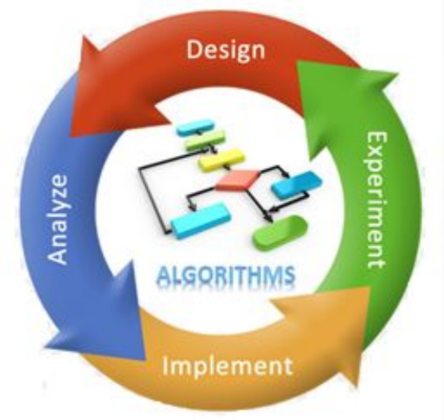
Analysis and Design of Algorithms (ADA)
GTU # 3150703

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Unit-8:
String Matching



Dr. Gopi Sanghani
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
gopi.sanghani@darshan.ac.in
9825621471

Outline

- Introduction
- The Naive String Matching Algorithm
- The Rabin-Karp Algorithm
- String Matching with Finite Automata
- The Knuth-Morris-Pratt Algorithm

Introduction

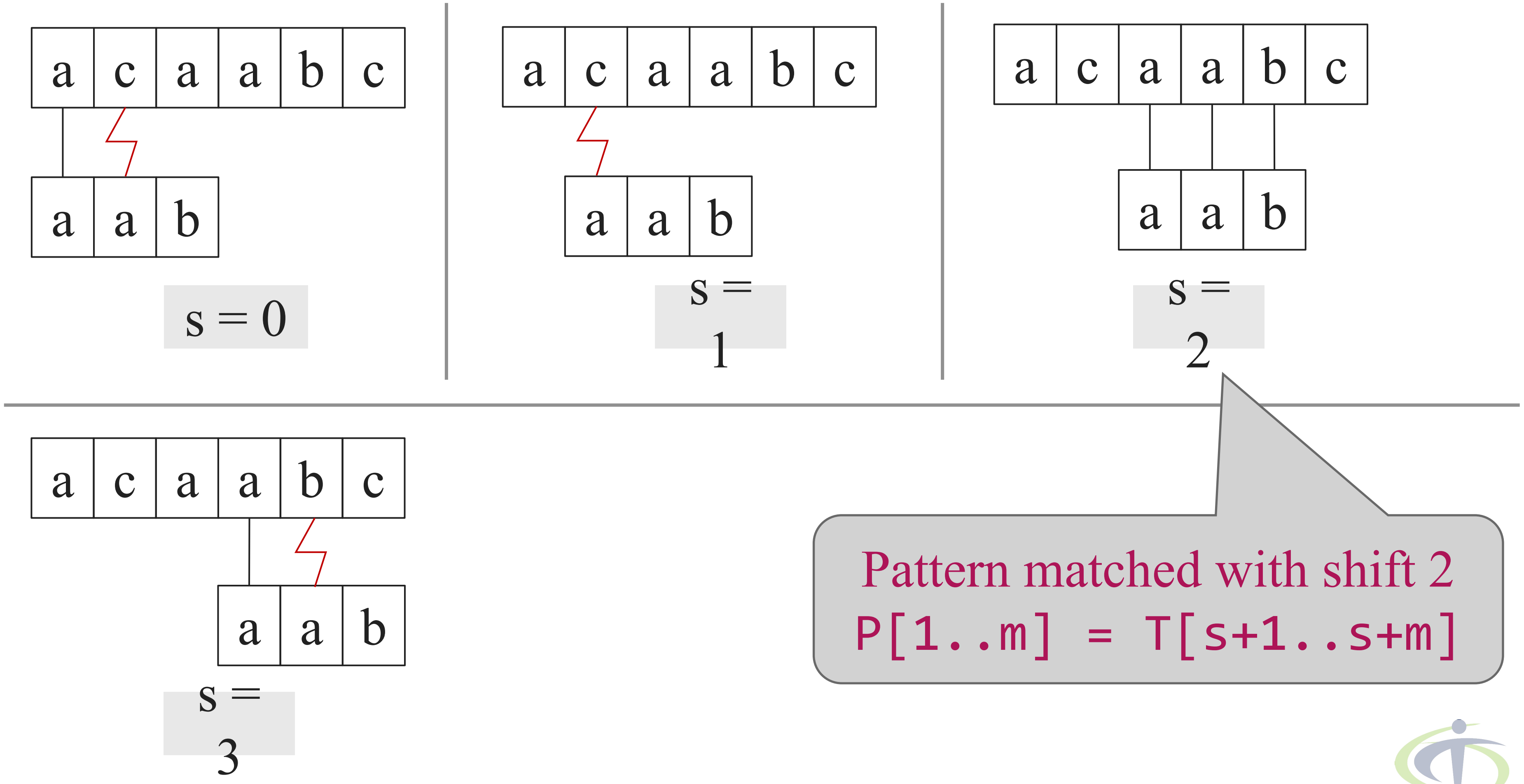
- Text-editing programs frequently need to find all occurrences of a pattern in the text.
- Efficient algorithms for this problem is called String-Matching Algorithms.
- Among its many applications, "String-Matching" is highly used in Searching for patterns in DNA and Internet search engines.
- Assume that the text is represented in the form of an array $T[1 \ldots n]$ and the pattern is an array $P[1 \ldots m]$.

Text T[1..13]	a	b	c	a	b	a	a	b	c	a	b	a	c
Pattern P[1..4]	a	b	a	a									

Naive String Matching Algorithm

Naive String Matching - Example

- The naive algorithm finds all valid shifts using a loop that checks the condition **P[1..m] = T[s+1..s+m]**



Naive String Matching - Algorithm

```
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
```

T[1..6]	a	c	a	a	b	c

P[1..3]	a	a	b	b	b	b

s = 3

Pattern occurs with shift 2

Naive String Matcher takes time O((n-m+1)m)

Rabin-Karp Algorithm

Text T

3	1	4	1	5	9	2	6	5	3	5

Pattern P

2	6

Choose a random prime number `q = 11`

Let, p `= P mod q`
`= 26 mod 11 = 4`

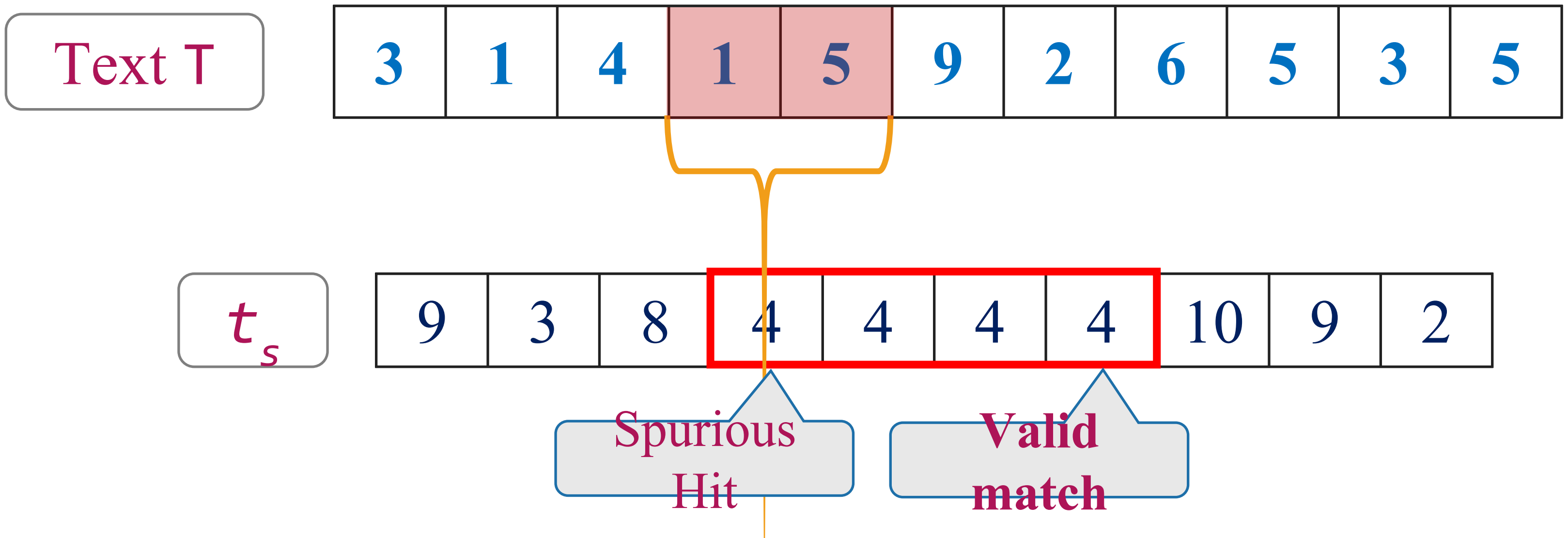
Let t_s denotes modulo `q` for text of length `m`

3	1	4	1	5	9	2	6	5	3	5

Pattern P

2	6

p = P mod q = 26 mod 11 = 4




```
if  t_s == p
    if P[1..m] == T[s+1..s+m]
       print "pattern occurs with shift" s
```

Rabin-Karp Algorithm

- We can compute t_s using following formula

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s + m + 1]$$

$\boxed{3}$	$\boxed{1}$	4	1	5	9	2	6	5	3	5

For m=2 and s=0 $t_s = 31$

We wish to remove higher order digit T[s+1]=3 and bring the new lower order digit T[s+m+1]=4

$$t_{s+1} = 10(31-10\cdot3) + 4$$
$$= 10(1) + 4 = \mathbf{14}$$

$$t_{s+2} = 10(14-10\cdot1) + 1$$
$$= 10(4) + 1 = \mathbf{41}$$

Rabin-Karp-Matcher

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

$n \leftarrow \text{length}[T];$

$m \leftarrow \text{length}[P];$

$h \leftarrow d^{m-1} \bmod q;$

$p \leftarrow 0;$

$t_0 \leftarrow 0;$

for $i \leftarrow 1$ to m do

  $p \leftarrow (d_p + P[i]) \bmod q$

  $t_0 \leftarrow (dt_0 + T[i]) \bmod q$

for $s \leftarrow 0$ to $n - m$ do

  if $p == t_s$ then

   if $P[1..m] == T[s+1..s+m]$ then

  print “pattern occurs with shift” s

  if $s < n\text{-}m$ then

   $t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$

T	3	1	4	1	5	9	2	6	5	3	5

P	2 6	d	10	q	11
n	11	m	2	h	10
p	0 4	t_0	0 9		

String Matching with Finite Automata

Introduction to Finite Automata

- Finite automaton (FA) is a simple machine, used to recognize patterns.
- It has a set of states and rules for moving from one state to another.
- It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
- At the time of transition, the automata can either move to the next state or stay in the same state.
- When the input string is processed successfully, and the automata reached its final state, then it will accept the input string.
- The string-matching automaton is very efficient: it examines each character in the text exactly once and reports all the valid shifts.

Introduction to Finite Automata

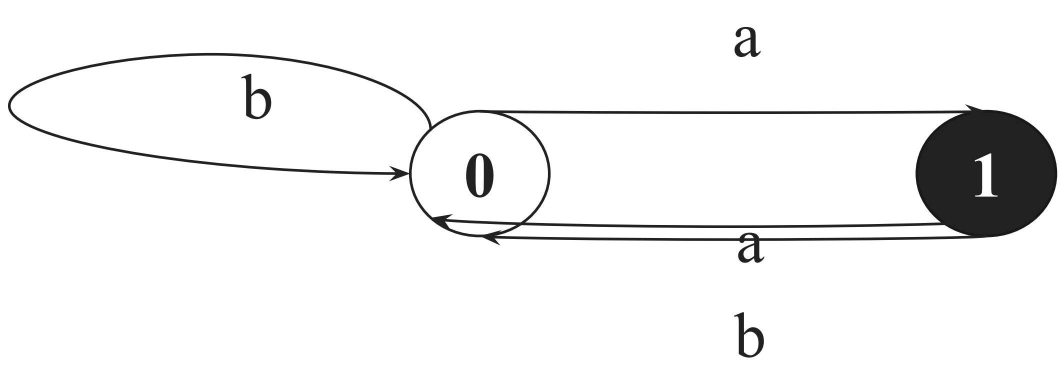
- A finite automaton M is a 5-tuple, which consists of,

$$(Q, q_0, A, \Sigma, \delta)$$

- Q is a finite set of **states**, $\quad Q = \{0,1\}$
- $q_0 \epsilon\ Q$ is a **start state**, $\quad q_0 = 0$
- $A \subseteq Q$ set of **accepting states**, $\quad A = \{1\}$
- Σ is a finite **input alphabet**, $\quad \Sigma = \{a, b\}$
- δ is a **transition function** of M. $\quad \delta(1, b) = 0$

	Input	
State	a	b
0	1	0
1	0	0

Transition Table



Finite Automaton

Suffix of String

- Suffix of a string is any number of trailing symbols of that string. If a string ω is a suffix of a string x then it is denoted by $\omega \sqsupset x$.

Compute Transition Function

COMPUTE-TRANSITION-FUNCTION(P, Σ)

$m \leftarrow length[P]$

for $q \leftarrow 0$ to m do

  for each character $\alpha \in \Sigma$ do

    $k \leftarrow \min(m + 1, q + 2)$

    repeat $k \leftarrow k - 1$ until $P_k \sqsupset P_q\alpha$

    $\delta(q, \alpha) \leftarrow k$

return δ

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

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

$m = 7$

for $q \leftarrow 0$ to m **do**
  for each character $\omega \in \Sigma$ **do**
    $k \leftarrow \min(m + 1, q + 2)$
    repeat $k \leftarrow k - 1$ **until** $P_k \sqsupset P_q\,\omega$
    $\delta(q, \omega) \leftarrow k$
return δ

input

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

q=0	ω=a	k=2	$P_2 \sqsupset P_0\omega$	ab ⊐ εa
		$\boxed{k=1}$	$P_1 \sqsupset P_0\omega$	a ⊐ εa
	ω=b	k=2	$P_2 \sqsupset P_0\omega$	ab ⊐ εb
		k=1	$P_1 \sqsupset P_0\omega$	a ⊐ εb
		$\boxed{k=0}$	$P_0 \sqsupset P_0\omega$	ε ⊐ εb
	ω=c	k=2	$P_2 \sqsupset P_0\omega$	ab ⊐ εc
		k=1	$P_1 \sqsupset P_0\omega$	a ⊐ εc
		$\boxed{k=0}$	$P_0 \sqsupset P_0\omega$	ε ⊐ εc

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

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

$m = 7$

for $q \leftarrow 0$ to m **do**
  for each character $\omega \in \Sigma$ **do**
    $k \leftarrow \min(m + 1, q + 2)$
    repeat $k \leftarrow k - 1$ **until** $P_k \sqsupset P_q\,\omega$
    $\delta(q, \omega) \leftarrow k$
return δ

input

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

q=1	ω=a	k=3	$P_3 \sqsupset P_1\omega$	aba $\sqsupset$ aa
		k=2	$P_2 \sqsupset P_1\omega$	ab $\sqsupset$ aa
		$\boxed{k=1}$	$P_1 \sqsupset P_1\omega$	a $\sqsupset$ aa
	ω=b	k=3	$P_3 \sqsupset P_1\omega$	aba $\sqsupset$ ab
		$\boxed{k=2}$	$P_2 \sqsupset P_1\omega$	ab $\sqsupset$ ab
	ω=c	k=3	$P_3 \sqsupset P_1\omega$	aba $\sqsupset$ ac
		k=2	$P_2 \sqsupset P_1\omega$	ab $\sqsupset$ ac
		k=1	$P_1 \sqsupset P_1\omega$	a $\sqsupset$ ac
		$\boxed{k=0}$	$P_0 \sqsupset P_1\omega$	ϵ $\sqsupset$ ac

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

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

$m = 7$

for $q \leftarrow 0$ to m **do**
 for each character $\omega \in \Sigma$ **do**
  $k \leftarrow \min(m + 1, q + 2)$
  repeat $k \leftarrow k - 1$ **until** $P_k \sqsupset P_q\,\omega$
  $\delta(q, \omega) \leftarrow k$
return δ

input

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

q=2	ω=a	k=4	$P_4 \sqsupset P_2\omega$	abab⊐aba
		k=3	$P_3 \sqsupset P_2\omega$	aba⊐aba
	ω=b	k=0	$P_0 \sqsupset P_2\omega$	ϵ⊐abb
	ω=c	k=0	$P_0 \sqsupset P_2\omega$	ϵ⊐abc

q=3	ω=a	k=1	$P_1 \sqsupset P_3\omega$	a⊐abaa
	ω=b	k=4	$P_4 \sqsupset P_3\omega$	abab⊐abab
	ω=c	k=0	$P_0 \sqsupset P_3\omega$	ϵ⊐abac

Finite Automata Matcher

FINITE-AUTOMATON MATCHER(T, δ, m)
n ← length[T]
q ← 0
for i ← 1 **to** n **do**
    q ← δ(q, T[i])
    if q == m **then**
    print "Pattern occurs with shift" i – m

i = 7

q = 5

$q = \delta(0, a) = 1$
$q = \delta(1, b) = 2$
$q = \delta(2, a) = 3$
$q = \delta(3, b) = 4$
$q = \delta(4, a) = 5$
$q = \delta(5, b) = 4$
$q = \delta(4, a) = 5$

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	4	6
6	7	0	0
7	1	2	0

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

	1	2	3	4	5	6	7	8	9	10	11
Text	a	b	a	b	a	b	a	c	a	b	a

Finite Automata Matcher

FINITE-AUTOMATON MATCHER(T, δ, m)
n ← length[T]
q ← 0
for i ← 1 **to** n **do**
  q ← δ(q, T[i])
  if q == m **then**
  print "Pattern occurs with shift" i – m

i = 9

q = 7

$q = \delta(0, a) = 1$
$q = \delta(1, b) = 2$
$q = \delta(2, a) = 3$
$q = \delta(3, b) = 4$
$q = \delta(4, a) = 5$
$q = \delta(5, b) = 4$
$q = \delta(4, a) = 5$
$q = \delta(5, c) = 6$
$q = \delta(6, a) = 7$

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

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

	1	2	3	4	5	6	7	8	9	10	11
Text	a	b	a	b	a	b	a	c	a	b	a

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Suffix & Prefix of a String

Suffix of a string	

Prefix of a string	

String Matching with Knuth-Morris-Pratt Algorithm

Introduction

- The KMP algorithm relies on prefix function (π).
- Proper prefix: All the characters in a string, with one or more cut off the end. “S”, “Sn”, “Sna”, and “Snap” are all the proper prefixes of “Snape”.
- Proper suffix: All the characters in a string, with one or more cut off the beginning. “agrid”, “grid”, “rid”, “id”, and “d” are all proper suffixes of “Hagrid”.
- KMP algorithm works as follows:
  - Step-1: Calculate Prefix Function
  - Step-2: Match Pattern with Text

Longest Common Prefix and Suffix

	1	2	3	4	5	6	7
Pattern	a	b	a	b	a	c	a
Prefix(π)							

ababa

Possible prefix = a, ab, aba, abab

Possible suffix = a, ba, aba, baba

Calculate Prefix Function - Example

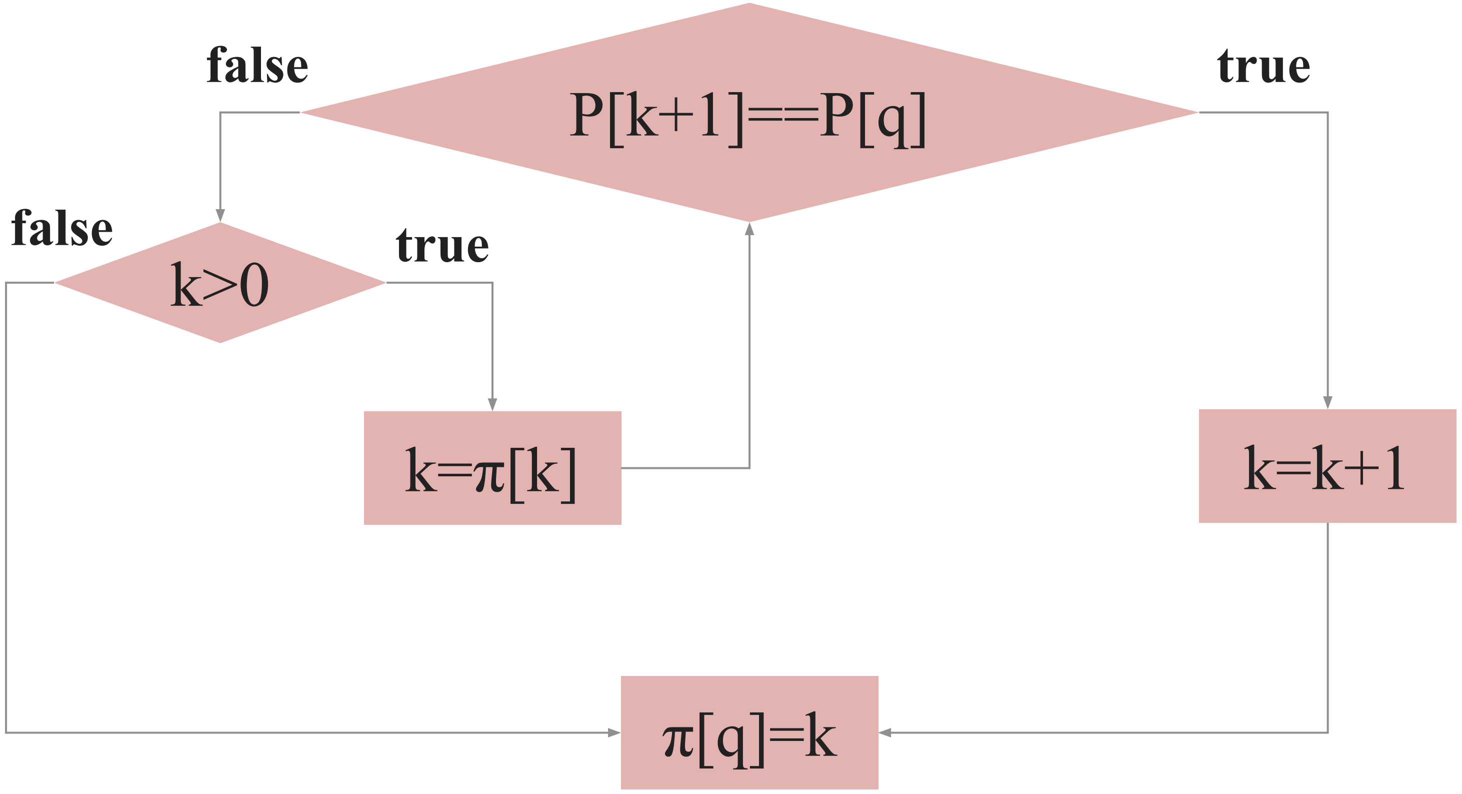
k+1 q q

	1	2	3	4	5	6	7
P	a	c	a	c	a	g	t
π							

k = **0**

q = **7**

Initially set **π[1] = 0**
k is the longest prefix found
q is the current index of pattern



KMP- Compute 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]
        end while
        if P[k + 1] == P[q] then
            k ← k + 1
        end if
        π[q] ← k
    return π
```

KMP String Matching

	1	2	3	4	5	6	7
Pattern	a	c	a	c	a	g	t
Prefix(π)	0	0	1	2	3	0	0

T	a	c	a	t	a	c	g	a	c	a	c	a	g	t
	a	c	a	c	a	g	t							
✗		a	c	a	c	a	g	t						

Mismatch ?
Check value in prefix table

We can skip 2 shifts
(Skip unnecessary shifts)

T	a	c	a	t	a	c	g	a	c	a	c	a	g	t
			a	c	a	c	a	g	t					

Mismatch ?
Check value in prefix table

T	a	c	a	t	a	c	g	a	c	a	c	a	g	t
				a	c	a	c	a	g	t				

Mismatch ?
Check value in prefix table

KMP String Matching

	1	2	3	4	5	6	7
Pattern	a	c	a	c	a	g	t
Prefix(π)	0	$\boxed{0}$	1	2	3	0	0

T	a	c	a	t	a	c	g	a	c	a	c	a	g	t

a	c	a	c	a	g	t

Mismatch ?
Check value in prefix table

We can skip 2 shifts
(Skip unnecessary shifts)

T	a	c	a	t	a	c	g	a	c	a	c	a	g	t

a	c	a	c	a	g	t

T	a	c	a	t	a	c	g	a	c	a	c	a	g	t

a	c	a	c	a	g	t

Pattern matches with shift $i - m$

KMP-MATCHER

```
KMP-MATCHER(T, P)
n ← length[T]
m ← length[P]
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] then
        then q ← q + 1       //Next character matches.
    if q == m then           //Is all of P matched?
        print "Pattern occurs with shift" i - m
        q ← π[q]    //Look for the next match.
```

Thank You!