String Matching with Finite Automata

A finite automaton (FA) consists of a tuple (Q, q_0, A, Σ, δ), where

i) Q is a finite set of states,

ii) $q_0 \in Q$ is the starting state,

iii) $A \subseteq Q$ is the set of accepting states,

iv) Σ is the input alphabet (finite),

v) $\delta: Q \times \Sigma \to Q$ is the transition function.

Example:

$Q = \{0,1,2,3,4,5\}$,

$q_0 = 0$,

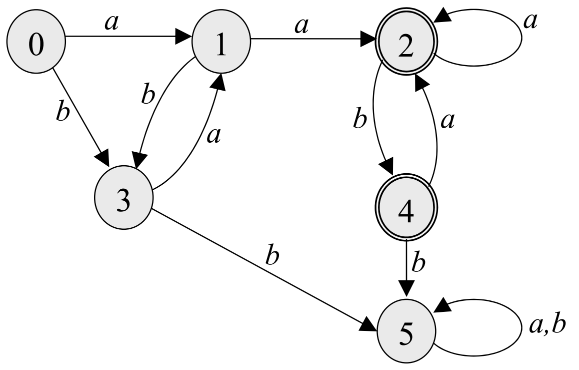
$A = \{2,4\}$,

$\Sigma = \{a,b\}$,

δ(*q*,σ)

q \ σ	***a***	***b***
0	1	3
1	2	3
2	2	4
3	1	5
4	2	5
5	5	5

We can represent our FA graphically like this:



State 0 = starting state.

Double line boundary = accepting state

Given any input string x over the alphabet Σ, a FA

- starts in start q_0, and
- reads the string x, character by character, changing state after each character read.

When the FA is in state q and reads character σ, it enters state $\delta(q,\sigma)$.

The finite automaton

- accepts the string x if it ends up in an accepting state, and
- rejects x if it does not end up in an accepting state.

Example: Suppose our FA reads the string x = *bababaabaaabbaab.*

symbol read		*b*	*a*	*b*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*	*b*	*b*	*a*	*a*	*b*
new state	0	3	1	3	1	3	1	2	4	2	2	2	4	5	5	5	5

The final state (5) is not an accepting state. So the FA rejects string x.

Our FA accepts exactly those strings that contain two consecutive *a*s but do not contain two consecutive *b*s.

We can think of the states of our finite automaton as recording certain information about the characters read so far?

State	Have two consecutive *b*s been found?	Have two consecutive *a*s been found?	Last character read
0	no	no	none
1	no	no	*a*
2	no	yes	*a*
3	no	no	*b*
4	no	yes	*b*
5	yes	---	---

We can construct an FA to search for a pattern $P = p_1 p_2 \ldots p_m$ in a text $T = t_1 t_2 \ldots t_n$.

- The FA will have m+1 states, which we number 0, 1, ..., m.
- State 0 will be the starting state, and state m will be the only accepting state.
- In general, the FA will be in state k if k characters of the pattern have been matched.
    - In other words, the FA is in state k if the k most recently read characters of the text match the first k pattern characters.

last char-acter read

$$\begin{array}{llll|llllll|ll} \text{T:} & t_1 & t_2 & \ldots & t_j & t_{j+1} & t_{j+2} & \ldots & t_{j+k-2} & t_{j+k-1} & t_{j+k} & \ldots \\ \text{P:} & & & & p_1 & p_2 & p_3 & \ldots & p_{k-1} & p_k & p_{k+1} & \ldots \end{array}$$

- If the next text character t_{j+k} equals p_{k+1}, we have matched k+1 characters, and the FA enters state k+1.
    - In other words, $\delta(k, p_{k+1}) = k+1$.
- If the next text character t_{j+k} differs from p_{k+1}, then the FA enters a state 0, 1, 2, ..., or k, depending on how many initial pattern characters match text characters ending with t_{j+k}.
    - We shift the pattern right till we obtain a match, or exhaust the pattern.

$$\begin{array}{lllll|llllll|l} \text{T:} & t_1 & t_2 & \ldots & t_j & t_{j+1} & t_{j+2} & \ldots & t_{j+k-2} & t_{j+k-1} & t_{j+k} & \ldots \\ \text{P:} & & & & & p_1 & p_2 & \ldots & p_{k-2} & p_{k-1} & p_k & \ldots \end{array}$$

If match, enter state k; else continue

$$\begin{array}{llllll|lllll|l} \text{T:} & t_1 & t_2 & \ldots & t_j & t_{j+1} & t_{j+2} & \ldots & t_{j+k-2} & t_{j+k-1} & t_{j+k} & \ldots \\ \text{P:} & & & & & & p_1 & \ldots & p_{k-3} & p_{k-2} & p_{k-1} & \ldots \end{array}$$

If match, enter state k–1; else continue

We continue like this till we reach

$$\begin{array}{llllllllll|l|l} \text{T:} & t_1 & t_2 & \ldots & t_j & t_{j+1} & t_{j+2} & \ldots & t_{j+k-2} & t_{j+k-1} & t_{j+k} & \ldots \\ \text{P:} & & & & & & & & & & p_1 & \ldots \end{array}$$

if match, enter state 1; else state 0

- If $\sigma = t_{j+k} \neq p_{k+1}$, then $\delta(k, \sigma)$ = largest integer d such that

$$t_{j+k-(d-1)} \ldots t_{j+k-2}\, t_{j+k-1}\, t_{j+k} = p_1 \ldots p_{d-2}\, p_{d-1}\, p_d$$

- This makes it appear that $\delta(k, \sigma)$ depends on the subject as well as the pattern, but recall that to be in state k to begin with we must have a match

$$\begin{array}{llll|llllll|ll} \text{T:} & t_1 & t_2 & \ldots & t_j & t_{j+1} & t_{j+2} & \ldots & t_{j+k-2} & t_{j+k-1} & t_{j+k} & \ldots \\ \text{P:} & & & & p_1 & p_2 & p_3 & \ldots & p_{k-1} & p_k & p_{k+1} & \ldots \end{array}$$

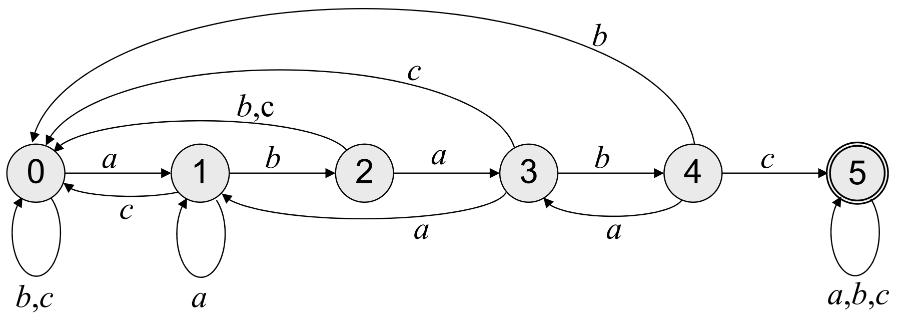
meaning $t_{j+i} = p_{i+1}$ for i = 1,2,..., k–1. So we can say that $\delta(k, \sigma)$ = largest integer d such that

$$p_{k-d+2} \ldots p_{k-1}\, p_k\, \sigma = p_1 \ldots p_{d-2}\, p_{d-1}\, p_d$$

Thus $\delta(k, \sigma)$ depends only on the pattern, k, and σ.

- If the FA reaches state m, a match has been found, and the FA remains in state m. (In practice, the computation could stop at this point.)

Example: A *finite automaton* to match pattern *ababc* over alphabet $\Sigma = \{a,b,c\}$.



Matching pattern *ababc* in text *caabaabcabababccb.*

char read		***c***	***a***	***a***	***b***	***a***	***a***	***b***	***c***	***a***	***b***	***a***	***b***	***a***	***b***	***c***	***c***	***b***
new state	0	0	1	1	2	3	1	2	0	1	2	3	4	3	4	5	5	5

Match succeeds – can stop here

Recall we compute the transition function by

$\boldsymbol{\delta(m, \sigma) = m}$ for all σ.

$\boldsymbol{\delta(k, p_k) = k+1}$.

$\boldsymbol{\delta(k, \sigma) = d}$ if $\sigma \neq p_k$, where $d \leq k$ is maximal subject to a match

$$\begin{array}{cccccc} p_{k-d+2} & p_{k-d+3} & \cdots & p_{k-1} & p_k & \sigma \\ \parallel & \parallel & \parallel & \parallel & \parallel & \parallel \\ p_1 & p_2 & \cdots & p_{d-2} & p_{d-1} & p_d \end{array}$$

Straight from the definition, we can compute

- $\delta(i,x)$ in $O(m^2)$ time,
- all $(m+1)|\Sigma|$ entries of δ in $O(m^3)$ time, if we treat $|\Sigma|$ as constant. (It may be a fairly large constant, e.g., 256.)

This isn't too bad, since typically the pattern is fairly short compared to the text.

But much more efficient constructions are known. (They are fairly simple, and reduce the time to O(m) if $|\Sigma|$ is treated as constant.)

Once we have constructed a finite automaton for the pattern, searching a text $t_1t_2....t_n$ for the pattern works wonderfully.

- Search time is $O(n)$.
- Each character in the text is examined just once, in sequential order.

```
state = 0;
for ( i = 1, 2, ...., n )
        state = δ( state, t_i );
        if ( state == m )
                Match succeeds in position i–m+1; stop;
 Match fails;
```