$ and $\sigma(P_5\mathrm{b}) = 4$
  - $\delta(7, \mathrm{a}) = 1$, since $P_7\mathrm{a} = \mathrm{ababaca}\mathbf{a}$ and $\sigma(P_7\mathrm{a}) = 1$
  - $\delta(7, \mathrm{b}) = 2$, since $P_7\mathrm{b} = \mathrm{ababac}\mathbf{ab}$ and $\sigma(P_7\mathrm{b}) = 2$

String-Matching Automata

FINITE-AUTOMATON-MATCHER(T, P, Σ, m)

$n \leftarrow length[T]$
$\delta \leftarrow$ COMPUTE-TRANSITION-FUNCTION(P, Σ)
$q \leftarrow 0$
for $i \leftarrow 1$ **to** n
    do $q \leftarrow \delta(q, T[i])$
        if $q = m$
            then print "Pattern occurs with shift" $i - m$

Matching time on a text of length n is $\Theta(n)$
–simple loop structure with n iterations
–does not account for the time required to compute the transition function δ

Computing the Transition Function δ

COMPUTE-TRANSITION-FUNCTION(P, Σ)

$m \leftarrow length[P]$
for $q \leftarrow 0$ **to** m
  do for each character $a \in \Sigma$
    do $k \leftarrow \min(m+1, q+2)$
      repeat $k \leftarrow k-1$
       until $P_k \sqsupset P_q a$
      $\delta(q, a) \leftarrow k$
return δ

Computing transition function takes time $\underline{O(m^3|\Sigma|)}$

–outer loop : $m|\Sigma|$

–inner loop can run at most m+1 times

–test $P_k \sqsupset P_q a$ can require up to m comparisons

Computing the Transition Function δ

- Much faster procedures for computing the transition function exist. The time required to compute P can be improved to $\underline{O(m|\Sigma|)}$
- Matching time on a text of length n is $\underline{\Theta(n)}$
- This brings the total runtime to:

$$\underline{O(m|\Sigma| + n)}$$

- Not bad if your string is fairly small relative to the text you are searching in.