CMP(N)302: Design and Analysis of Algorithms

Lecture 11: String Matching

Ahmed Hamdy

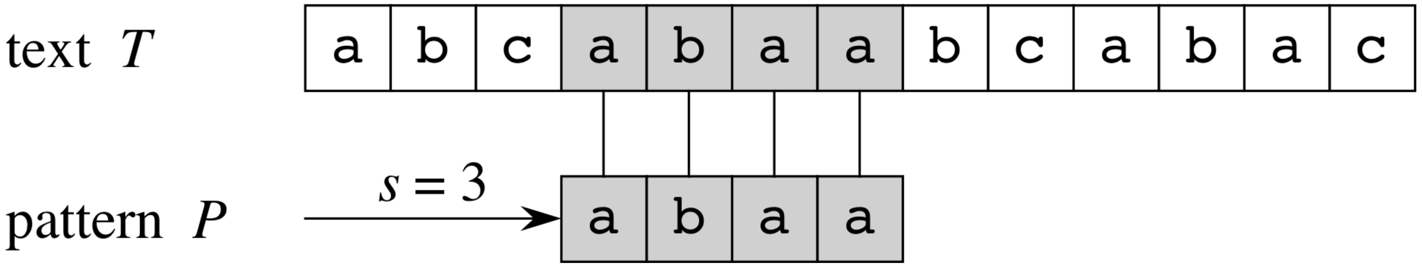
Computer Engineering Department

Cairo University

Fall 2019

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 $(0 \leq s \leq n - m)$ by which the P appears in T

String matching algorithms

- Performance of algorithms
- Σ denotes the alphabet

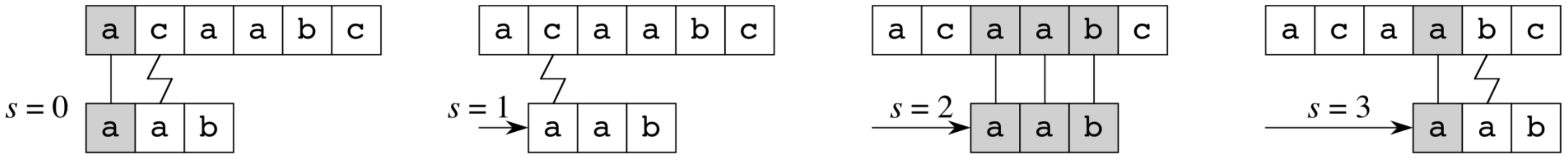
Algorithm	Preprocessing time	Matching time
Naive	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\,\lvert\Sigma\rvert)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$

- When is each algorithm suitable??

Naive string-matching

NAIVE-STRING-MATCHER(T, P)

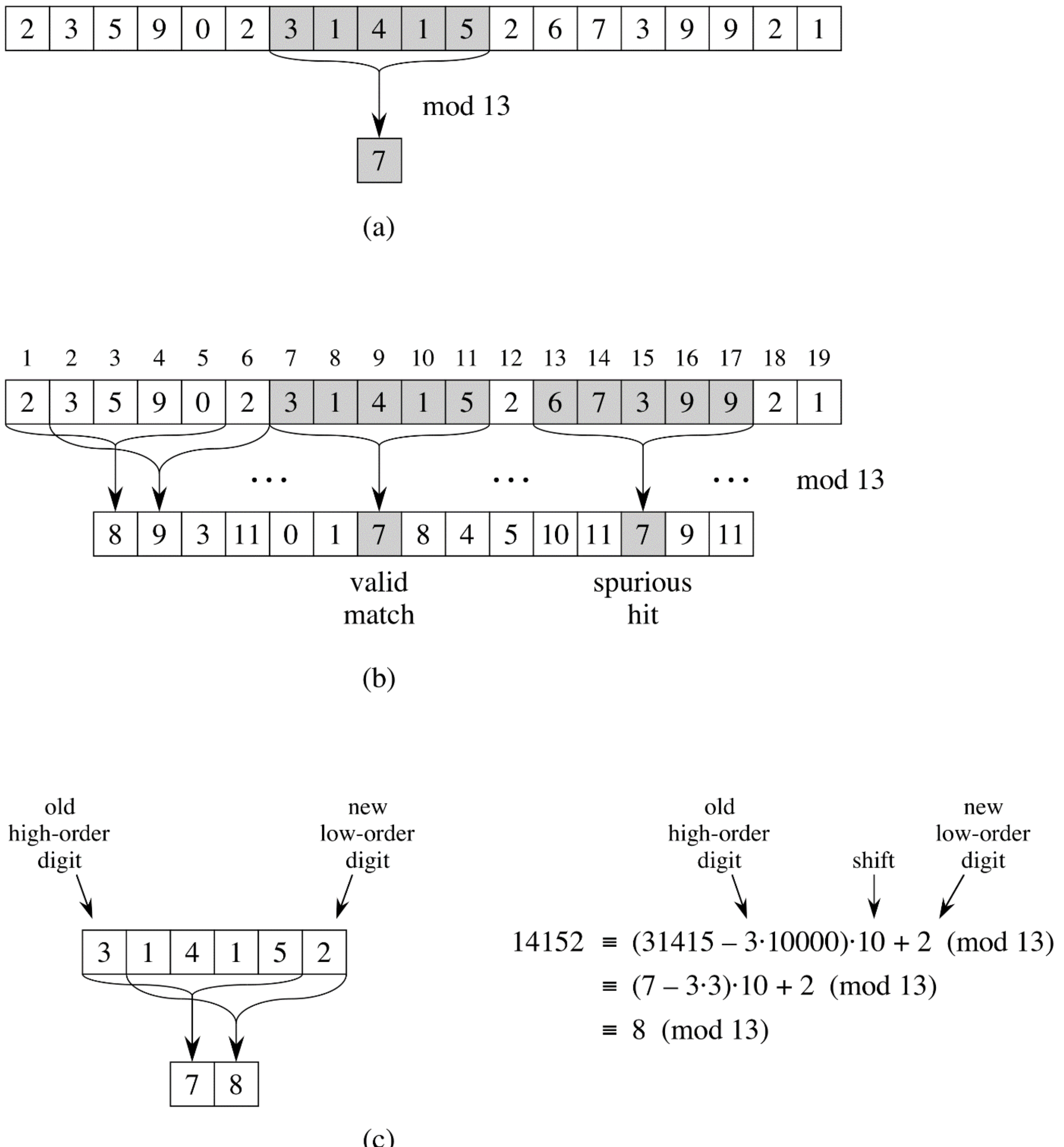
```
n = T.length
m = P.length
for s = 0 to n − m
    if P[1..m] == T[s + 1..s + m]
        print "Pattern occurs with shift" s
```



(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



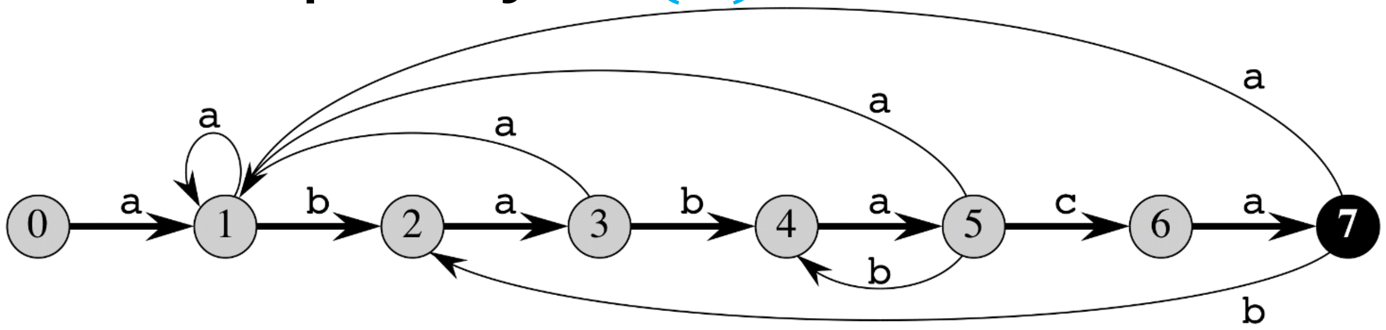
Rabin-Karp algorithm

```
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
    t_0 = (dt_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 $O\big((n-m+1)+cm\big) = O(n+m)$

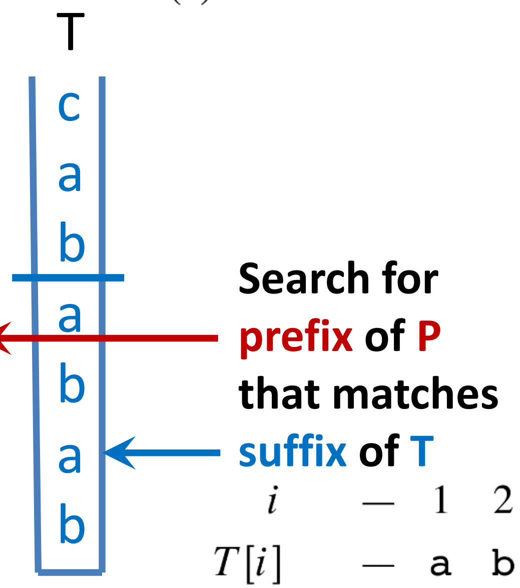
Finite automata

- 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

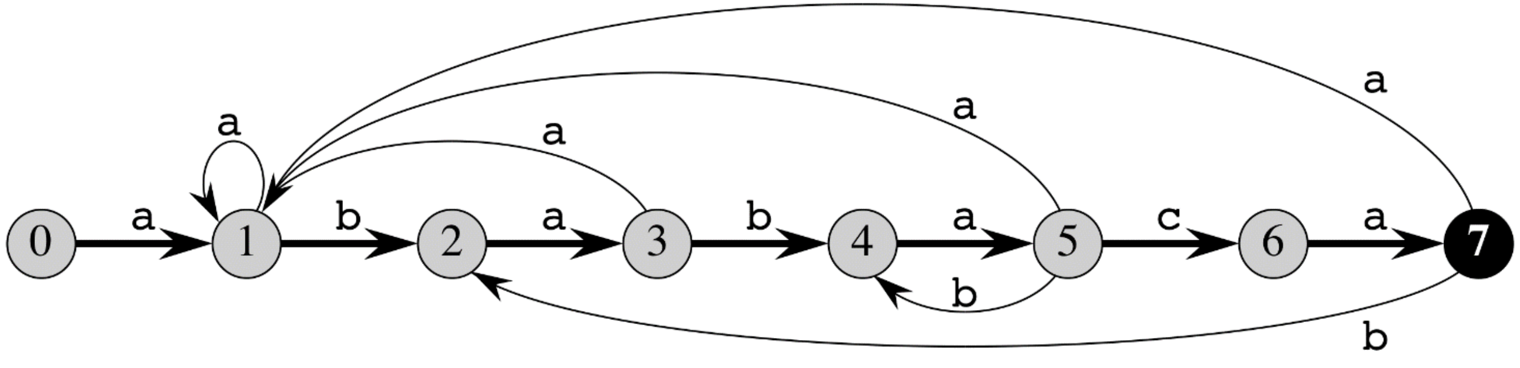
(c)

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)

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 $k = q + 2 - 1 = 6$

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

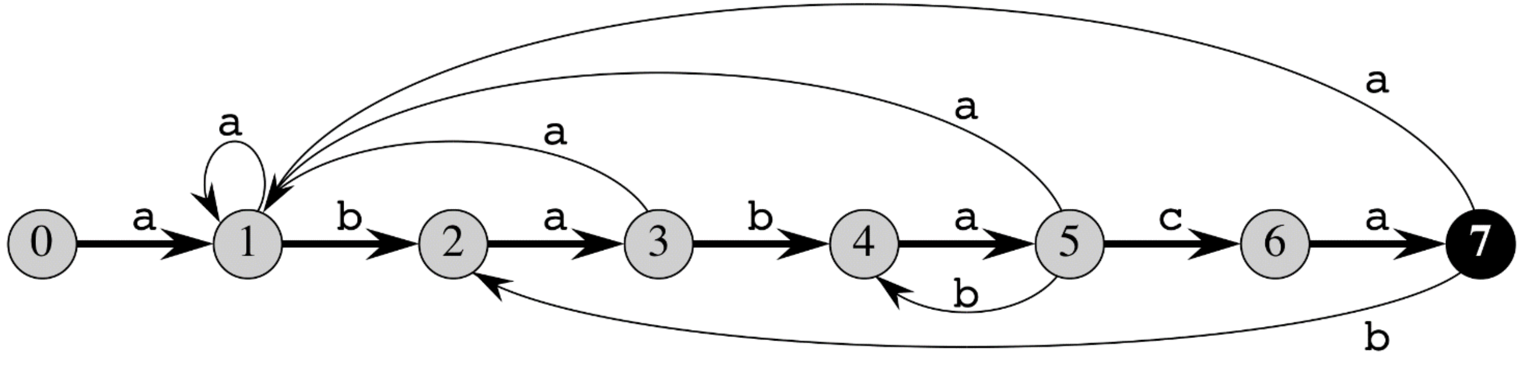
i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

(c)

COMPUTE-TRANSITION-FUNCTION(P, Σ)

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

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	

T: c a b a b a b

(b)

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

Search for prefix of P that matches suffix of T

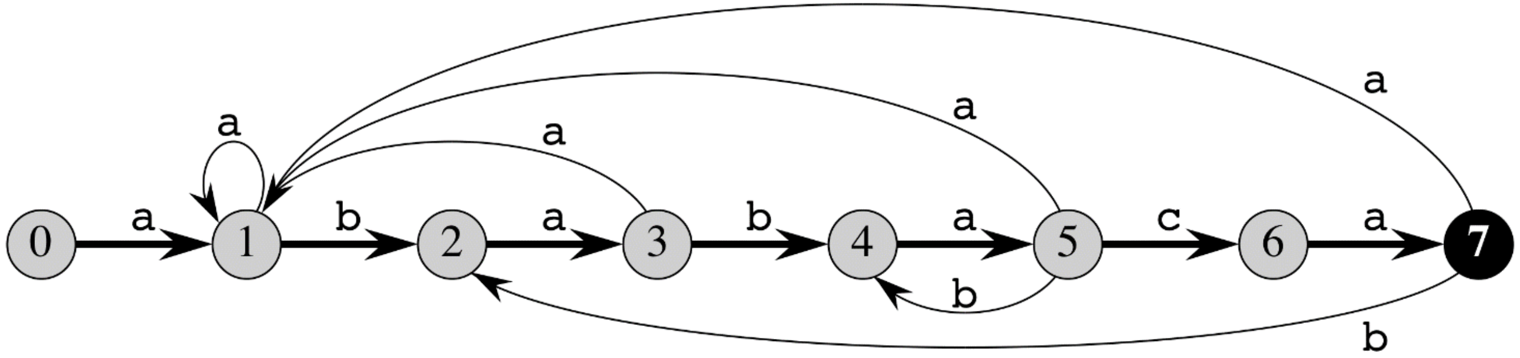
i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

(c)

COMPUTE-TRANSITION-FUNCTION(P, Σ)

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

Finite Automata



(a)

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

input

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**

(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)

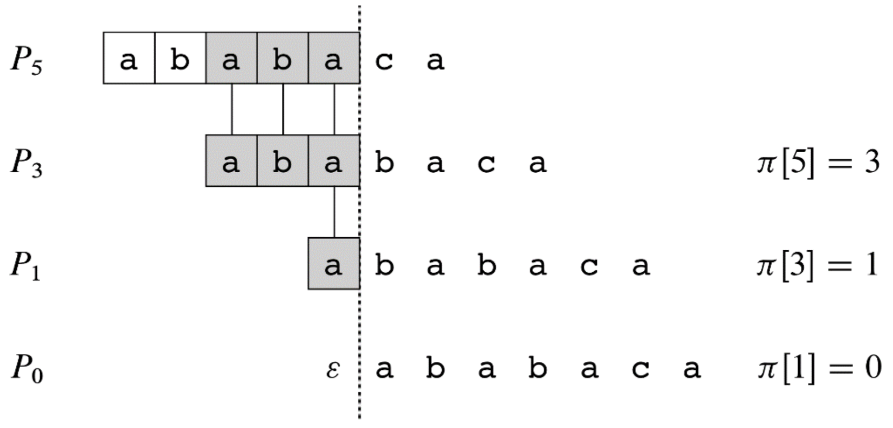
COMPUTE-TRANSITION-FUNCTION(P, Σ)

```
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
```

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

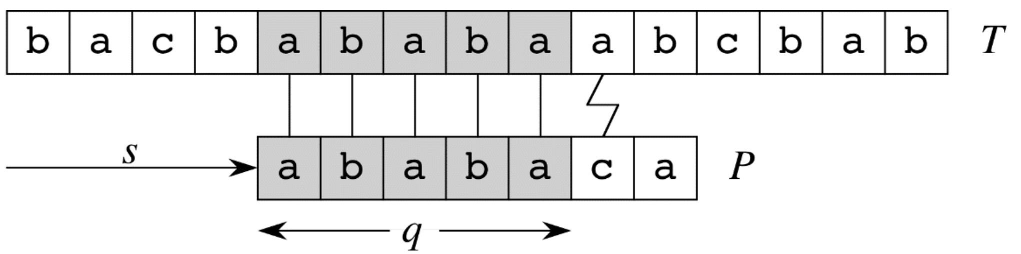
(a)



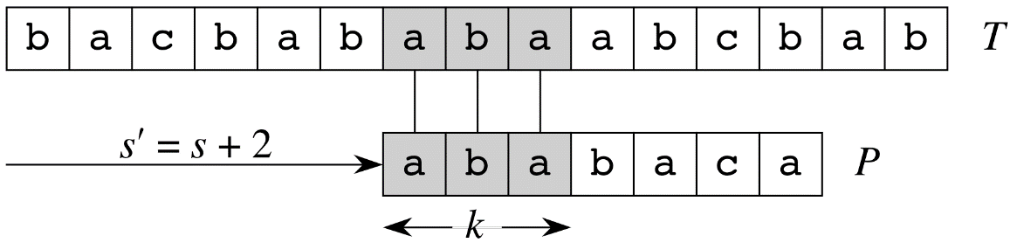
(b)

KMP-MATCHER(T, P)

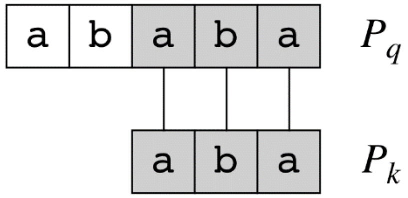
```
n = T.length
m = P.length
π = COMPUTE-PREFIX-FUNCTION(P)
q = 0                                   // number of characters matched
for i = 1 to n                          // scan the text from left to right
    while q > 0 and P[q + 1] ≠ T[i]
        q = π[q]                        // next character 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
```



(a)



(b)



(c)

Knuth-Morris-Pratt algorithm

COMPUTE-PREFIX-FUNCTION(P)

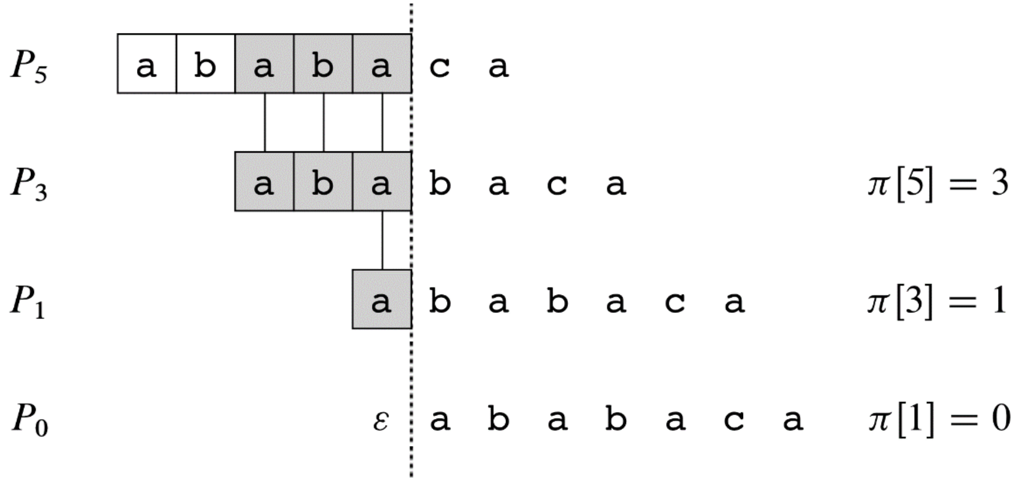
```
m = P.length
let π[1..m] be a new array
π[1] = 0
k = 0
for q = 2 to m
    while k > 0 and P[k + 1] ≠ P[q]
        k = π[k]
    if P[k + 1] == P[q]
        k = k + 1
    π[q] = k
return π
```

// 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)