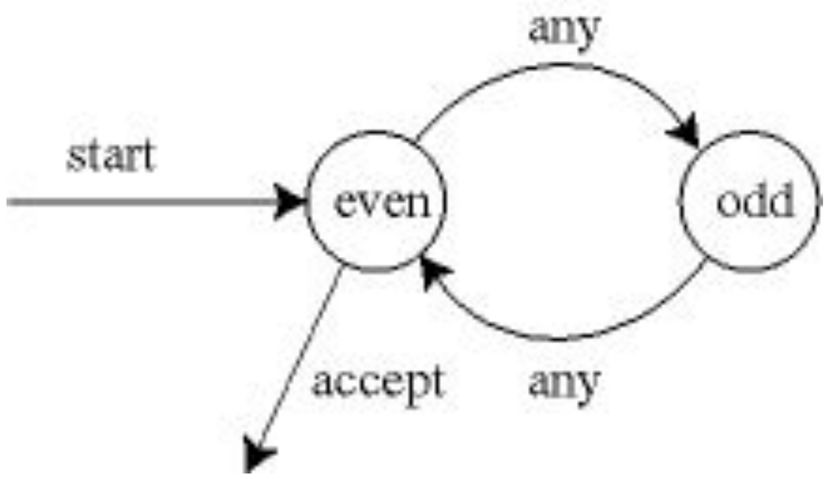
Finite Automata

A finite automaton is a quintuple $(Q, \Sigma, \delta, s, F)$:

- Q: the finite **set of states**
- Σ: the finite **input alphabet**
- δ: the "transition function" from $Q \times \Sigma$ to Q
- $s \in Q$: the start state
- $F \subset Q$: the set of final (accepting) states

How it works

A finite automaton accepts strings in a specific language. It begins in state q_0 and reads characters one at a time from the input string. It makes transitions (φ) based on these characters, and if when it reaches the end of the tape it is in one of the accept states, that string is accepted by the language.



Graphic: Eppstein, David. http://www.ics.uci.edu/~eppstein/161/960222.html

The Suffix Function

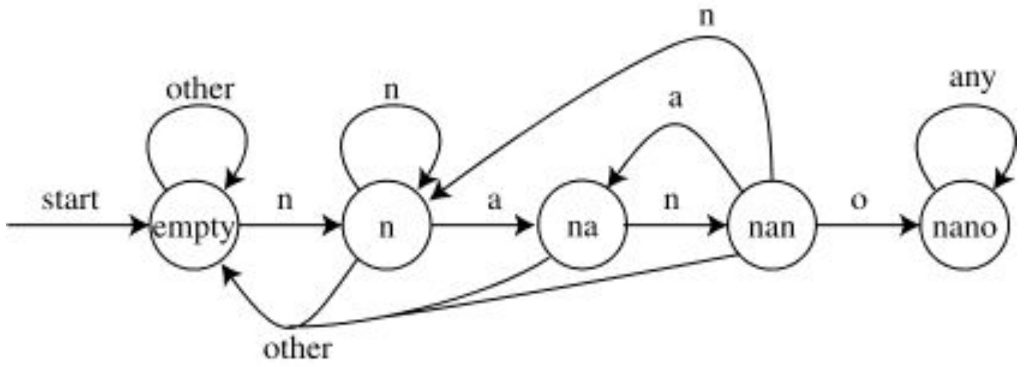
In order to properly search for the string, the program must define a **suffix function (σ)** which checks to see how much of what it is reading matches the search string at any given moment.

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

$$\begin{aligned}
&\mathrm{P} = \mathrm{abaabc}\\
&\mathrm{P}_1 = \mathrm{a}\\
&P_2 = \mathrm{ab}\\
&P_3 = \mathrm{aba}\\
&P_4 = \mathrm{abaa}\\
&\sigma(\mathrm{abbaba}) = \mathrm{aba}
\end{aligned}$$

Graphic: Reif, John. http://www.cs.duke.edu/education/courses/cps130/fall98/lectures/lect14/node31.html

Example: nano



	n	a	o	other
empty:	n	ε	ε	ε
n:	n	na	ε	ε
na:	nan	ε	ε	ε
nan:	n	na	nano	ε
nano:	nano	nano	nano	nano

Graphic & Example: Eppstein, David. http://www.ics.uci.edu/~eppstein/161/960222.html

String-Matching Automata

- For any pattern P of length m, we can define its string matching automata:

  $Q = \{0,\ldots,m\}$ (states)

  $q_0 = 0$ (start state)

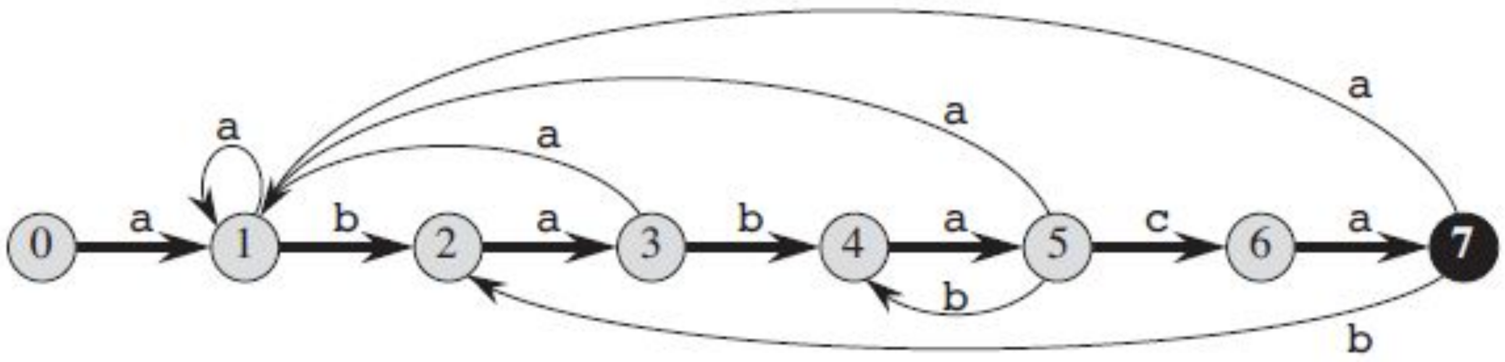
  $F = \{m\}$ (accepting state)

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

The transition function chooses the next state to maintain the invariant:

$$\varphi(T_i) = \sigma(T_i)$$

After scanning in i characters, the state number is the longest prefix of P that is also a suffix of T_i.



(a)

	input			
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)

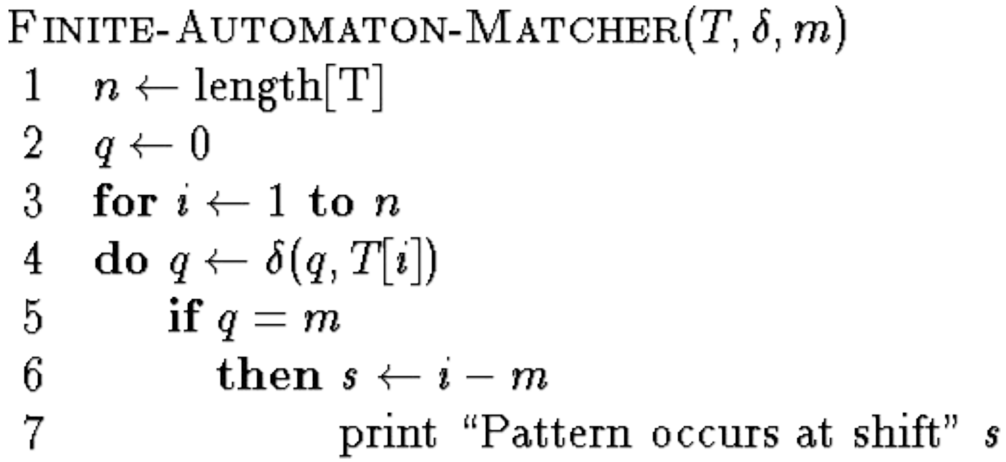
i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

(c)

Finite-Automaton-Matcher

The simple loop structure implies a running time for a string of length *n* is O(n).

However: this is only the running time for the actual string matching. It does not include the time it takes to compute the transition function.



```
FINITE-AUTOMATON-MATCHER($T, \delta, m$)
$n \leftarrow$ length[T]
$q \leftarrow 0$
for $i \leftarrow 1$ to $n$
do $q \leftarrow \delta(q, T[i])$
    if $q = m$
        then $s \leftarrow i - m$
            print "Pattern occurs at shift" $s$
```

Graphic: http://www.cs.duke.edu/education/courses/cps130/fall98/lectures/lect14/node33.html

Computing the Transition Function

```
Compute-Transition-Function (P,Σ)
$m$ □ length[P]
For $q$ □ 0 to $m$
    do for each character $a \in \Sigma$
        do $k$ □ min($m$+1, $q$+2)
            repeat $k$ □ $k$-1
                until $P_k \supset P_q a$
            $\delta(q,a)$ □ $k$
return $\delta$
```

This procedure computes $\delta(q,a)$ according to its definition. The loop on line 2 cycles through all the states, while the nested loop on line 3 cycles through the alphabet. Thus all state-character combinations are accounted for. Lines 4-7 set $\delta(q,a)$ to be the largest k such that $P_k \supset P_q a$.

Running Time of Compute-Transition-Function

Running Time: $O(m^3 |\Sigma|)$

Outer loop: $m\ |\Sigma|$

Inner loop: runs at most $m+1$

$P_k \supset P_q a$: requires up to m comparisons

Improving Running Time

Much faster procedures for computing the transition function exist. The time required to compute P can be improved **to O(m|Σ|)**.

The time it takes to find the string is linear: **O(n)**.

This brings the total runtime to:

O(n + m|Σ|)

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