طراحی الگوریتم

۲۴ آذر ۹۸

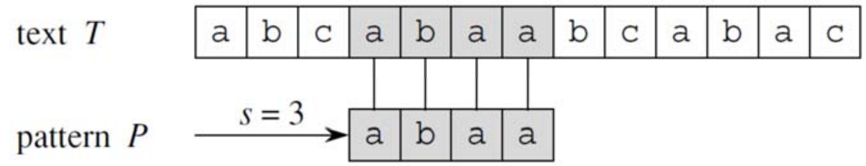
ملکی مجد

Topic	Reference
Recursion and Backtracking	Ch.1 and Ch.2 JeffE
Dynamic Programming	Ch.3 JeffE and Ch.15 CLRS
Greedy Algorithms	Ch.4 JeffE and Ch.16 CLRS
Amortized Analysis	Ch.17 CLRS
Elementary Graph algorithms	Ch.6 JeffE and Ch.22 CLRS
Minimum Spanning Trees	Ch.7 JeffE and Ch.23 CLRS
Single-Source Shortest Paths	Ch.8 JeffE and Ch.24 CLRS
All-Pairs Shortest Paths	Ch.9 JeffE and Ch.25 CLRS
Maximum Flow	Ch.10 JeffE and Ch.26 CLRS
String Matching	Ch.32 CLRS
NP-Completeness	Ch.34 CLRS

String matching – تطابق رشته ها

- Finding all occurrences of a pattern in a text

Formal definition of string marching



- The text is an array $\boldsymbol{T}\,[\mathbf{1}\,..\,\boldsymbol{n}]$ of length n
- That the pattern is an array $\boldsymbol{P}[\mathbf{1}\,..\,\boldsymbol{m}]$ of length $m \le n$
- Elements of P and T are characters drawn from a finite alphabet Σ
  - For example $\Sigma = \{0,1\}$ or $\Sigma = \{a, b, \ldots, z\}$.
  - Character arrays P and T are often called ***strings*** of characters

Pattern *P* ***occurs with shift s*** in text *T*:(*P* ***occurs beginning at position s*** **+ 1** in *T*)

If $0 \le s \le n - m$ and $T\,[s + 1\,..\,s + m] = P[1\,..\,m]$

(that is, if $T\,[s + j\,] = P[\,j\,]$, for $1 \le j \le m$).

Some string-matching algorithms:

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	$\Theta(m)$	$\Theta(n)$

The preprocessing time is based on the pattern

This course covers the algorithms of Naïve, Radin-Karp and Finite automaton

The naive string-matching algorithm

```
NAIVE-STRING-MATCHER(T, P)
n ← length[T]
m ← length[P]
for s ← 0 to n − m
        do if P[1 . . m] = T[s + 1 . . s + m]
              then print "Pattern occurs with shift" s
```

takes time $O((n - m + 1)m)$, and this bound is tight in the worst case.

Example : text a^n and pattern a^m

RABIN-KARP-MATCHER(T, P, d, q)

```
n ← length[T]
m ← length[P]
h ← d^(m-1) mod q
p ← 0
t_0 ← 0
for i ← 1 to m                              ▷ Preprocessing.
     do p ← (dp + P[i]) mod q
        t_0 ← (dt_0 + T[i]) mod q
for s ← 0 to n − m                          ▷ Matching.
     do if p = t_s
           then if P[1 . . m] = T[s + 1 . . s + m]
                   then print "Pattern occurs with shift" s
        if s < n − m
           then t_{s+1} ← (d(t_s − T[s + 1]h) + T[s + m + 1]) mod q
```

Use hashing

```
RabinKarp(S[1..n], P[1..m])
        hpattern = hash(P[1..m]);
        for i from 1 to n-m+1
                hs = hash(S[i..i+m-1])
                        if hs == hpattern
                                if s[i+1..i+m] == pattern[1..m]
                                        print "Pattern occurs with shift i"
```

String matching with finite automata

Finite automata - definition

A finite automaton M is a 5-tuple $(Q, q_0, A, \Sigma, \delta)$, where

Q	a finite set of states
q_0	the start state
A	$\subseteq Q$ – a distinguished set of accepting states
Σ	a finite input alphabet
δ	a function from $Q \times \Sigma$ into Q, called the transition function of M.

Finite automata – how it works

A finite automaton M is a 5-tuple $(Q, q_0, A, \Sigma, \delta)$, where

Q a finite set of states

q_0 the start state

A $\subseteq Q$ – a distinguished set of accepting states

Σ a finite input alphabet

δ a function from $Q \times \Sigma$ into Q, called the transition function of M.

The finite automaton **begins in state *q*0** and

reads the characters of its **input string one at a time**.

If the automaton is in state ***q* and reads input character *a***, it moves ("makes a transition") from **state *q* to state *δ(q, a)*.**

Whenever its current state ***q* is a member of *A***, the machine M is said to have ***accepted*** the string read so far. (An input that is not accepted is said to be ***rejected)***

Finite automata – example

A finite automaton M is a 5-tuple $(Q, q_0, A, \Sigma, \delta)$, where

Q a finite set of states

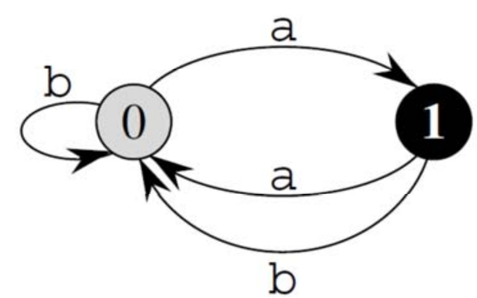
q_0 the start state

A $\subseteq Q$ – a (accepting states)

Σ a finite input alphabet

δ a function from $Q \times \Sigma$ into Q, (transition function).

	input	
state	a	b
0	1	0
1	0	0



begins in state q0 and

reads **input string one at a time**.

in state ***q* and reads input character *a***, it moves **to state *δ(q, a)***.

current state ***q* is a member of *A***: ***accepted*** the string read so far.

final-state function

- from Σ^* to Q such that $\phi(w)$ is the state M ends up in after scanning the string *w*
- *M* accepts a string *w* if and only if $\phi(w) \ \in \ A$.
- The function ϕ is defined by the recursive relation

$$\phi(\varepsilon) \ = \ q0\,,$$

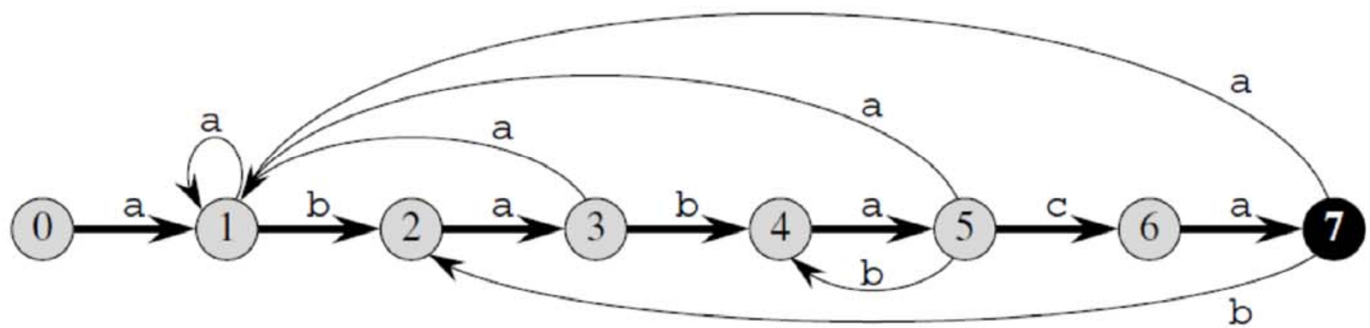
$$\phi(wa) \ = \ \delta(\phi(w), a) \text{ for } w \in \Sigma^*,\ a \in \Sigma$$

String-matching automata

- There is a string-matching automaton for every pattern *P*;

- Let see an example for pattern *P* = 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	



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

String-matching automata
suffix function

- Define an auxiliary function σ , called the ***suffix function*** corresponding to *P*.
- Which is a mapping from Σ^* to $\{0, 1, \ldots, m\}$ such that $\sigma(x)$ is the length of the longest prefix of *P* that is a suffix of *x*:

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

- The suffix function σ is well defined since the empty string $P_0 = \varepsilon$ is a suffix of every string

As examples, for the pattern *P* = ab, we have $\sigma(\varepsilon) = 0$, $\sigma(\text{ccaca}) = 1$, and $\sigma(\text{ccab}) = 2$.

For a pattern *P* of length *m*, we have *σ(x)* = *m* if and only if *P* ⊐ *x*.

string-matching automaton
corresponds to a given pattern

define the string-matching automaton that corresponds to a given pattern $P[1\,..\,m]$:

- The state set Q is $\{0, 1, \ldots, m\}$. The start state q_0 is state 0, and state m is the only accepting state.
- The transition function δ is defined by the following equation, for any state q and character a: $\delta(q, a) = \sigma(P_q a)$

How construct automaton

COMPUTE-TRANSITION-FUNCTION(P, Σ)

```
m ← length[P]
for q ← 0 to m
      do for each character a ∈ Σ
            do k ← min(m + 1, q + 2)
               repeat k ← k − 1
                 until P_k ⊐ P_q a
               δ(q, a) ← k
return δ
```

Time complexity?

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

Construct the string-matching automaton for the pattern P = aabab and illustrate its operation on the text string T = aaababaabaababaab