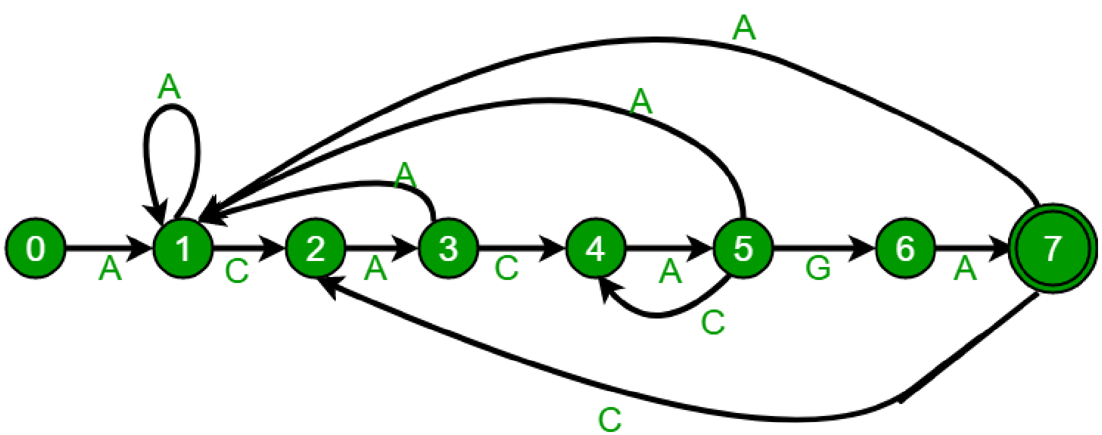
String Matching

String Matching with finite automata

Design and Analysis of Algorithm
Spring 2022

FINITE-STATE AUTOMATON

- A **finite-state machine (FSM)** or **finite-state automaton** (**FSA**, plural: *automata*), **finite automaton**, or simply a **state machine**, is a mathematical model of computation.
- It is an abstract machine that can be in exactly one of a finite number of *states* at any given time.
- The FSM can change from one state to another in response to some inputs; the change from one state to another is called a *transition*



String Matching with finite automata

- These algorithms build a finite automaton that scans the text string T for all occurrences of pattern P.
- Each text character is examined only once
- Time to build the automaton can be large if $\sum$ is large.

Finite Automata

- A *finite automaton M* is a 5-tuple $(Q, q_o, A, \sum, \delta)$, where
  - Q is finite set of *states*,
  - $q_o \in$ Q is the *start state*,
  - $A \subseteq$ Q is a distinguished set of *accepting states*,
  - $\sum$ is finite *input alphabet*,
  - δ is a function from Q x $\sum$ into Q, called the *transition function* of M

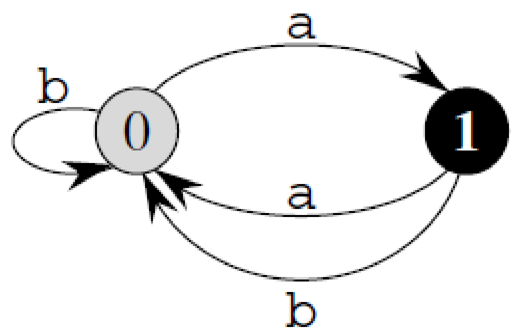
Finite Automata (Cont'd)

- Suppose M is in state q_o.
- It reads char. a, it moves from state q to state $\delta(q,a)$

- Whenever current state $q \in A$, the machine M has *accepted* the string read so far.
- An input that is not accepted is said to be *rejected*

state	input a	input b
0	1	0
1	0	0

(a)



(b)

Figure 32.6 A simple two-state finite automaton with state set $Q = \{0, 1\}$, start state $q_0 = 0$, and input alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$. **(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 $\delta(1, \texttt{b}) = 0$. 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 $z = \texttt{a}^k$, 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.

Final-State Function

- The automaton M has a *final-state function* Φ from $\sum^*$ to Q, such that:
- $\Phi(w)$ is the state, M ends up in, after scanning the string w.
- M accepts string w if and only if $\Phi(w) \in A$.
- It is defined recursively as follows:
  - $\Phi(w) = q_o$ *if* $w = \varepsilon$
  - $\Phi(wa) = \delta(\Phi(w), a)$ for $w \in \sum^*$, a $\in \sum$

String Matching Automata

- Every pattern *P* has finite automaton
- It must be built in the preprocessing step
- In order to do so, we first define a function called *suffix-function* σ, <u>corresponding to *P*</u>
- It is a mapping from ∑* to {0,1,…,m} such that:
- $\sigma(x)$ = length of the longest prefix of *P* that is a suffix of *x*
- $\sigma(x) = max\{k : P_k \sqsupset x\}$

String Matching Automata

- Suffix function is well defined, since $P_o = \varepsilon$ is a suffix of every string.

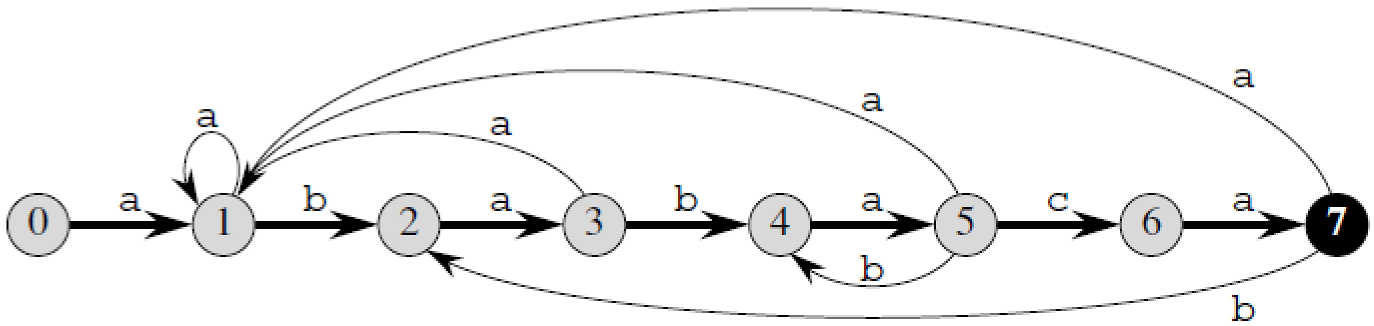
- If $P=ab$, then:
- $\sigma(\varepsilon) = 0$
- $\sigma(x) = 0$
- $\sigma(ccaca) = 1$, $\sigma(ccab) = 2$

String Matching Automata

- $\sigma(x) = m$ *if an only if P is a suffix of x*

- String Matching Automaton corresponding to a given pattern is defined as:

  - State Set Q = { 0, 1, 2, ….. M}
  - Transition function
    - $\delta(q, a) = \sigma(P_q a)$



(a)

P = ababaca

state	input a	input b	input 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)

Fig 32.7

String Matching Automata

- Figure 32.7 (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 $a \in$, 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 $\varphi(Ti\ )$ the automaton is in after processing the prefix Ti . One occurrence of the pattern is found, ending in position 9.

FINITE-AUTOMATON-MATCHER(T, δ, m)

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

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

COMPUTE-TRANSITION-FUNCTION(P, Σ)

```
1  m ← length[P]
2  for q ← 0 to m
3       do for each character a ∈ Σ
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
```

The running time of COMPUTE-TRANSITION-FUNCTION is $O(m^3 |\Sigma|)$.

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 7 can require comparing up to m characters

Reference

- **Introduction to Algorithms**
  - Thomas H. Cormen
  - Chapter # 32