Parul® University

NAAC A++

String Matching & NP Completeness
Chapter-7

Mrs. Bhumi Shah
Assistant Professor
Department of Computer Science and Engineering

Parul® University

NAAC A++

Content

1. String Matching:
- Introduction to String Matching,
- Naive String Matching,
- Rabin-Karp Algorithm,

2. Kruth-Morris-Pratt Algorithm,
- String Matching using Finite Automata

3. NP Completeness:
- Introduction to NP Completeness,
- P class Problems,
- NP Class Problems,
- Hamiltonian Cycle

INDEX

Introduction to String Matching

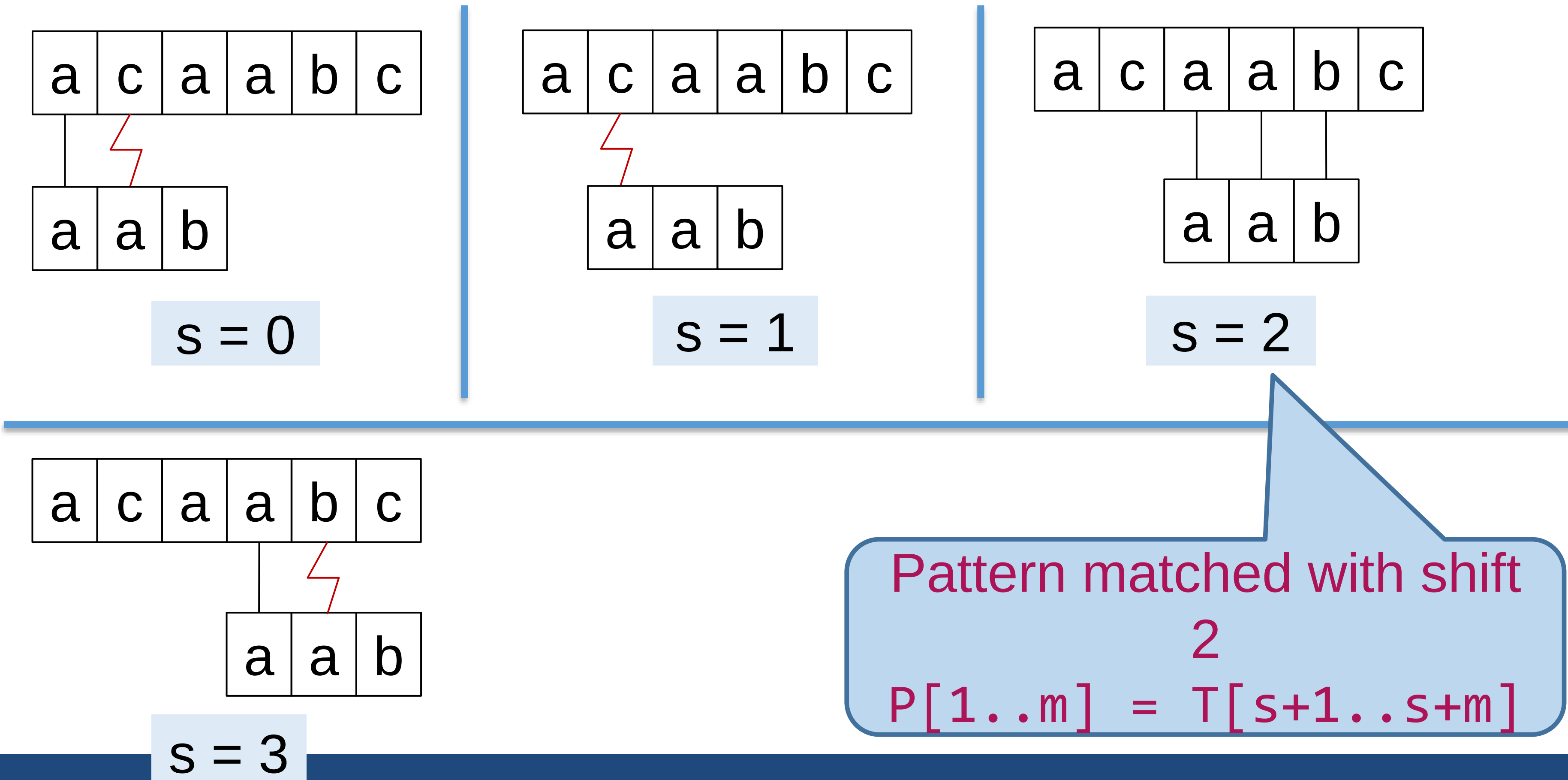
- Text-editing programs frequently need to find all occurrences of a pattern in the text.
- Efficient algorithms for this problem is called String-Matching Algorithms.
- Among its many applications, “String-Matching” is highly used in Searching for patterns in DNA and Internet search engines.
- Assume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

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

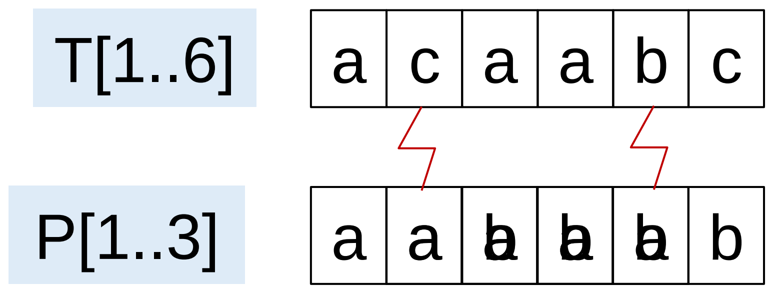
- The naive algorithm finds all valid shifts using a loop that checks the condition **P[1..m] = T[s+1..s+m]**



Naive String Matching

```
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
```

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



s = 3

Pattern occurs with shift 2

Rabin-Karp Algorithm

RABIN-KARP-MATCHER(T, P, d, q)
- $n \leftarrow$ length[T];
- $m \leftarrow$ 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	t_0	0		

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

Text T

3	1	4	1	5	9	2	6	5	3	5

t_s

9	3	8	4	4	4	4	10	9	2

Spurious Hit

Valid match

```
if  ts == p
    if P[1..m] == T[s+1..s+m]
       print “pattern occurs with shift” s
```

- We can compute t_s using following formula

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

3	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

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

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

String Matching with 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.

String Matching with 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**,

$q_0 \in Q$ is a **start state**,

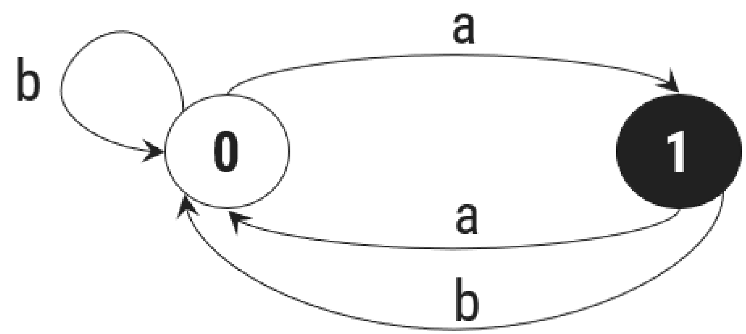
$A \subseteq Q$ set of **accepting states**,

Σ is a finite **input alphabet**,

δ is a **transition function** of M.

	Input	
State	a	b
0	1	0
1	0	0

Transition Table



Finite Automaton

String Matching with Finite Automata

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

$P = ababa$	

Compute Transition Function

COMPUTE-TRANSITION-FUNCTION(P, Σ)
m ← length[P]
for q ← 0 to m do
  for each character α Є Σ do
    k ← min(m + 1, q + 2)
    repeat k ← k – 1 until $P_k \sqsupset P_q\alpha$
    δ(q, α) ← 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 ← 0 to m **do**
  for each character ω ∈ Σ **do**
    k ← min(m + 1, q + 2)
    repeat k ← k – 1 **until** $P_k \sqsupset P_q\,\omega$
    δ(q, ω) ← 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 ← 0 to m **do**
  for each character ω Є Σ **do**
    k ← min(m + 1, q + 2)
    repeat k ← k – 1 **until** $P_k \sqsupset P_q\,\omega$
    δ(q, ω) ← 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⊐aa
		k=2	$P_2 \sqsupset P_1\omega$	ab⊐aa
		k=1	$P_1 \sqsupset P_1\omega$	a⊐aa
	ω=b	k=3	$P_3 \sqsupset P_1\omega$	aba⊐ab
		k=2	$P_2 \sqsupset P_1\omega$	ab⊐ab
	ω=c	k=3	$P_3 \sqsupset P_1\omega$	aba⊐ac
		k=2	$P_2 \sqsupset P_1\omega$	ab⊐ac
		k=1	$P_1 \sqsupset P_1\omega$	a⊐ac
		k=0	$P_0 \sqsupset P_1\omega$	ϵ⊐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 ← 0 to m **do**
  for each character ω ∈ Σ **do**
    k ← min(m + 1, q + 2)
    repeat k ← k – 1 **until** $P_k \sqsupset P_q\,\omega$
    δ(q, ω) ← 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-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	a	b	c
0	$\boxed{1}$	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

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
```

$i = 9$

$q = 7$

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

input

State	a	b	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

Suffix of a string	
$P = string$	
$P_1 = g$	$P_1 \sqsupset P$
$P_2 = ng$	$P_2 \sqsupset P$
$P_3 = ing$	$P_3 \sqsupset P$
$P_4 = ring$	$P_4 \sqsupset P$
$P_5 = tring$	$P_5 \sqsupset P$

Prefix of a string	
$P = string$	
$P_1 = s$	$P_1 \sqsubset P$
$P_2 = st$	$P_2 \sqsubset P$
$P_3 = str$	$P_3 \sqsubset P$
$P_4 = stri$	$P_4 \sqsubset P$
$P_5 = strin$	$P_5 \sqsubset P$

Parul® University | NAAC GRADE A++

https://paruluniversity.ac.in/

Parul® University

NAAC A++

String Matching & NP Completeness
Chapter-7

Mrs. Bhumi Shah
Assistant Professor
Department of Computer Science and Engineering

Parul® University

NAAC A++

Content

1. String Matching:
- Introduction to String Matching,
- Naive String Matching,
- Rabin-Karp Algorithm,

2. Kruth-Morris-Pratt Algorithm,
- String Matching using Finite Automata

3. NP Completeness:
- Introduction to NP Completeness,
- P class Problems,
- NP Class Problems,
- Hamiltonian Cycle

INDEX

Kruth-Morris-Pratt Algorithm

- 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)	0	0	1	2	3	0	1

ababa

Possible prefix = a, ab, aba, abab

Possible suffix = a, ba, aba, baba

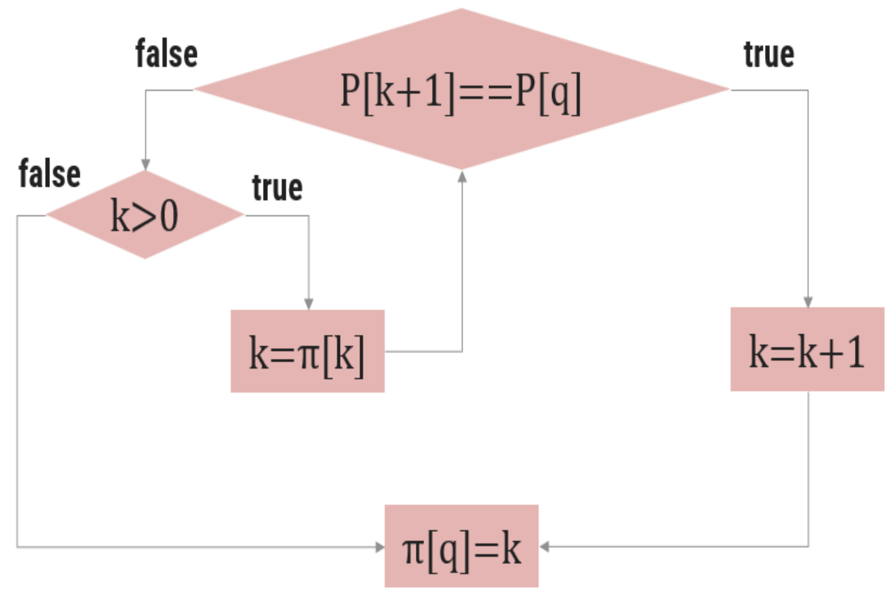
Calculate Prefix Function - Example

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

$k = \mathbf{0}$

$q = \mathbf{7}$

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

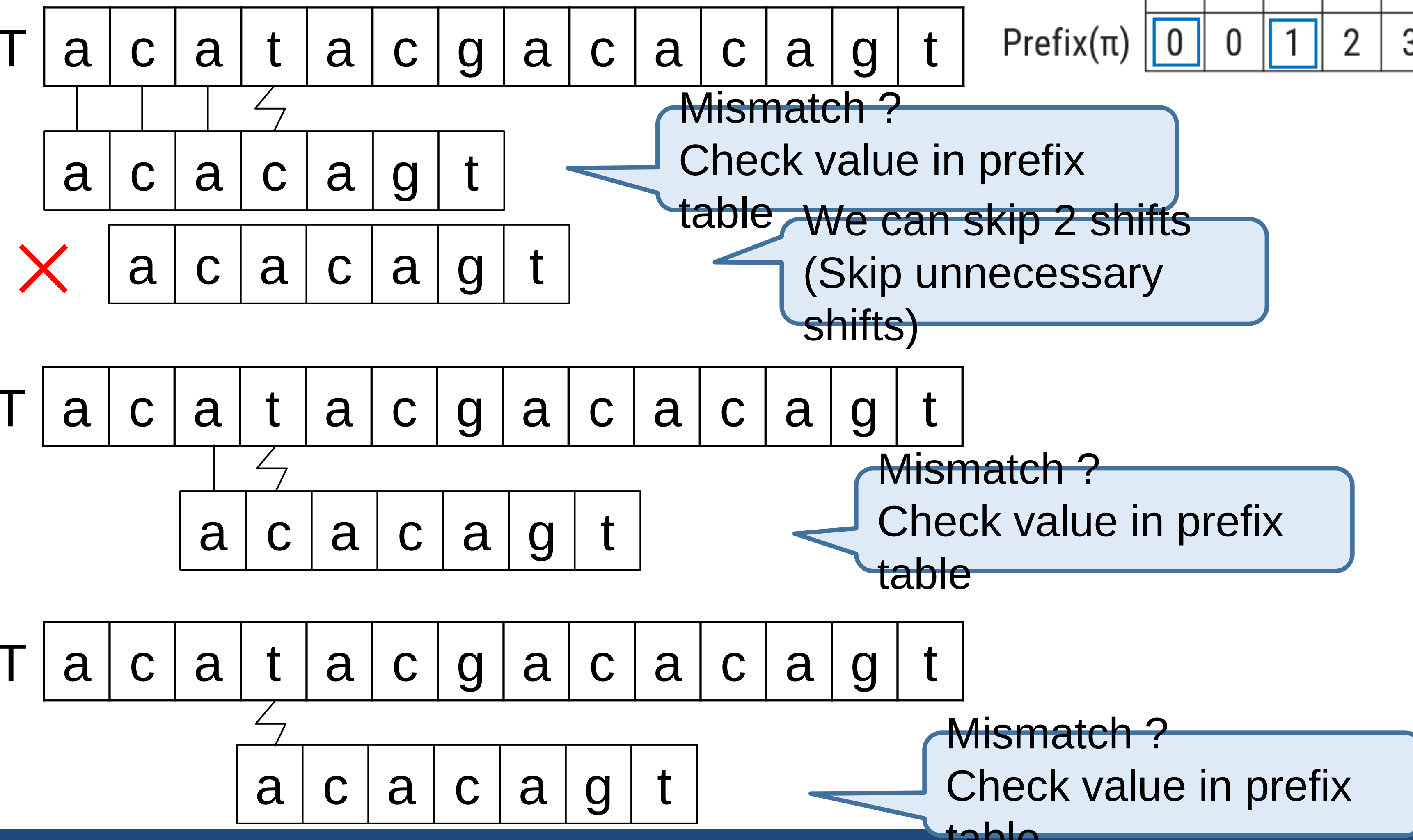


KMP- Compute Prefix Function

```
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.
```

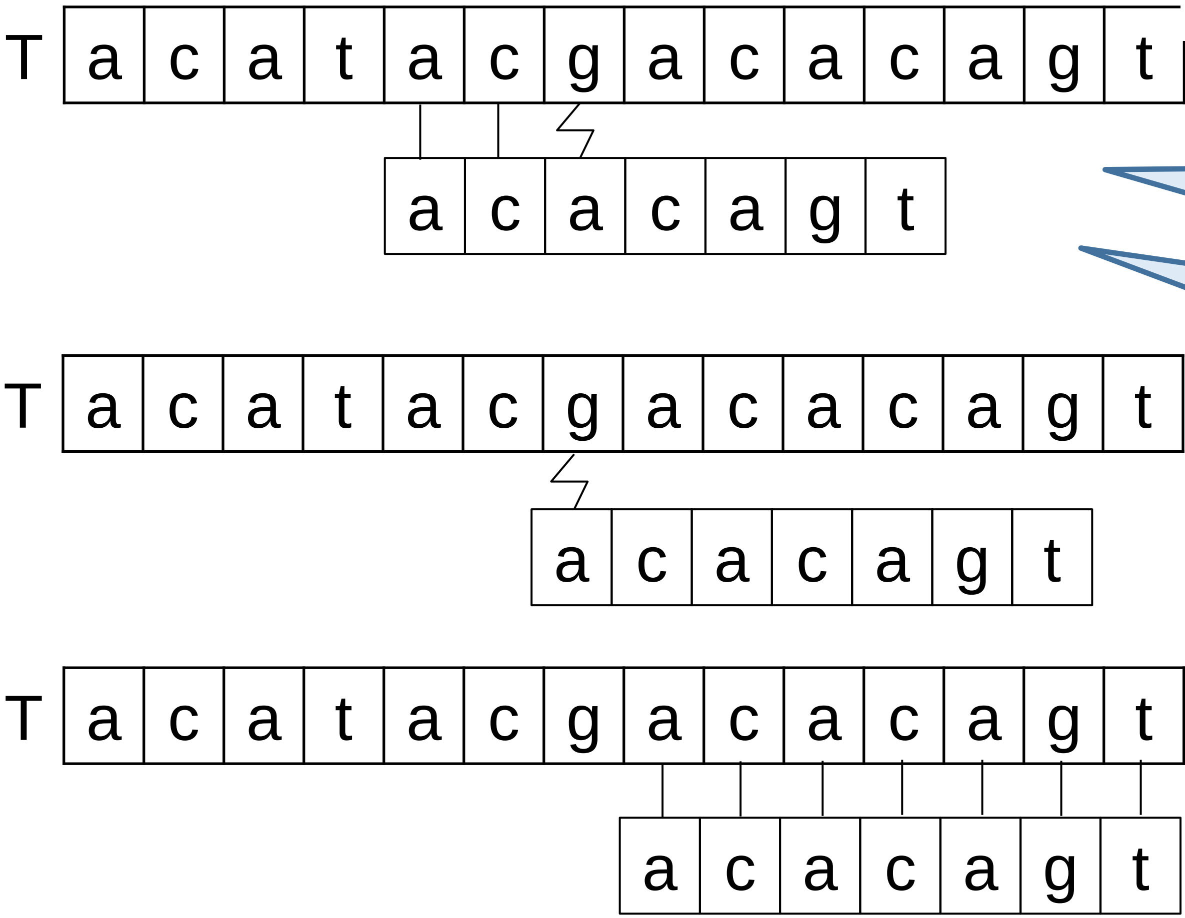
KMP- Compute Prefix Function

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



KMP- Compute Prefix Function

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



Mismatch ?
Check value in prefix table

We can skip 2 shifts
(Skip unnecessary shifts)

Pattern matches with shift i –

KMP- Compute Prefix Function

```
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.
```

String Matching with 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.

String Matching with 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**,

$q_0 \epsilon\, Q$ is a **start state**,

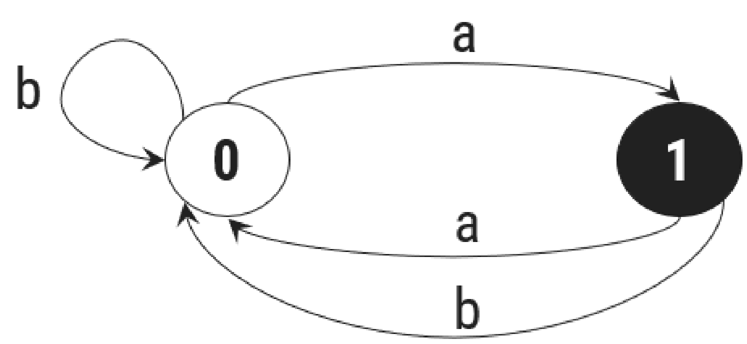
$A \subseteq Q$ set of **accepting states**,

Σ is a finite **input alphabet**,

δ is a **transition function** of M.

	Input	
State	a	b
0	1	0
1	0	0

Transition Table



Finite Automaton

String Matching with Finite Automata

- 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 $\omega \sqsupset x$.

$P = ababa$	

Compute Transition Function

COMPUTE-TRANSITION-FUNCTION(P, Σ)
m ← length[P]
for q ← 0 to m do
  for each character α Є Σ do
    k ← min(m + 1, q + 2)
    repeat k ← k – 1 until $P_k \sqsupset P_q\alpha$
    δ(q, α) ← 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 ← 0 to m **do**
  for each character ω ∈ Σ **do**
    k ← min(m + 1, q + 2)
    repeat k ← k – 1 **until** $P_k \sqsupset P_q\,\omega$
    δ(q, ω) ← 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
		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
		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
		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 ← 0 to m **do**

for each character ω ∈ Σ **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⊐aa
		k=2	$P_2 \sqsupset P_1\omega$	ab⊐aa
		$\boxed{k=1}$	$P_1 \sqsupset P_1\omega$	a⊐aa
	ω=b	k=3	$P_3 \sqsupset P_1\omega$	aba⊐ab
		$\boxed{k=2}$	$P_2 \sqsupset P_1\omega$	ab⊐ab
	ω=c	k=3	$P_3 \sqsupset P_1\omega$	aba⊐ac
		k=2	$P_2 \sqsupset P_1\omega$	ab⊐ac
		k=1	$P_1 \sqsupset P_1\omega$	a⊐ac
		$\boxed{k=0}$	$P_0 \sqsupset P_1\omega$	ϵ⊐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 ← 0 to m **do**
  for each character ω ∈ Σ **do**
    k ← min(m + 1, q + 2)
    repeat k ← k − 1 **until** $P_k \sqsupset P_q\omega$
    δ(q, ω) ← 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-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	$\boxed{1}$	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

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
```

$i = 9$

$q = 7$

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

input

State	a	b	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

Suffix of a string	
$P = string$	
$P_1 = g$	$P_1 \sqsupset P$
$P_2 = ng$	$P_2 \sqsupset P$
$P_3 = ing$	$P_3 \sqsupset P$
$P_4 = ring$	$P_4 \sqsupset P$
$P_5 = tring$	$P_5 \sqsupset P$

Prefix of a string	
$P = string$	
$P_1 = s$	$P_1 \sqsubset P$
$P_2 = st$	$P_2 \sqsubset P$
$P_3 = str$	$P_3 \sqsubset P$
$P_4 = stri$	$P_4 \sqsubset P$
$P_5 = strin$	$P_5 \sqsubset P$

Parul® University

NAAC GRADE A++

https://paruluniversity.ac.in/

Parul® University

NAAC A++

String Matching & NP Completeness
Chapter-7

Mrs. Bhumi Shah
Assistant Professor
Department of Computer Science and Engineering

Parul® University

NAAC A++

Content

1. String Matching:
- Introduction to String Matching,
- Naive String Matching,
- Rabin-Karp Algorithm,

2. Kruth-Morris-Pratt Algorithm,
- String Matching using Finite Automata

3. NP Completeness:
- Introduction to NP Completeness,
- P class Problems,
- NP Class Problems,
- Hamiltonian Cycle

INDEX

NP completeness

- is a foundational concept in computational complexity theory that addresses the classification of problems based on their solvability and the resources required to solve them.
- This framework helps categorize problems into manageable classes and provides insight into the limits of algorithmic solutions.
- The terms P, NP, and NP-complete are critical in understanding this complex web of computational challenges.

The Class P

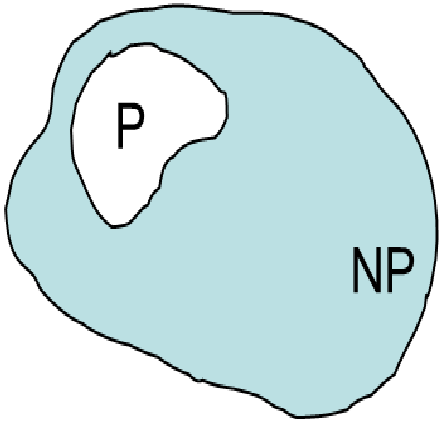
- The class P consists of those problems that are solvable in polynomial time by deterministic algorithms.
- More specifically, they are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem.
- For example, $O(n^3)$, $O(n^4)$, $O(logn)$, Fractional Knapsack, MST, Sorting algorithms etc…
- P is a complexity class that represents the set of all decision problems that can be solved in polynomial time.
- That is, given an instance of the problem, the answer yes or no can be decided in polynomial time.

The NP class

- NP is Non-Deterministic polynomial time.
- The class NP consists of those problems that are verifiable in polynomial time.
- NP is the class of decision problems for which it is easy to check the correctness of a claimed answer, with the help of a little extra information.
- Hence, we are not asking for a way to find a solution, but only to verify that an alleged solution really is correct.
- Every problem in this class can be solved in exponential time using exhaustive search.

P and NP Class Problems

- P = set of problems that **can be solved** in polynomial time
- NP = set of problems for which a solution **can be verified** in polynomial time
- $P \subseteq NP$



Classification of NP Problems

NP Complete

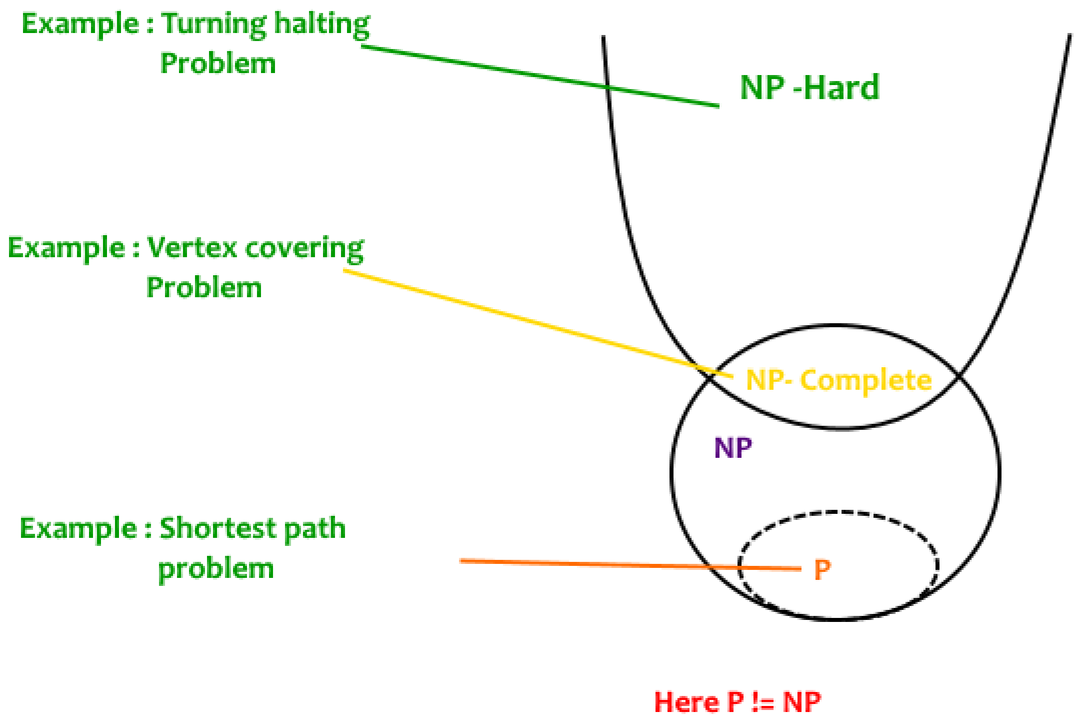
- NP-complete problems are a set of problems to each of which any other NP-problem can be reduced in polynomial time, and whose solution may still be verified in polynomial time.
- No polynomial-time algorithm has been discovered for an NP-Complete problem.
- NP-Complete is a complexity class which represents the set of all problems X in NP for which it is possible to reduce any other NP problem Y to X in polynomial time.

Classification of NP Problems

NP Hard

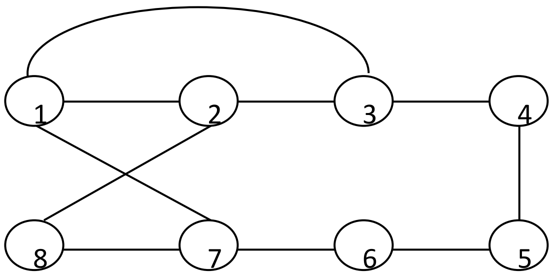
- NP-hard problems are those at least as hard as NP problems, i.e., all NP problems can be reduced (in polynomial time) to them.
- NP-hard problems need not be in NP, i.e., **they need not have solutions verifiable in polynomial time.**
- *The precise definition here is that a problem X is NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time.*

P, NP Complete and NP Hard



Hamiltonian Cycles

- Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once.
- A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian Path.



The graph has Hamiltonian cycles:

1, 3, 4, 5, 6, 7, 8, 2, 1 and 1, 2, 8, 7, 6, 5, 4, 3, 1.

- Given a list of vertices and to check whether it forms a Hamiltonian cycle or not:
- Counts the vertices to make sure they are all there, then checks that each is connected to the next by an edge, and that the last is connected to the first.

Hamiltonian Cycles

- It takes time proportional to n, because there are n vertices to count and n edges to check. n is a polynomial, so the check runs in polynomial time.
- To find a Hamiltonian cycle from the given graph: There are $n!$ different sequences of vertices that might be Hamiltonian paths in a given n-vertex graph, so a brute force search algorithm that tests all possible sequences can not be solved in polynomial time.
- In the traveling salesman Problem, a salesman must visits n cities.
- We can say that salesman wishes to make a tour or Hamiltonian cycle, visiting each city exactly once and finishing at the city he starts from.

Parul® University

NAAC GRADE A++

https://paruluniversity.ac.in/