String Matching II

Chapter 32

Lecture Overview

- String matching with Automata
  - DFAs
  - String-matching automata
  - The transition function
- Knuth-Morris-Pratt
  - The prefix function

(Deterministic) Finite Automata

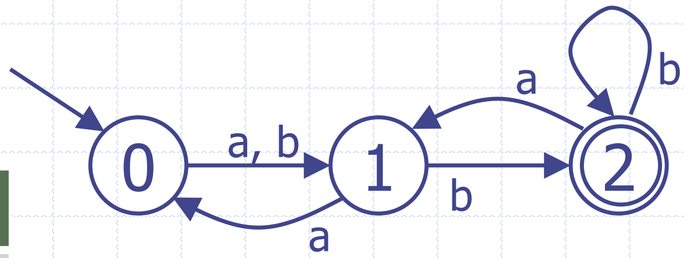
A (deterministic) finite automaton is a mathematical machine that operates on a string. Formally, it is a 5-tuple $M = (Q, q_0, A, \Sigma, \delta)$ where

- Q is a finite set of states,
- $q_0 \in Q$ is the start state,
- $A \subseteq Q$ is the set of accepting states,
- Σ is a finite input alphabet, and
- $\delta: Q \times \Sigma \to Q$ is the transition function.

(Deterministic) Finite Automata

- $Q = \{0, 1, 2\}$,
- $q_0 = 0$,
- $A = \{2\}$,
- $\Sigma = \{a, b\}$

δ	0	1	2
a	1	0	1
b	1	2	2



The automaton begins in state q_0 and reads the characters of its input string one at a time. If it is in state q and reads input character a, it moves ("makes a transition") to state δ(q, a). Whenever its current state q is in A, the machine is said to have accepted the string read so far. We are particularly interested in the state that the machine is in when it has read the entire input string; the machine accepts the string if it is in an accepting state, otherwise it rejects the string.

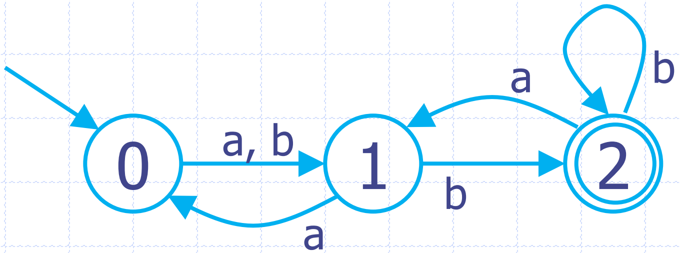
Final State Function

Given a DFA $M = (Q, q_0, A, \Sigma, \delta)$, we can define its final state function $\Phi_M: \Sigma^* \to Q$ recursively as:

$$\Phi_M(\varepsilon) = q_0$$

$$\Phi_M(wa) = \delta(\Phi_M(w), a) \qquad \text{for } w \in \Sigma^* \text{ and } a \in \Sigma$$

$\Phi_M(w)$ is simply the state that machine M ends up in when its input string is w.

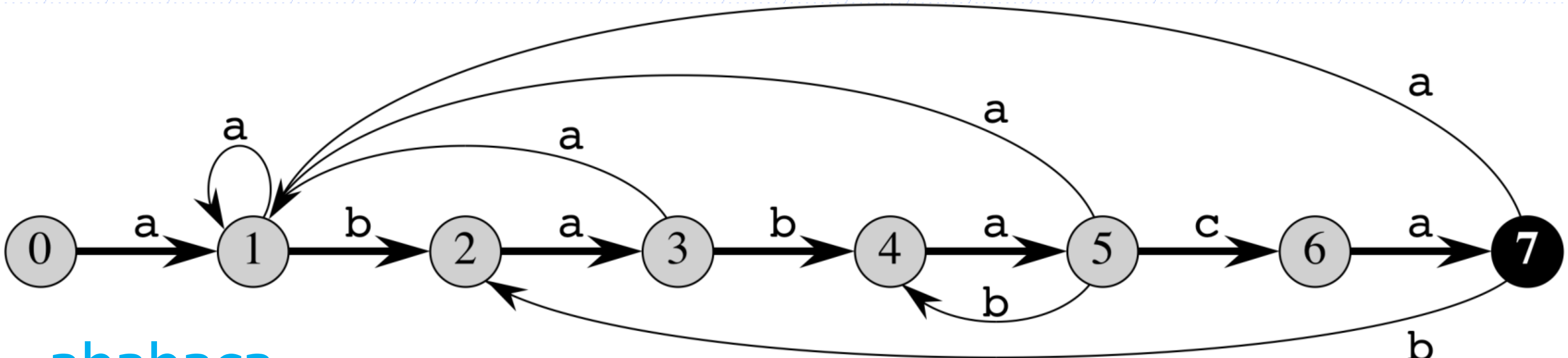


$\Phi(\varepsilon) = 0$
$\Phi(a) = 1$
$\Phi(aa) = 0$
$\Phi(aab) =1$
$\Phi(aaba) = 0$

$\Phi(babb) = 2$
$\Phi(abab) = 2$
$\Phi(abba) = 1$
$\Phi(aba) =1$
$\Phi(abab) = 2$

String-matching Automata

There is a string-matching automaton M_P for every pattern P. It will be constructed from the pattern in a preprocessing step and then be used to search the text string T. We want M_P to have the property that $\Phi_M(w) \in A$ iff $P \sqsupset w$. Then we can find the valid shifts by finding the accepting states in Φ_M for text T and its prefixes.



ababaca

Suffix Function

Given a pattern P, we define the suffix function $\sigma: \Sigma^* \rightarrow \mathbf{Z}_{m+1}$ such that $\sigma(x)$ is the length of the longest prefix of P that is a suffix of x:

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

This is well-defined since $P_0 = \varepsilon$ is a suffix of x.

For pattern aba:

$\sigma(\text{baa}) = 1$
$\sigma(\text{baab}) = 2$
$\sigma(\text{baba}) = 3$
$\sigma(\text{babb}) = 0$

String-Matching Automaton

Given a pattern P[1..m], $M_p = (Q, q_0, A, \Sigma, \delta)$, where:

- $Q = \{0, 1, \ldots m\}$
- $q_0 = 0$
- $A = m$
- $\delta(q, a) = \sigma(P_q a)$

Lemma. This automaton maintains the invariant

$$\Phi(T_i) = \sigma(T_i)$$

Proof. Suppose $T_{i+1} = T_i a$. Then because $\Phi(T_i) = \sigma(T_i)$,
$\Phi(T_{i+1}) = \delta(\Phi(T_i)\ , a) = \sigma(P_{\Phi(Ti)}\ a) = \sigma(T_i a) = \sigma(T_{i+1})$.
Also, $\Phi(\varepsilon) = 0 = \sigma(\varepsilon)$. ■

Finite Automaton Matcher

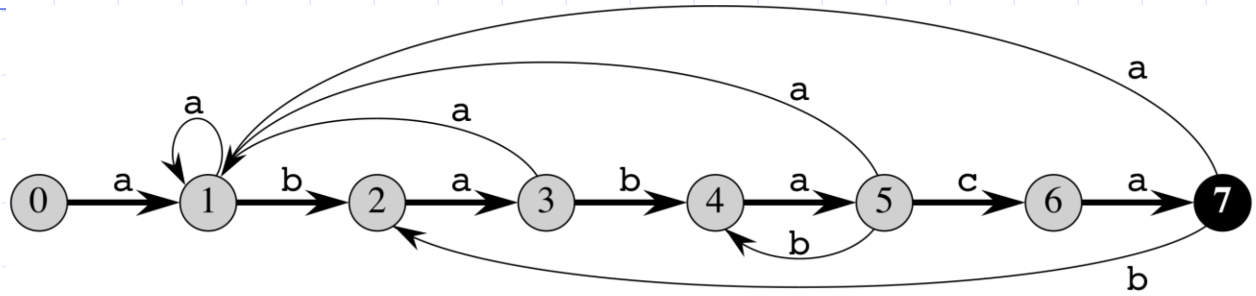
FINITE-AUTOMATON-MATCHER(T, δ, m)

1. n = length(T) — O(1)
2. q = 0 — O(1)
3. **for** i = 1 to n — n iterations
4. q = δ(q, T[i]) — O(1)
5. **if** q = m — O(1)
6. **output** i – m — O(1)

matching time: O(n)

preprocessing time: ?

Finite Automaton Matcher



(a)

ababaca

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)

Transition Function

```
COMPUTE-TRANSITION-FUNCTION(P, Σ)
m = length(P)                            O(1)
for q = 0 to m                           O(m) iterations
    for each character a ∈ Σ               O(|Σ|) iterations
        k = min(m+1, q+2)                     O(1)
        repeat k = k – 1                      O(m) iterations
            until P_k ⊐ P_qa                      O(m)
        δ(q, a) = k                           O(1)
return δ                                 O(1)
```

total time: $O(m^3|\Sigma|)$

String Matching with Automata

- So string matching with automata takes $O(m^3|\Sigma|)$ preprocessing and $O(n)$ matching time, for a total of $O(n + m^3|\Sigma|)$ time.

- The automaton construction was not optimal, and in fact it can be done in optimal $O(m|\Sigma|)$ time. This is optimal because $m|\Sigma|$ is the size of the transition table δ. This gives a total of $O(n + m|\Sigma|)$ time.

- The next algorithm, Knuth-Morris-Pratt, achieves $O(n)$ matching time with only $O(m)$ preprocessing time. It doesn't compute the transition function at all.

The Prefix Function

The Knuth-Morris-Pratt (KMP) algorithm uses a prefix function π that allows a transition to be computed in O(1) amortized time. One nice thing about π is that it is independent of the input character and only contains m elements.

Given a pattern P[1..m], the prefix function π for P maps {1..m} to {0..m-1} such that:

$$\pi[q] = \max \{k : k<q \text{ and } P_k \sqsupset P_q\}$$

It's the length of the maximum proper prefix of P that is a suffix of P_q.

The Prefix Function

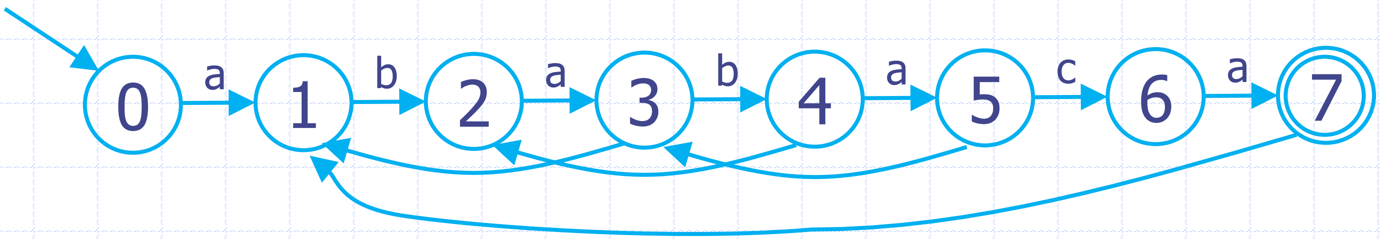
i	1	2	3	4	5	6	7
$P[i]$	a	b	a	b	a	c	a
$\pi[i]$	0	0	1	2	3	0	1

- for q=3, P_1 = a is the longest proper prefix of P_3 = aba that is also a suffix.
- for q=4, P_2 = ab is the longest proper prefix of P_4 = abab that is also a suffix.
- for q=5, P_3 = aba is the longest proper prefix of P_5 = ababa that is also a suffix.
- for q = 6, P_0 = ε is the longest proper prefix of P_6 = ababac that is also a suffix.

Using The Prefix Function

i	1	2	3	4	5	6	7
$P[i]$	a	b	a	b	a	c	a
$\pi[i]$	0	0	1	2	3	0	1

rule: if you fail to go forward, follow the back edge and do not consume input character.



back edges to 0 are omitted from diagram.

	a	b	a	c	b	a	b	a	b	a	b	a	c	a	a	b	a	b	a	b	b	a
0	1	2	3	0	0	1	2	3	4	5	4	5	6	7	1	2	3	4	5	4	0	1

KMP Matcher

```
KMP-MATCHER(T, P)
n = length(T)
m = length(P)
π = COMPUTE-PREFIX-FUNCTION(P)
q = 0
for i = 1 to n
    while q > 0 and P[q+1] ≠ T[i]
        q = π [q]
    if P[q+1] = T[i]
        q = q + 1
    if q = m
        output i – m
        q = π [q]
```

KMP Matcher

```
KMP-MATCHER(T, P)
n = length(T)
m = length(P)
π = COMPUTE-PREFIX-FUNCTION(P)
q = 0
for i = 1 to n                                  O(n) iterations
    while q > 0 and P[q+1] ≠ T[i]                  ???
        q = π [q]
    if P[q+1] = T[i]
        q = q + 1
    if q = m                                       O(1)
        output i – m
        q = π [q]
```

KMP Matcher

```
KMP-MATCHER(T, P)
n = length(T)
m = length(P)
π = COMPUTE-PREFIX-FUNCTION(P)
q = 0                    // invariant: coin on every state < q
for i = 1 to n                                  O(n) iterations
    while q > 0 and P[q+1] ≠ T[i]                    O(1)
        q = π [q]             // paid for by coin on π [q]
    if P[q+1] = T[i]
        q = q + 1  // add a coin to q first
    if q = m                                         O(1)
        output i – m
        q = π [q]
```

Computing the Prefix Function

```
COMPUTE-PREFIX-FUNCTION(P)
m = length(P)
π[1] = 0
k = 0
for q = 2 to m
    while k > 0 and P[k+1] ≠ P[q]
        k = π[k]
    if P[k+1] = P[q]
        k = k + 1
    π[q] = k
return π
```

	a	b	a	b	a	c	a
q	1	2	3	4	5	6	7
k	0	0	1	2	3	0	1
π	0	0	1	2	3	0	1

String Matching

Algorithm	Preprocessing time	Matching time
Naive	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\,\lvert\Sigma\rvert)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$