Design and Analysis of Algorithms

Lecture 11: String Matching

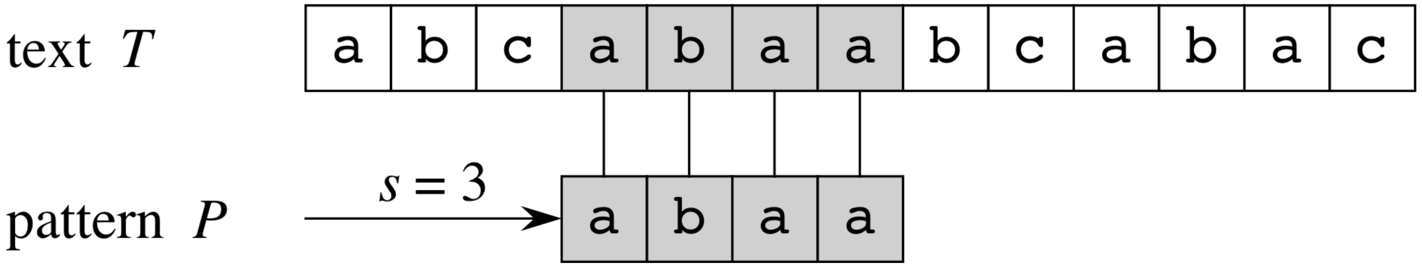
Ahmed Hamdy

Agenda

- Naïve algorithm
- Rabin-Karp algorithm
- Finite Automata (FA) algorithm
- Knuth-Morris-Pratt (KMP) algorithm

String matching

- Simply put, find *all occurrences* of string called *pattern* P (of length m) inside another one called *text* T (of length n).



- Can be viewed as find the *shift* s ($0 \le s \le n - m$) by which P appears in T.
- Σ denotes the alphabet; the unique characters in P (a and b in the above example).

String matching algorithms

Performance of algorithms

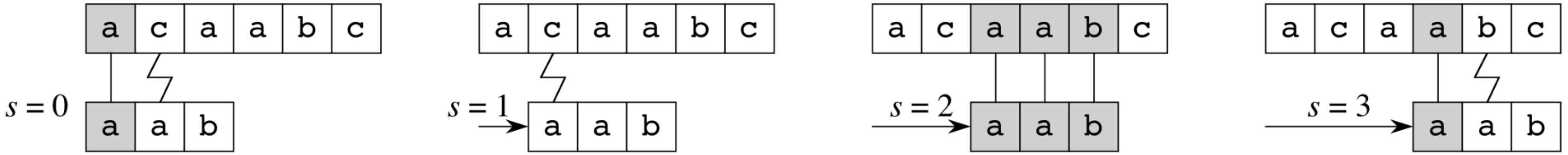
- Σ denotes the alphabet

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

- When is each algorithm suitable??

Naive string-matching

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



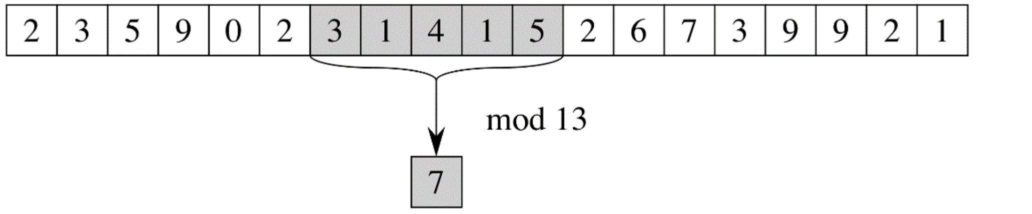
(a) (b) (c) (d)

- Worst case running time $O((n-m+1)m)$ which is $O(n^2)$ if $m = \lfloor n/2 \rfloor$.
- Room for optimization where the algorithm does not make use of information from previous iteration.

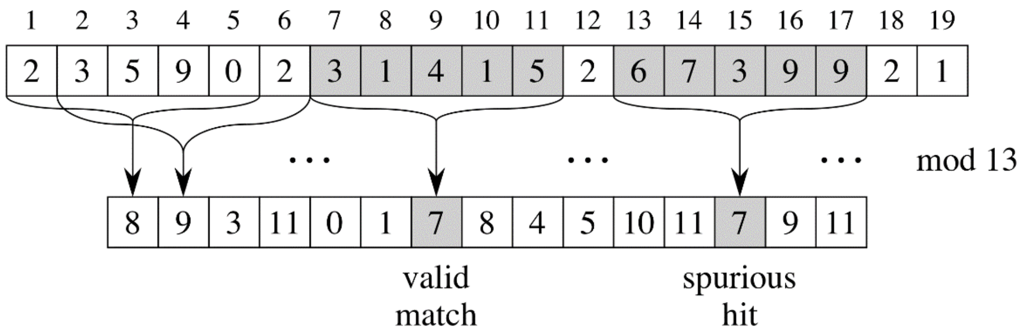
Rabin-Karp algorithm

- Compute the hash for the *pattern* P.
- Compute the hash for the *text* T at each shift $s = 0..(n - m + 1)$.
- Compare $hash(P)$ with each $hash(T[s + 1: s + m])$
  - If equal, compare character by character to verify the collision of hashes.
- How to optimize the computation of hashes for each s?

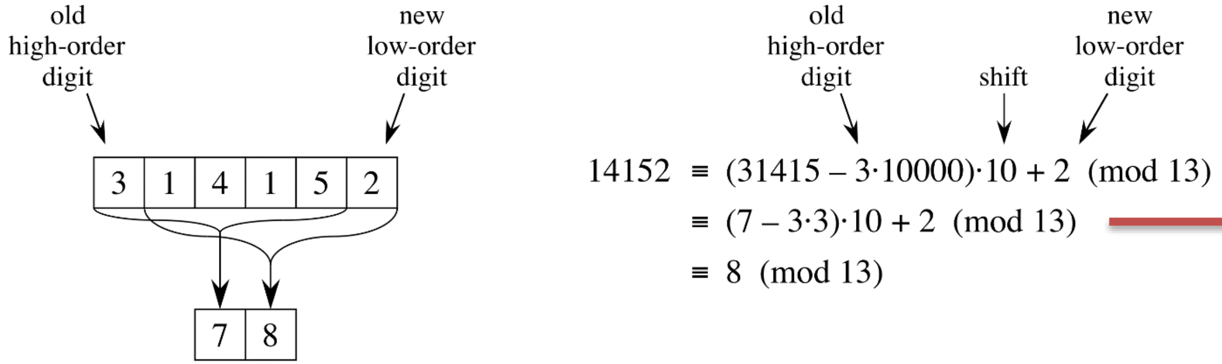
Rabin-Karp algorithm



(a)



(b)



(c)

$$\boldsymbol{hash(s+1)}$$
$$\boldsymbol{= (hash(s) - T[s+1].const1).const2 + T[s+m+1] \bmod const3}$$

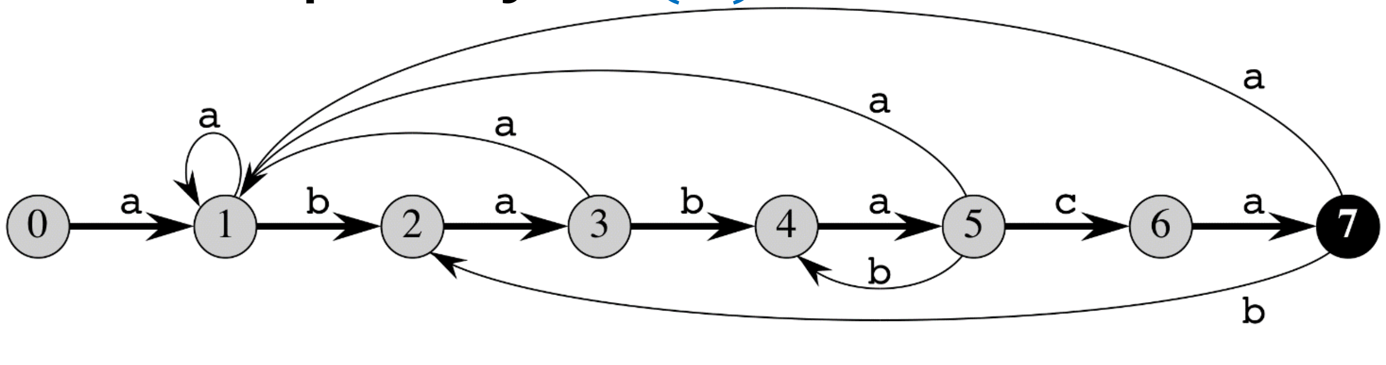
Rabin-Karp algorithm

```
RABIN-KARP-MATCHER(T, P, d, q)
n = T.length
m = P.length
h = d^(m-1) mod q
p = 0
t_0 = 0
for i = 1 to m                          // preprocessing
    p = (dp + P[i]) mod q               // (d . p + P[i]) mod q
    t_0 = (dt_0 + T[i]) mod q           // (d . t_0 + T[i]) mod q
for s = 0 to n - m                      // matching
    if p == t_s
        if P[1..m] == T[s+1..s+m]
            print "Pattern occurs with shift" s
    if s < n - m
        t_(s+1) = (d(t_s - T[s+1]h) + T[s+m+1]) mod q
```

- Though worst case is not better than the naïve algorithm, the average case is much better, typically $O\big((n-m+1)+cm\big) = O(n+m)$

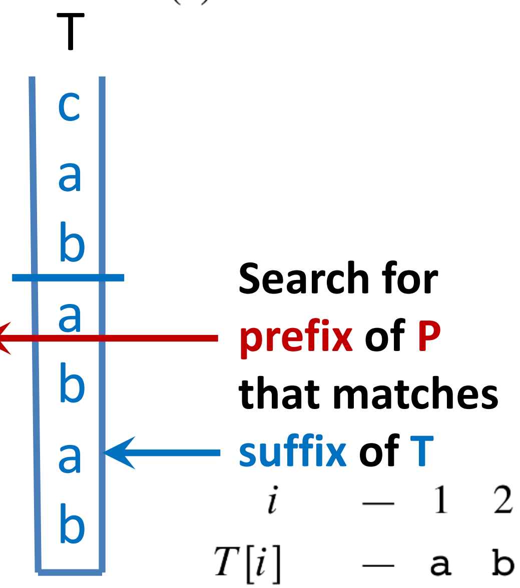
Finite automata

- Complexity: $\Theta(n)$



(a)

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	$\boxed{4}$	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

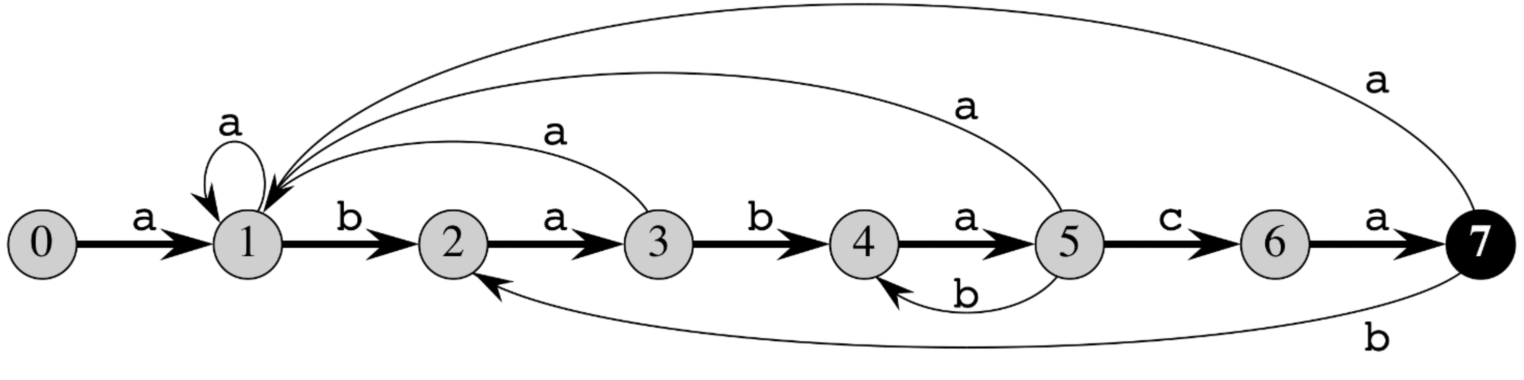
(c)

FINITE-AUTOMATON-MATCHER(T, δ, m)

```
n = T.length
q = 0
for i = 1 to n
    q = δ(q, T[i])
    if q == m
        print "Pattern occurs with shift" i − m
```

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 omitted; by convention, if a state i has no outgoing edge labeled a for some $a \in \Sigma$, 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 $T[i]$ appears the state $\phi(T_i)$ that the automaton is in after processing the prefix T_i. The automaton finds one occurrence of the pattern, ending in position 9.

Finite Automata



(a)

COMPUTE-TRANSITION-FUNCTION(P, Σ)

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

When $q = 5$, initially $k = (q + 2) - 1 = 6$.

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	$\boxed{4}$	6	c
6	7	0	0	a
7	1	2	0	

T

c

a

b

a

b

a

b

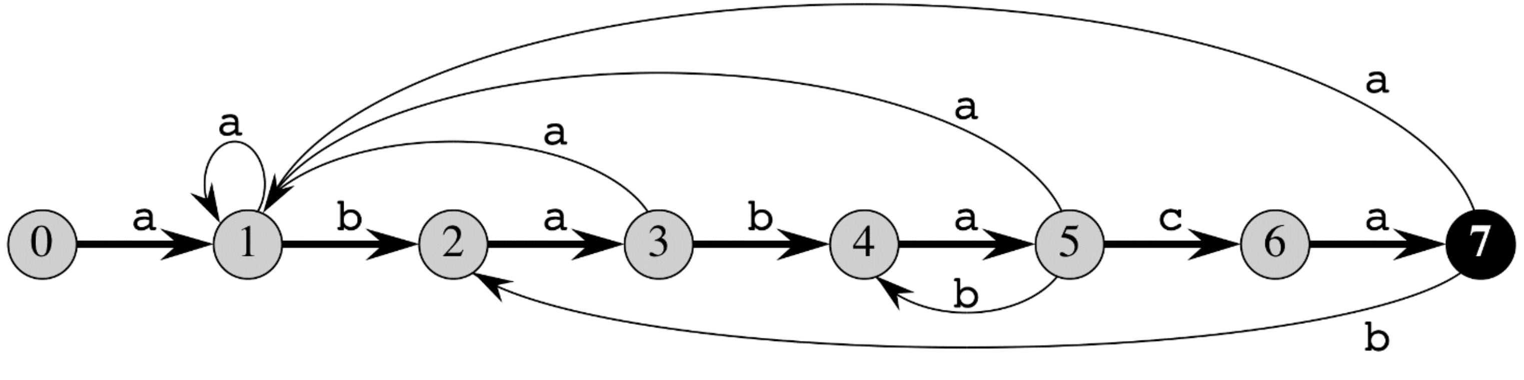
Search for **prefix of P** that matches **suffix of T**

(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 Automata

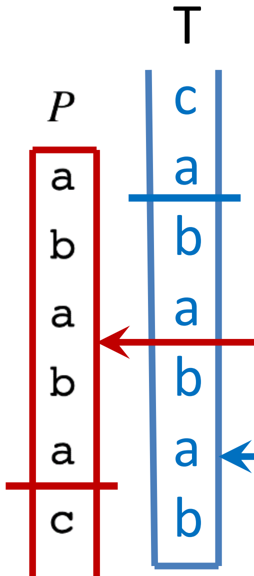


(a)

COMPUTE-TRANSITION-FUNCTION(P, Σ)

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

state	input a	input b	input c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	4	0
4	5	0	0
5	1	$\boxed{4}$	6
6	7	0	0
7	1	2	0



When $q = 5$, initially $k = k - 1 = 5$.

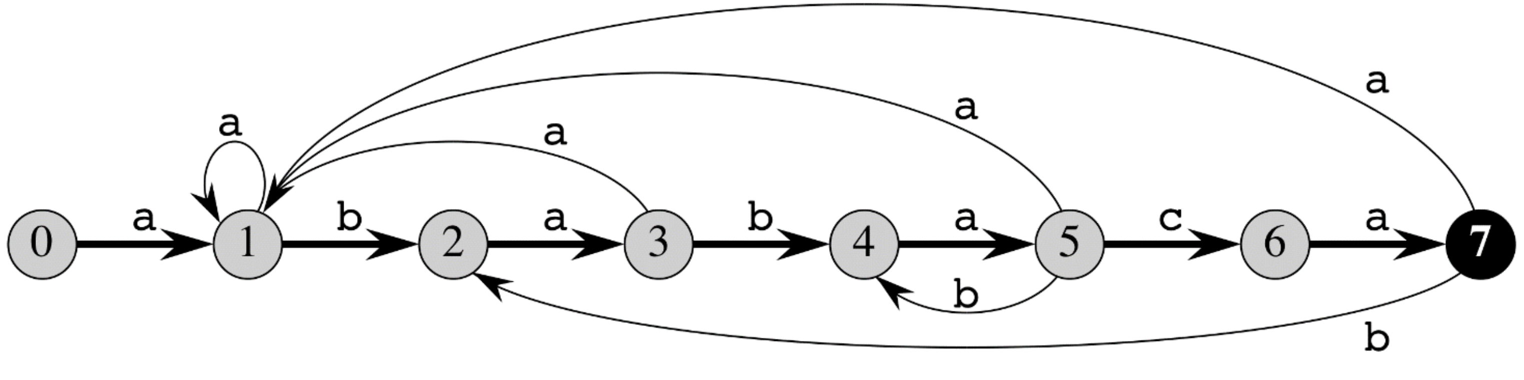
Search for **prefix of P** that matches **suffix of T**

(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 Automata



(a)

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	$\boxed{4}$	6	c
6	7	0	0	a
7	1	2	0	

(b)

T: c a b a b a b

When $q = 5$, initially $k = k - 1 = 4$.

Search for prefix of P that matches suffix of T

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)

COMPUTE-TRANSITION-FUNCTION(P, Σ)

$m = P.length$
for $q = 0$ **to** m
  for each character $a \in \Sigma$
    $k = \min(m + 1, q + 2)$
    repeat
      $k = k - 1$
    until $P_k \sqsupset P_q a$
    $\delta(q, a) = k$
return δ

From FA to KMP

- The table produced by FA can be squeezed to be one column!!
- Don’t compute where to go back for each character (‘a’, ’b’, and ‘c’).
- Instead, go back to what is common and check there.
- Now preprocessing complexity is $\Theta(m)$ instead of $O(m|\Sigma|)$.

From FA to KMP

$q = 0$

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

state	input a	b	c	
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	

- A variable q holds the current state
- Instead of having the straightforward entries (1,2,3,..7) as we match each character in P, q will hold this value and gets incremented each time.
- Initially $q = 0$ (points before the table, corresponds to i inside the above table).

From FA to KMP: Correct-matching

~~$q=0$~~ $q=1$

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

$T = a \ldots \ldots$

state	input a	input b	input c	
0	$\boxed{1}$	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	

a $\rightarrow$ If $T[1] = P[q+1] = a$, then $q = q + 1 = 1.$

* It can be any $T[.]$, $T[1]$ is used for simplicity.

From FA to KMP: Correct-matching

~~$q=1$~~ $q=2$

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

$T = ab \ldots \ldots$

state	input a	input b	input c	
0	1	0	0	a
1	1	$\boxed{2}$	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	

If $T[2] = P[q+1] = b$, then $q = q + 1 = 2.$

From FA to KMP: Correct-matching

~~$q = 2$~~ $q = 3$

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

$T = aba\ldots\ldots$

state	input a	input b	input c	
0	1	0	0	a
1	1	2	0	b
2	$\boxed{3}$	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	

If $T[3] = P[q + 1] = a$, then $q = q + 1 = 3$.

From FA to KMP: Correct-matching

~~$q = 3$~~ $q = 4$

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

$T = abab \ldots \ldots$

state	input a	input b	input c	
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	$\boxed{4}$	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

If $T[4] = P[q+1] = b$, then $q = q + 1 = 4$.

From FA to KMP: Correct-matching

$\cancel{q=4}$ $q=5$

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

$T = ababa \ldots \ldots$

state	input a	b	c	
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	$\boxed{5}$	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

If $T[5] = P[q+1] = a$, then $q = q + 1 = 5.$

From FA to KMP: Correct-matching

~~$q=5$~~ $q=6$

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

$T = ababac \dots\dots$

state	input a	b	c	
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	$\boxed{6}$	c
6	7	0	0	a
7	1	2	0	

If $T[6] = P[q+1] = c$, then $q = q + 1 = 6.$

From FA to KMP: Correct-matching

~~$q=6$~~ $q=7$

$T = ababaca \dots\dots$

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

	input			
state	a	b	c	
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	$\boxed{7}$	0	0	a
7	1	2	0	

If $T[7] = P[q+1] = a$, then $q = q + 1 = 7$.

From FA to KMP: Correct-matching

$q=1$ $\cancel{q=7}$

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

$T = ababaca \ldots \ldots$

	input			
state	a	b	c	
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	

After completing a match, we go back to the overlap between P and itself which is of size 1.
So $q = \pi[q] = 1$. Means start a **new** match with a.

From FA to KMP: Miss-Matching

$q = 0$

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

state	input a	input b	input c	
0	1	$\boxed{0}$	$\boxed{0}$	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	

If $T[1] = b$ or $T[1] = c$ while $P[q+1] = a$, then still $q = 0$.

From FA to KMP: Miss-matching

$q = 0$ $\cancel{q = 1}$

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

state	input a	b	c	
0	1	0	0	a
1	$\boxed{1}$	2	$\boxed{0}$	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	

If $T[2] = a$ or $T[2] = c$ while $P[q + 1] = b$, then $q = \pi[q] = \pi[1] = 0.$

From FA to KMP: Miss-matching

$q = 0$ $\cancel{q = 2}$

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

state	input a	input b	input c	
0	1	0	0	a
1	1	2	0	b
2	3	$\boxed{0}$	$\boxed{0}$	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	

If $T[3] = b$ or $T[3] = c$ while $P[q+1] = a$, then $q = \pi[q] = \pi[2] = 0.$

From FA to KMP: Miss-matching

$q=1$ ~~$q=3$~~

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

state	input a	b	c	
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	$\boxed{1}$	4	$\boxed{0}$	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

If $T[4]=a$ or $T[4]=c$ while $P[q+1]=b$, then $q=\pi[q]=\pi[3]=1.$

From FA to KMP: Miss-matching

$q=0$ $\cancel{q=1}$ $\cancel{q=3}$

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

state	input a	input b	input c	
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	

If $T[4] = a$ or $T[4] = c$ while $P[q+1] = b$, then $q = \pi[q] = \pi[3] = 1.$
At $q = 1$, check again whether a or c matches $P[q+1] = b$ which is false again, so $q = \pi[q] = \pi[1] = 0.$

From FA to KMP: Miss-matching

$q=2$ ~~$q=4$~~

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

state	input a	input b	input c	
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	

If $T[5] = b$ or $T[5] = c$ while $P[q + 1] = a$, then $q = \pi[q] = \pi[4] = 2.$

From FA to KMP: Miss-matching

$q=0$ ~~$q=2$~~   ~~$q=4$~~

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

state	input a	b	c	
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	$\boxed{0}$	$\boxed{0}$	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

If $T[5] = b$ or $T[5] = c$ while $P[q+1] = a$, then $q = \pi[q] = \pi[4] = 2.$
At $q = 2$, check again whether b or c matches $P[q+1] = a$ which is false again, so $q = \pi[q] = \pi[2] = 0.$

From FA to KMP: Miss-matching

$q = 3$ ~~$q = 5$~~

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

state	input a	input b	input c	
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	$\boxed{1}$	$\boxed{4}$	6	c
6	7	0	0	a
7	1	2	0	

If $T[6] = a$ or $T[6] = b$ while $P[q+1] = c$, then $q = \pi[q] = \pi[5] = 3.$

From FA to KMP: Miss-matching

$q=1$ $\cancel{q=3}$ $\cancel{q=5}$

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

	input			
state	a	b	c	
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	

If $T[6] = a$ or $T[6] = b$ while $P[q+1] = b$, then $q = \pi[q] = \pi[5] = 3.$

At $q = 3$, check again whether a or b matches $P[q+1] = b$:

If $T[6] = a$ which is false again, so $q = \pi[q] = \pi[2] = 1.$

From FA to KMP: Miss-matching

$q=0$ ~~$q=1$~~ $q=3$ ~~$q=5$~~

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

state	input a	b	c	
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	$\boxed{1}$	$\boxed{4}$	6	c
6	7	0	0	a
7	1	2	0	

If $T[6] = a$ or $T[6] = b$ while $P[q+1] = b$, then $q = \pi[q] = \pi[5] = 3.$
At $q = 3$, check again whether a or b matches $P[q+1] = b$:
If $T[6] = a$ which is false again, so $q = \pi[q] = \pi[2] = 1.$
At $q = 1$, check again if a matches $P[q+1] = b$ which is false, so $q = \pi[q] = \pi[1] = 0.$

From FA to KMP: Miss-matching

$q = 3$ $\cancel{q = 5}$

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

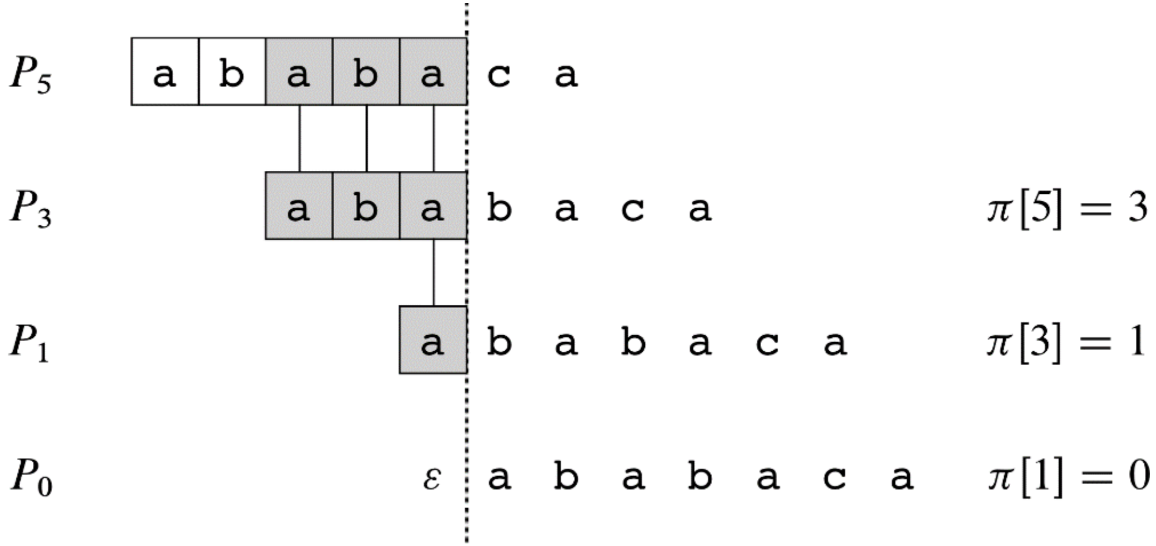
state	input a	input b	input c	
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	$\boxed{1}$	$\boxed{4}$	6	c
6	7	0	0	a
7	1	2	0	

If $T[6] = a$ or $T[6] = b$ while $P[q+1] = b$, then $q = \pi[q] = \pi[5] = 3.$
At $q = 3$, check again whether a or b matches $P[q+1] = b$:
If $T[6] = b$ which is true, so continue normal matching with $q = 3.$

Knuth-Morris-Pratt algorithm

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

(a)



(b)

KMP-MATCHER(T, P)

```
n = T.length
m = P.length
π = COMPUTE-PREFIX-FUNCTION(P)
q = 0                                    // number of characters matched
for i = 1 to n                           // scan the text from left to right
    while q > 0 and P[q + 1] ≠ T[i]
        q = π[q]                         // next character does not match
    if P[q + 1] == T[i]
        q = q + 1                        // next character matches
    if q == m                            // is all of P matched?
        print "Pattern occurs with shift" i − m
        q = π[q]                         // look for the next match
```

KMP: π-table computation

- $\pi[1] = 0$ always
- For $q = 2..m$:
  - Find the longest suffix length in P that is equal to the prefix length of P.
  - Store it as $\pi[q]$.

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

KMP: π-table computation

- $\pi[1] = 0$ always
- For $q = 2..m$:
  - Find the longest suffix length in $\boldsymbol{P}$ that is equal to the prefix length of $\boldsymbol{P}$.
  - Store it as $\pi[q]$.

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

No suffix found to match prefix inside this window

KMP: π-table computation

- $\pi[1] = 0$ always
- For $q = 2..m$:
  - Find the longest suffix length in $\boldsymbol{P}$ that is equal to the prefix length of $\boldsymbol{P}$.
  - Store it as $\pi[q]$.

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

Suffix of length 1 matches prefix inside this window

KMP: π-table computation

- $\pi[1] = 0$ always
- For $q = 2..m$:
  - Find the longest suffix length in $\boldsymbol{P}$ that is equal to the prefix length of $\boldsymbol{P}$.
  - Store it as $\pi[q]$.

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

Suffix of length 2 matches
prefix inside this window

KMP: π-table computation

- $\pi[1] = 0$ always
- For $q = 2..m$:
  - Find the longest suffix length in $\boldsymbol{P}$ that is equal to the prefix length of $\boldsymbol{P}$.
  - Store it as $\pi[q]$.

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

Suffix of length 3 matches prefix inside this window

KMP: π-table computation

- $\pi[1] = 0$ always
- For $q = 2..m$:
  - Find the longest suffix length in $\boldsymbol{P}$ that is equal to the prefix length of $\boldsymbol{P}$.
  - Store it as $\pi[q]$.

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

No suffix found to match prefix inside this window

KMP: π-table computation

- $\pi[1] = 0$ always
- For $q = 2..m$:
  - Find the longest suffix length in $\boldsymbol{P}$ that is equal to the prefix length of $\boldsymbol{P}$.
  - Store it as $\pi[q]$.

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

Suffix of length 1 matches prefix inside this window

KMP: π-table computation algorithm

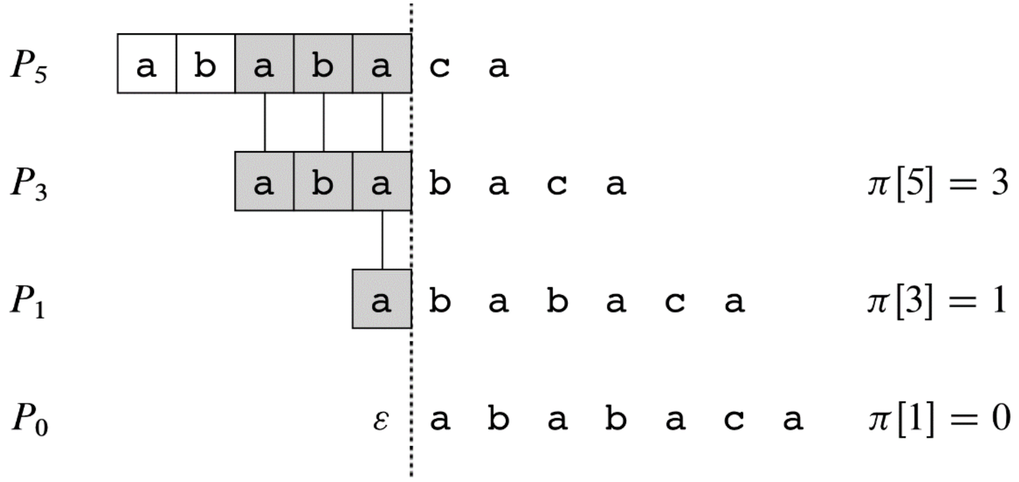
```
COMPUTE-PREFIX-FUNCTION(P)
m = P.length
let π[1..m] be a new array
π[1] = 0
k = 0
for q = 2 to m
    while k > 0 and P[k + 1] ≠ P[q]
        k = π[k]
    if P[k + 1] == P[q]
        k = k + 1
    π[q] = k
return π
```

// While next character does not match, keep backtracking through the π table

// If next character matches, increment the matching size k

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

(a)



(b)