AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

ONE SHOT Revision

(Crash Course)

By Dr. Nidhi Parashar Ma'am

- M.Tech Gold Medalist
- Net Qualified

Unit-5 : Selected Topics

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-5: Selected Topics

Lecture-1

Today's Target

- String Matching
  - Naïve string matching
  - Rabin Karp algorithm
- AKTU PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

String Matching Problem

Given a text string T[1...n] and a pattern string P[1...m], the goal is to find all occurrences of the pattern P within the text T. The elements of T and P are characters from an alphabet Σ.

$\Sigma = \{0, 1\}$ or $\{a, b, c, \ldots z\}$.

"String matching problem is to find all the occurrence of P in T."

- A pattern P occurs with a shift s in T,

if P[1...m] is = T [s + 1...s + m]

- The string-matching problem is to find all values of s where

$$0 \leq s \leq n - m$$

(AKTU) (Naive String Matching)

	1	2	3	4	5	6	7	8	9	10
T:	b	a	c	a	b	c	a	b	c	a

$n = 10$

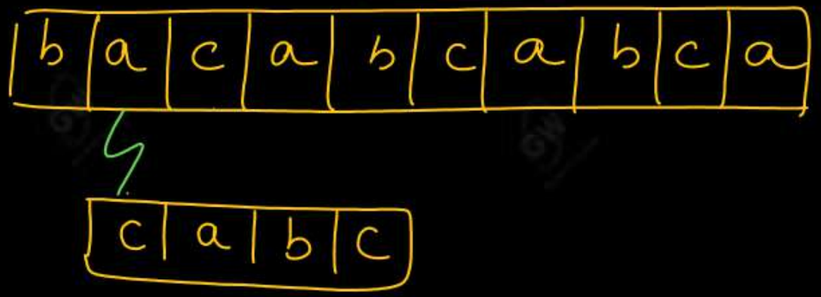
	1	2	3	4
P:	c	a	b	c

$m = 4$

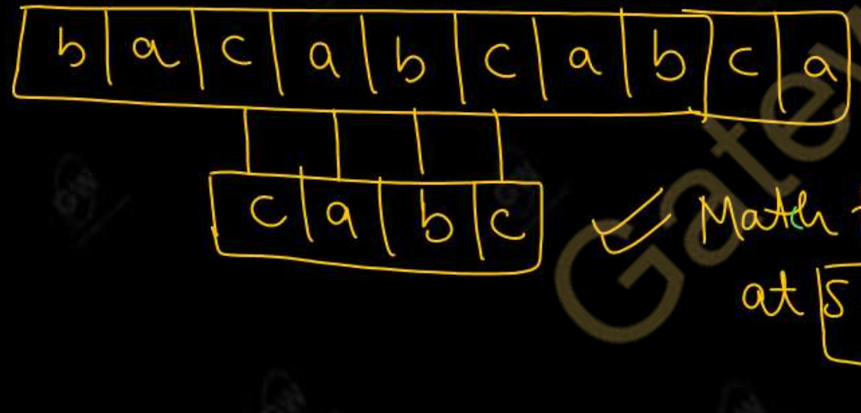
$0 \le s \le n - m = 6$

S = 0

S = 1

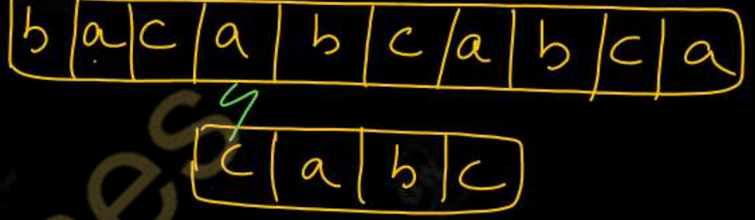


S = 2

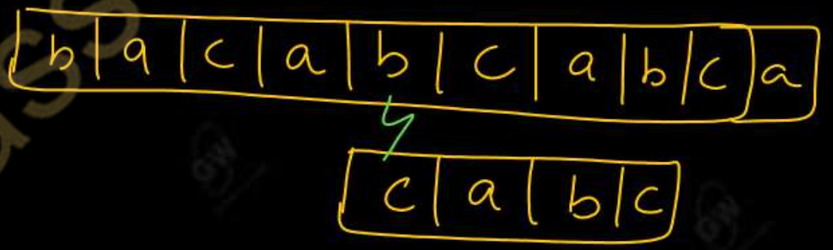


✓ Match found at S = 2

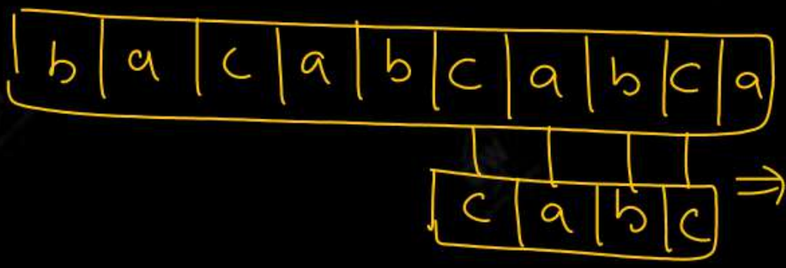
S = 3



S = 4

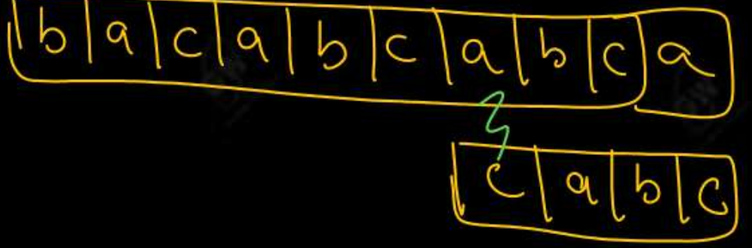


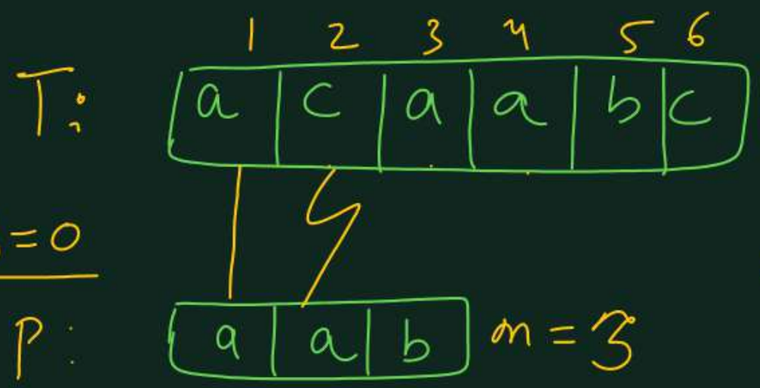
S = 5



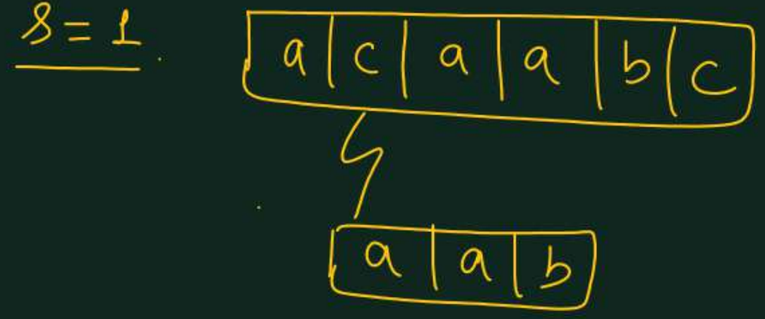
⇒ Match found at S = 5

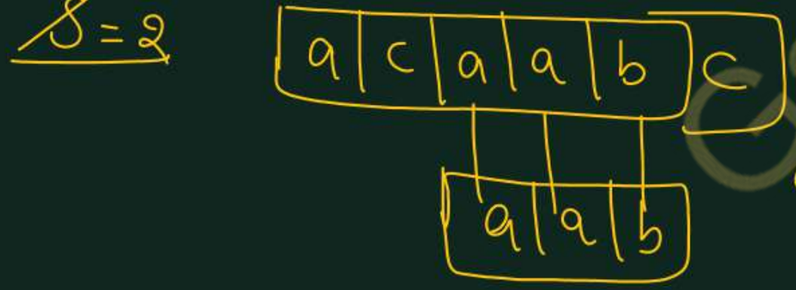
S = 6



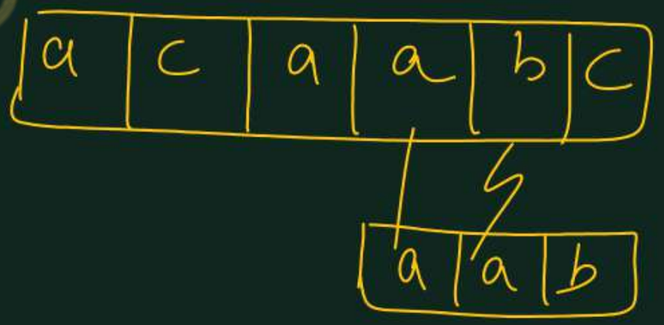


$s = 0 \cdots 3$





at $s=0$ pattern is found



Naive (Brute Force) Algorithm

Initially P is aligned with T at the first index position. P is then compared with T from left –to – right. If a mismatch occurs, "Slide P" to right by 1 position and start the comparison again."

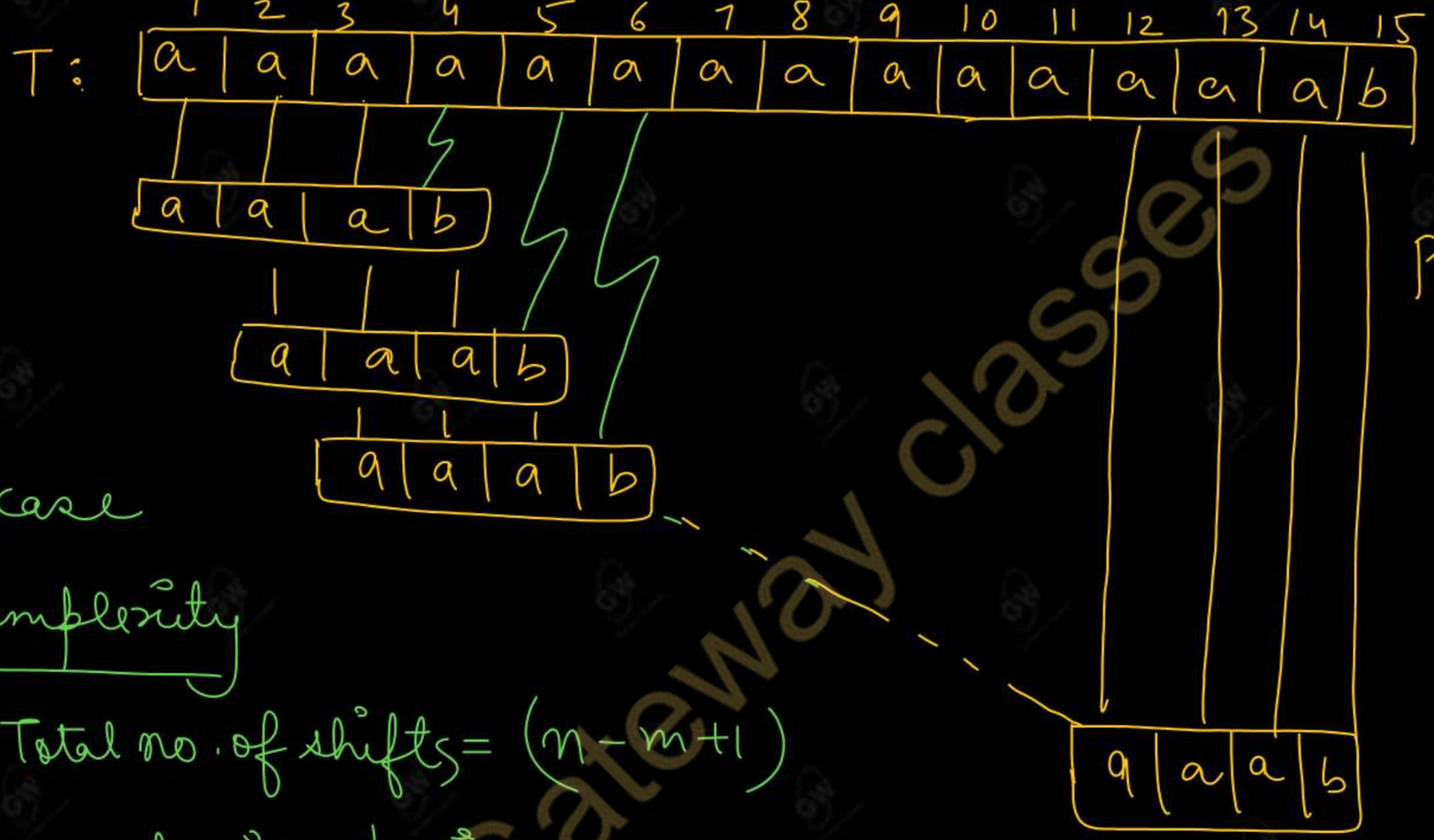
Pseudo code -

Text

pattern

```
Naïve String Matcher (T, P)
1.      n ← T. length
2.      m ← P. length
3.      for s ← 0 to n-m
4.              if P[1...m] == T[s+1...s + m]
5.                      print "Pattern occurs with shift" s
```

Naive Algorithm (Worst case analysis)



Worst case

Time Complexity

Total no. of shifts = $(n-m+1)$

No. of comparisons / shift = m

Total Time taken = $(n-m+1)m$ $\Rightarrow$ $O(mn)$

Rabin-Karp Algorithm [V. Imp for AKTU]

- It calculates a hash value for the pattern as well as for each m-character subsequence of text to be compared.
- If hash values are unequal, the algorithm will calculate the hash value for next m-character sequence in the text .
- If hash values are equal, the algorithm will do a brute force comparison(naïve algorithm procedure) b/w the pattern and m-character sequence in the text.
- So, there is only one comparison per text subsequence and brute force is only needed when hash values match.
- The algorithm compares strings' hash values rather than the strings themselves.
- For efficiency, the hash value of the next position in the text is easily computed from the hash value of the current position.

Basic Idea of Rabin-Karp

T: | a | b | c | d | a | b | c | e | $n=8$

P: | b | c | e | $m=3$

$0 \le s \le n-m$

$0 \le s \le 5$

$P_h = 2+3+5$

$P_h = 10$

a — 1
b — 2
c — 3
d — 4
e — 5
f — 6
g — 7
h — 8
i — 9
j — 10

s = 0

1 + 2 + 3 ⇓ 7

s = 1

2 + 3 + 4 ⇓ 9

s = 2

3 + 4 + 1 ⇓ 8

s = 3

4 + 1 + 2 ⇓ 7

s = 4

1 + 2 + 3 ⇓ 6

s = 5

2 + 3 + 5 ⇓ 10

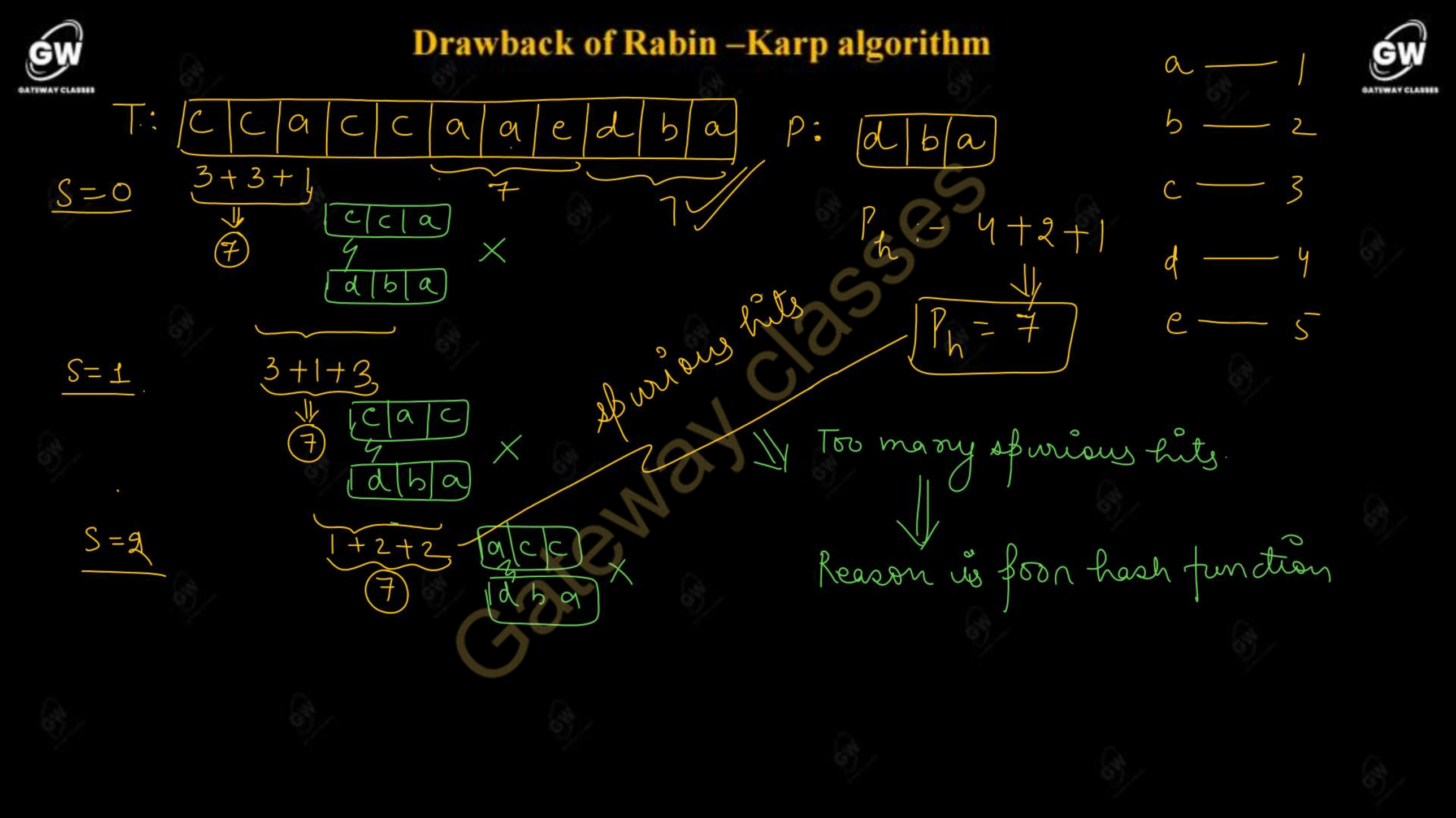
s = 5

| a | b | c | d | a | b | c | e |

hash values does not match with the hash value of pattern

| b | c | e |

Drawback of Rabin –Karp algorithm



Spurious hits

When the hash value of m-character subsequence of text is same as the hash value of pattern, but the pattern does not match with the m-character subsequence of the text. Such hits of equal hash values are called spurious hits.

no. of shifts = $(n-m+1)m$

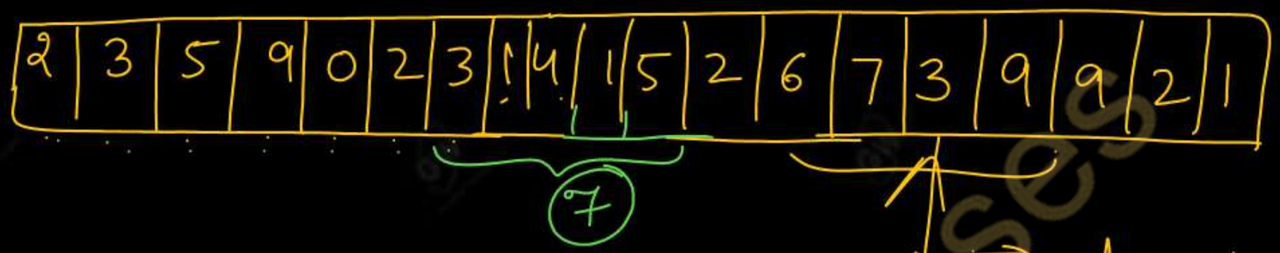
In worst case $\Rightarrow$ $O(mn)$

needs to be minimized.

Average case $\Rightarrow$ $O(n-m+1)$

Calculating Hash values –

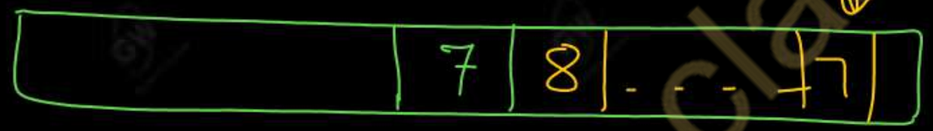
$q = 13$



P: 3 1 4 1 5

$P_h \Rightarrow 7$

$\Rightarrow$ spurious hit



$$(31415 - 30000) \times 10 + 2$$

$$= 1415 \times 10 + 2$$

$$= 14152 \bmod 13$$

Ques. - For q=11, how many valid and spurious hits are found for the given text and pattern?

P: 2 6 (1 2)

m=2

$P_h = 26 \bmod 11$

4 ✓

Total no. of hits = 4

valid hits = 1

spurious hits = 3

Rabin-Karp-Matcher (T, P, d, q)

prime no.

d → Base value

d = 10

```
n ← length[T]
m ← length[P]
h ← d^(m-1) mod q
p ← 0
t0 ← 0
for i ← 1 to m
    p ← (dp + P[ i ]) mod q
    t0 ← (dt0 + T[ i ]) mod q
for s ← 0 to n-m
    if p = ts
        then if P[1...m] = T[s+1... s+m]
                then print "Pattern occurs with shift" s.
    if s < n-m
        ts+1 ← (d (ts - T[s+1]h) + T[s+m+1]) mod q
```

Pattern hash value ← p

Text hash value at s=0 ← t_0

i = 1

i = 2

n = 16, m = 2

$h = (10)^1 \bmod 11 \Rightarrow 10$

h = 10

$p = (10 \times 0 + 2) \bmod 11$

p = 2

$t_0 = (10 \times 0 + 3) \bmod 11$

$t_0 = 3$

$p = (10 \times 2 + 6) \bmod 11 \Rightarrow 4$

$t_0 = (10 \times 3 + 1) \bmod 11 \Rightarrow 9$

p = 4 ⇒ hash value of pattern

$t_0 = 9$ ⇒ hash value of first 2-charath of text at s=0

```
Rabin-Karp-Matcher (T, P, d, q)
n ← length[T]
m ← length[P]
h ← d^(m-1) mod q
p ← 0
t0 ← 0
for i ← 1 to m
        p ← (dp + P[ i ]) mod q
        t0 ← (dt0 + T[ i ]) mod q
for s ← 0 to n-m
        if p = ts
                then if P[1...m] = T[s+1... s+m]
                        then print "Pattern occurs with shift" s.
        if s < n-m
                ts+1 ← (d (ts - T[s+1]h) + T[s+m+1]) mod q
```

for $s=0$ $p \neq t_0$ as $4 \neq 9$

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

$$t_1 = \left(10(9 - T[0+1].10) + T[0+2+1] \right) \bmod q.$$

$$= (10(9 - 3.10) + 4) \bmod 11$$

$$= (10(-21) + 4) \bmod 11$$

$$= -206 \bmod 11$$

$$= (-206 - (-209)) \bmod 11$$

$$\Rightarrow (-206 + 209) \bmod 11$$

$3 \bmod 11$

$t_1 \Rightarrow 3$

we need to find the largest number which is less than -206 and divisible by 11.

⇓

subtract that no from -206

Rabin-Karp-Matcher (T, P, d, q)

```
n ← length[T]
m ← length[P]
h ← d^(m-1) mod q
p ← 0
t0 ← 0
for i ← 1 to m
        p ← (dp + P[ i ]) mod q
        t0 ← (dt0 + T[ i ]) mod q
for s ← 0 to n-m
        if p = ts
                then if P[1...m] = T[s+1... s+m]
                        then print "Pattern occurs with shift" s.
        if s < n-m
                ts+1 ← (d (ts - T[s+1]h) + T[s+m+1]) mod q
```

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

$S = 1$

$t_2 = (10(3 - T[1+1]\cdot 10) + T[1+2+1]) \bmod 11$

$= (10(3 - 1\cdot 10) + 1) \bmod 11$

$= (10\cdot(-7) + 1) \bmod 11$

$= (-70+1) \bmod 11$

$= -69 \bmod 11$

$= -69 - (-77) \bmod 11$

$= (-69+77) \bmod 11$

$8 \bmod 11$

$t_2 \Rightarrow 8$

AKTU PYQs

1. What is string matching algorithm? Explain Rabin-Karp method with examples.(AKTU 2022-23)
2. Write an algorithm for naïve string matcher. (AKTU 2020-21)
3. Write Rabin Karp string matching algorithm. Working modulo q = 11, how many spurious hits does the Rabin karp matcher in the text T = 3141592653589793, When looking for the pattern P = 26.

AKTU (2022-23)

4. Explain and Write the Naïve-String string matching algorithm.

   Suppose the given pattern p = aab and given text T = a c a a b c.

   Apply Naïve-string Matching algorithm on above Pattern (P) and Text (T) to find the number of occurrences of P in T.

AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-5 : Lecture-2

Today's Target

- Knuth Morris Pratt Algorithm for String Matching
- AKTU PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

Knuth Morris-Pratt (KMP) Algorithm

- Motivation: To study the pattern and find if there is any way to reduce the no. of comparisons. (Avoid too much shifts as we do in case of naive algorithm)

- The idea of KMP is that inside a given string, is there any prefix which is same as suffix? Find out…
- In other words -we will find out that if beginning part of the pattern is also repeating within the pattern?

P :-

a	b	c	d	a	b	c

Prefix :- a, ab, abc, abcd, abcda - - - -

Suffix :- c, bc, abc, dabc, - - - -

	1	2	3	4	5	6	7	8	9	10
P_1 :	a	b	c	d	a	b	e	a	b	f
LPS / π	0	0	0	0	1	2	0	1	2	0

	1	2	3	4	5	6	7	8	9	10	11
P_2 :	a	b	c	d	e	a	b	f	a	b	c
LPS / π	0	0	0	0	0	1	2	0	1	2	3

	1	2	3	4	5	6	7	8	9	10
P_3 :	a	a	b	c	a	d	a	a	b	e
LPS / π	0	1	0	0	1	0	1	2	3	0

Pseudo code

```
KMP (T, P)
n ← length (T)
m ← length (P)
π ← LPS (P)
q ← 0
for i ← 1 to n
        while q > 0 and P [q+1] ≠ T [i]
                q ← π[q]
        if P [q+1] = T[i]
                q ← q+1
        if q = m
                print "Pattern occurs with shift" i - m
                q ← π[q]
```

{ We never move backward in T, rather we only check P }

```
LPS (P) // Prefix function or π value
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]
        if P[k+1] = P[q]
                k <- k+1
        π[q] <- k
return π
```

Q- Find Prefix function (π) of the pattern

P: a b a c a b

	1	2	3	4	5	6
P:	a	b	a	c	a	b
π	0	0	1			

m = 6

K = 0, q will run from 2 to 6.

q = 2 |—

K ≯ 0 {loop condition is false}

P[0+1] ≠ P[2]

π[2] = 0

q = 3 |—

K ≯ 0 {loop condition is false}

P[0+1] = P[3]

K = 0+1 = 1

K = 1

π[3] = 1

```
LPS (P) // Prefix function or π value
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]
        if P[k+1] = P[q]
                k <- k+1
        π[q] <- k
return π
```

Q- Find Prefix function (π) of the pattern

P: a b a c a b

	1	2	3	4	5	6
P:	a	b	a	c	a	b
π	0	0	1	0		

m = 6

q = 4

K > 0 ✓ and P[1+1] ≠ P[4] ✓ {loop cond^n is true}

K = π[K] ⇒ K = π[1]

K = 0

K ≯ 0 {loop is false}

P[0+1] ≠ P[4]

π[4] = 0

```
LPS (P) // Prefix function or π value
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]
        if P[k+1] = P[q]
                k <- k+1
        π[q] <- k
return π
```

Q- Find Prefix function (π) of the pattern

P: a b a c a b

	1	2	3	4	5	6
P:	a	b	a	c	a	b
π	0	0	1	0	1	2

m = 6

q = 5 :- $K \neq 0$ { loop condition is false }

$P[0+1] = P[5]$

$K = K+1 \Rightarrow K = 0+1 \Rightarrow K = 1$

$\pi[5] = 1$

q = 6

$K > 0$ ✓ and $P[1+1] = P[6]$ { loop cond^n is false }

$P[1+1] = P[6]$.

so $K = K+1 \Rightarrow K = 1+1 \Rightarrow K = 2$

$\pi[6] = 2$

Q.2 Explain KMP matcher and also implement it by algorithm, where: P = a, a, b, a, b, b, a And T = b, a, b, a, a, b, a, b, b, a (AKTU 2023-24)

```
LPS (P) // Prefix function or π value
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]
        if P[k+1] = P[q]
                k <- k+1
        π[q] <- k
return π
```

P =

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

K=0 π

0	1	0				

q=2 :— K ≯ 0 {loop is false}

P[0+1] = P[2]

K = 0+1 ⇒ K=1

π[2] = 1

q=3 :— K > 0 ✓ P[1+1] ≠ P[3] ✓ {loop cond^n is True}

K = π[1] ⇒ K=0

K ≯ 0 {loop cond^n is false}

P[0+1] = P[3]

π[3] = 0

LPS (P) // Prefix function or π value

```
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]
        if P[k+1] = P[q]
                k <- k+1
        π[q] <- k
return π
```

	1	2	3	4	5	6	7
P =	a	a	b	a	b	b	a
K=0 π	0	1	0	1	0		

m = 7

q = 4 :— K ≠ 0 { loop condition is false }

P[0+1] = P[4]

K = 0+1 ⇒ K = 1

π[4] = 1

q = 5 :— K > 0 — P[1+1] ≠ P[5] — { loop cond^n is True }

K = π[K] ⇒ K = π[1] ⇒ K = 0

K ≠ 0 { loop condition is false }

P[0+1] ≠ P[5]

π[5] = 0

LPS (P) // Prefix function or π value

```
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]
        if P[k+1] = P[q]
                k <- k+1
        π[q] <- k
return π
```

	1	2	3	4	5	6	7
P =	a	a	b	a	b	b	a
K=0 π	0	1	0	1	0	0	1

✓

q=6 :- $K \neq 0$ { loop cond^n is false }

$P[0+1] \neq P[6]$

$\pi[6] = 0$

q=7 $K \neq 0$ { loop condition is false }

$P[0+1] = P[7]$

$K = K+1 \Rightarrow K = 0+1 \Rightarrow K = 1$

$\pi[7] = 1$

```
KMP (T, P)
n ← length (T)
m ← length (P)
π ← LPS (P)
q ← 0
for i ← 1 to n
        while q > 0 and P [q+1] ≠ T [i]
                q ← π[q]
        if P [q+1] = T[i]
                q ← q+1
        if q = m
                print "Pattern occurs with shift" i - m
                q ← π[q]
```

{ We never move backward in T, rather we only check P }

i=1

T :

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

i

s=0 s=1 s=2 s=3

q=0

q

P :

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

π

0	1	0	1	0	0	1

q=m

q=7

s=i-m

10-7

s=3

```
KMP (T, P)  // KMP Matcher
n ← length (T)
m ← length (P)
π ← LPS (P)
q ← 0
for i ← 1 to n
        while q > 0 and P [q+1] ≠ T [i]
                q ← π[q]
        if P [q+1] = T[i]
                q ← q+1
        if q = m
                print "Pattern occurs with shift" i - m
                q ← π[q]
```

{ We never move backward in T, rather we only check P }

i	1	2	3	4	5	6	7	8	9	10
T =	b	a	b	a	a	b	a	b	b	a

$n = 10$

q	1	2	3	4	5	6	7
P =	a	a	b	a	b	b	a
	0	1	0	1	0	0	1

$m = 7$

$q = 0$

$i = 1$:- $q \not> 0$ { loop condition is false }

$P[0+1] \neq T[1]$

$q \neq m$

$i = 2$:- $q \not> 0$ { loop condition is false }

$P[0+1] = T[2]$

$q = 0+1 \Rightarrow q = 1$

$q \neq m$

```
KMP (T, P)
n ← length (T)
m ← length (P)
π ← LPS (P)
q ← 0
for i ← 1 to n
        while q > 0 and P [q+1] ≠ T [i]
                q ← π[q]
        if P [q+1] = T[i]
                q ← q+1
        if q = m
                print "Pattern occurs with shift" i - m
                q ← π[q]
```

{ We never move backward in T, rather we only check P }

	1	2	3	4	5	6	7	8	9	10
T =	b	a	b	a	a	b	a	b	b	a

	1	2	3	4	5	6	7
P =	a	a	b	a	b	b	a
	0	1	0	1	0	0	1

i=3 :- q>0 ✓ P[1+1] ≠ T[3] ✓ { loop condition is true }

q = π[1] ⇒ q=0

q ≯ 0 { loop condition is false }

P[0+1] ≠ T[3]

q ≠ m

i=4 :- q ≯ 0 { loop condition is false }

P[0+1] = T[4]

q = 0+1 ⇒ q=1

q ≠ m

```
KMP (T, P)
n ← length (T)
m ← length (P)
π ← LPS (P)
q ← 0
for i ← 1 to n
        while q > 0 and P [q+1] ≠ T [i]
                q ← π[q]
        if P [q+1] = T[i]
                q ← q+1
        if q = m
                print "Pattern occurs with shift" i - m
                q ← π[q]
```

// m

// n

{We never move backward in T, rather we only check P}

	1	2	3	4	5	6	7	8	9	10
T =	b	a	b	a	a	b	a	b	b	a

	1	2	3	4	5	6	7
P =	a	a	b	a	b	b	a
	0	1	0	1	0	0	1

m = 7

i = 5 :-

q > 0 ✓ and P[1+1] = T[5] ✗ {loop cond" is false}

P[1+1] = T[5] ✓

q = q+1 ⇒ q = 1+1 ⇒ q = 2

q ≠ m

i = 6 :-

q > 0 ✓ P[2+1] = T[6] ✗ {loop cond" is false}

P[2+1] = T[6] ✓

q = 2+1 ⇒ q = 3

q ≠ m

```
KMP (T, P)
n ← length (T)
m ← length (P)
π ← LPS (P)
q ← 0
for i ← 1 to n
        while q > 0 and P [q+1] ≠ T [i]
                q ← π[q]
        if P [q+1] = T[i]
                q ← q+1
        if q = m
                print "Pattern occurs with shift" i - m
                q ← π[q]
```

{ We never move backward in T, rather we only check P }

	1	2	3	4	5	6	7	8	9	10
T =	b	a	b	a	a	b	a	b	b	a

	1	2	3	4	5	6	7
P =	a	a	b	a	b	b	a
	0	1	0	1	0	0	1

$i = 7$:- $q > 0$ ✓ $P[3+1] = T[7]$ ✗ { loop condition is false }

$P[3+1] = T[7]$ ✓

$q = 3+1 = 4$

$q \neq m$

$i = 8$:- $q > 0$ ✓ and $P[4+1] = T[8]$ ✗ { loop cond^n is false }

$P[4+1] = T[8]$ ✓

$q = 4+1 \Rightarrow 5 \Rightarrow q = 5$

$q \neq m$

```
KMP (T, P)
n ← length (T)
m ← length (P)
π ← LPS (P)
q ← 0
for i ← 1 to n
        while q > 0 and P [q+1] ≠ T [i]
                q ← π[q]
        if P [q+1] = T[i]
                q ← q+1
        if q = m
                print "Pattern occurs with shift" i - m
                q ← π[q]
```

{ We never move backward in T, rather we only check P }

	1	2	3	4	5	6	7	8	9	10
T =	b	a	b	a	a	b	a	b	b	a
	s=0	s=1	s=2	s=3						

	1	2	3	4	5	6	7
P =	a	a	b	a	b	b	a
	0	1	0	1	0	0	1

$m = 7$

$i = 9$:

$q > 0$ ✓ and $P[5+1] = T[9]$ ✗ { loop cond^n is false }

$P[5+1] = T[9]$ ✓

$q = 5+1 \Rightarrow q = 6$

$q \neq m$

$i = 10$:

$q > 0$ ✓ and $P[6+1] = T[10]$ ✗ { loop cond^n is false }

$P[6+1] = T[10]$ ✓

$q = 6+1 \Rightarrow q = 7$

$q = m$ ✓

" pattern occurs with shift " $(10-7)$ → 3.

$q = \pi[7] \Rightarrow 1$

Time Complexity

```
KMP (T, P)
n ← length (T)
m ← length (P)
π ← LPS (P)
q ← 0
for i ← 1 to n
        while q > 0 and P [q+1] ≠ T [i]
                q ← π[q]
        if P [q+1] = T[i]
                q ← q+1
        if q = m
                print "Pattern occurs with shift" i - m
                q ← π[q]
```

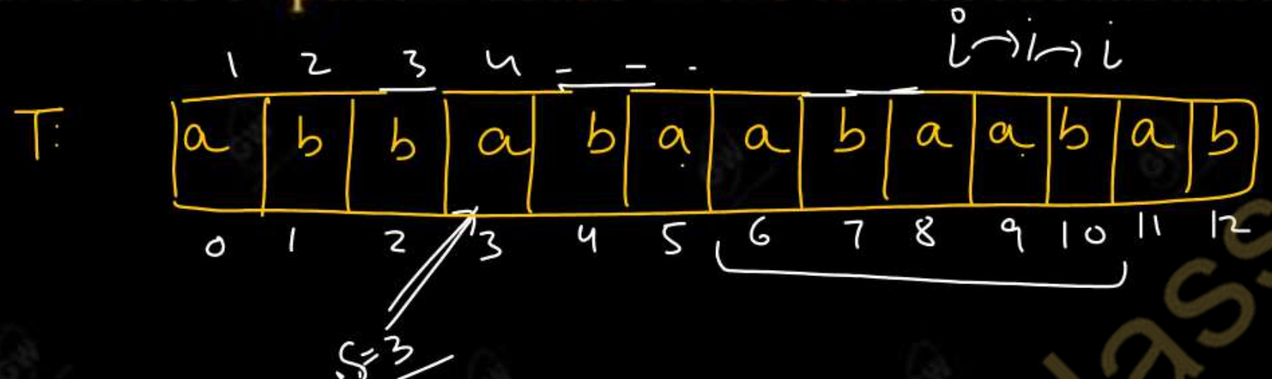
// $O(m)$ ⇒ Time taken to generate the π table.

$O(n)$

$T(n) = O(m+n)$

{ We never move backward in T, rather we only check P }

Q.3 Write KMP algorithm for staring matching. Perform it to search the occurrences of pattern abaab in the text abbabaabaabab. (AKTU 2020-21)



q=0

q → q → q → q

P:

	a	b	a	a	b
π	0	0	1	1	2

pattern occurs at $s = i - m$

$8 - 5$

s=3

pattern occurs at $s = i - m$

$= 11 - 5$

s ⇒ 6

AKTU PYQs

1. Explain KMP matcher and also implement it by an algorithm, where P= a, a, b, a, b, b, a and T = b, a, b, a, a, b, a, b, b, a (AKTU 2023-24)
2. Describe in detail Knuth Morris Pratt string matching algorithm. Compute the prefix function for the pattern ababbabbabbababbabb when input alphabet is {a,b}.(AKTU 2019-20)
3. Write KMP algorithm for staring matching. Perform it to search the occurrences of pattern abaab in the text abbabaabaabab.(AKTU 2020-21)
4. Compute prefix function π for the pattern P= a b a c a b using KNUTH MORRIS – PRATT.
5. Give a linear time algorithm to determine if a text T is a cycle rotation of another string T. For example, 'RAJA' and 'JARA' are cyclic rotations of each other.
6. Explain and write the Knuth-Morris-Pratt algorithms for pattern matching also write its time complexity. (AKTU 2021-22)

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-5 : Lecture-3

Today's Target

- Theory of NP completeness
- AKTU PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

Types of Computational Problems

Polynomial	Exponential
A problem is said to be polynomial time if there exists an algorithm that solves the problem in time. $T(n) = O(n^c)$ where c is a constant. Example. 1) Sorting: $O(n \log n) = O(n^2)$ 2) Searching: $O(n)$ 3) All pair shortest path: $O(n^3)$ 4) Minimum cost-panning Tree : $O(E \log E) = O(E^2)$	A problem is said to be exponential time problem if no polynomial time algorithm can be developed to solve it. $T(n) = O(2^n)$ or $T(n) = O(k^n)$ where K = constant Example : 1) 0/1 knapsack problem 2) Traveling salesman problem 3) Hamiltonian cycle problem 4) Graph coloring 5) sum of subsets

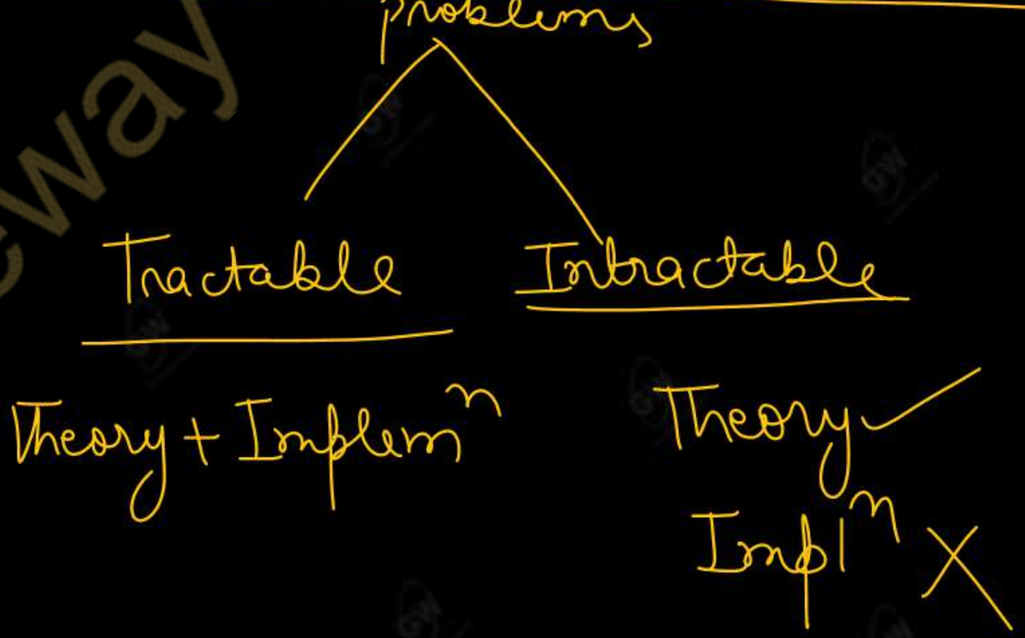
Q. : - What are P, NP, NP Hard and NP-complete problems? (AKTU)

P-class-

- The solution to **P problems** is easy to find.
- **P** is often a class of computational problems that are solvable and tractable. Tractable means that the problems can be solved in theory as well as in practice. But the problems that can be solved in theory but not in practice are known as intractable.
- P class problems can be solved in polynomial time. using deterministic algorithm. Ex- deterministic Turing Machine.

Example:

- Fractional knapsack
- Minimum Spanning Tree
- Merge Sort

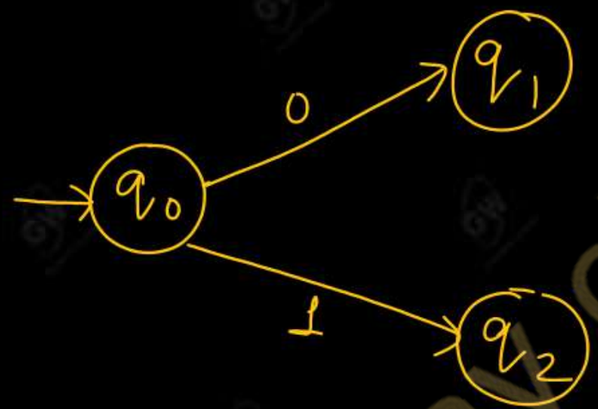


Deterministic Algorithm

(1)→ Given a particular input, the computer will give same output always.

$\Sigma = \{0, 1\}$

(2)



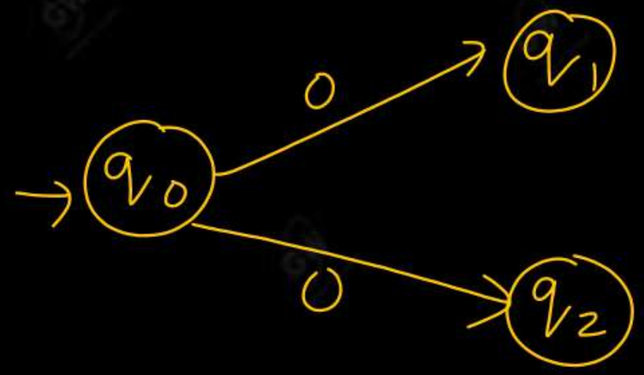
(3)→ Can solve problems in polynomial time.

(4)→ Can determine what will be the next step.

Non-Deterministic Algorithm

(1)→ Given a particular input, the computer do not understand what is the actual of output. It may give different outputs for same inputs.

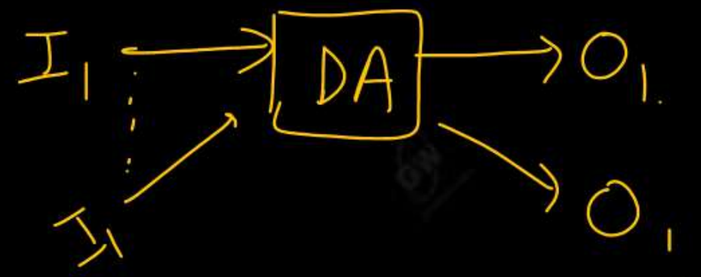
(2)



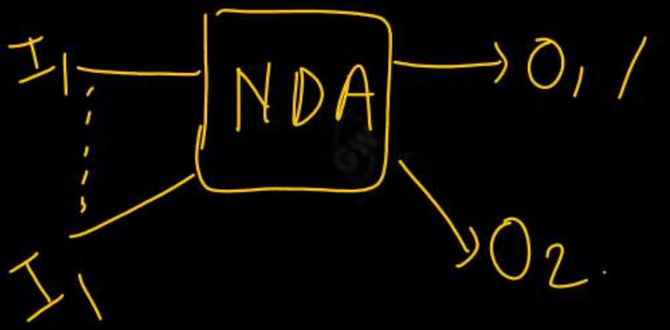
(3)→ Can not solve the problem in polynomial time

(4)→ Can not determine, the next step.

Deterministic



Non-deterministic

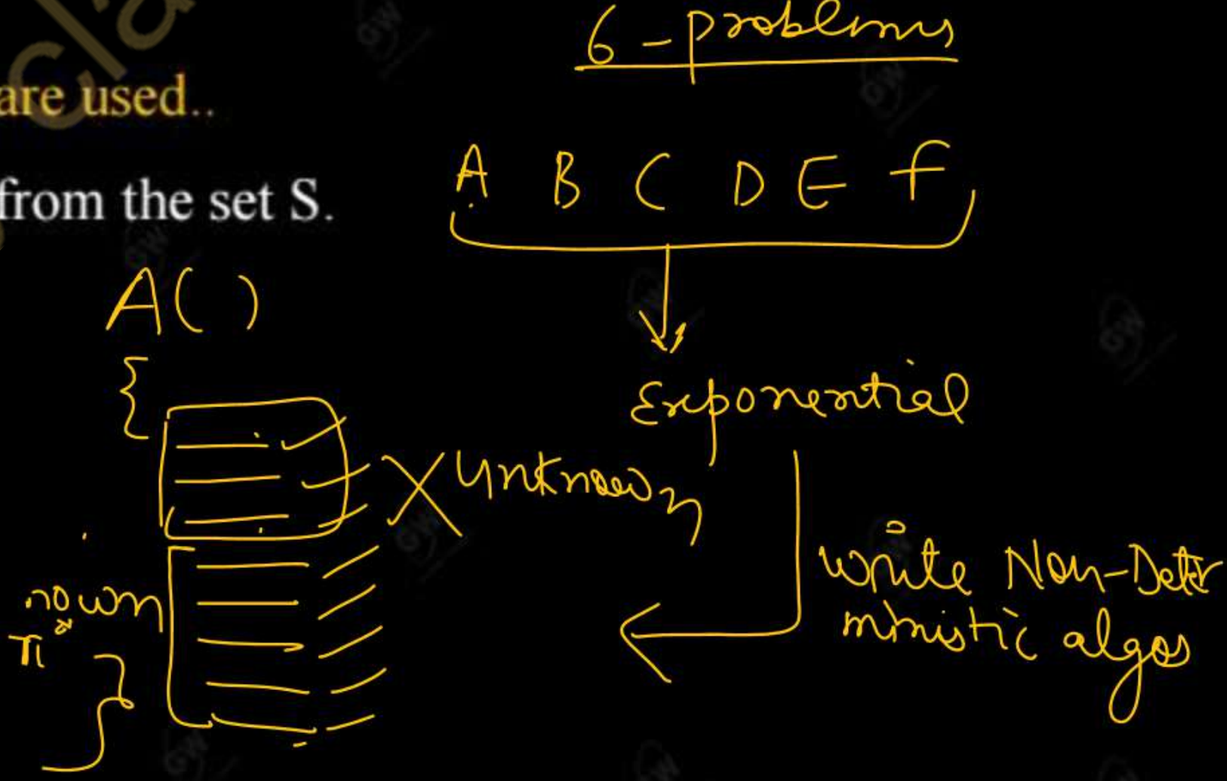


How to solve exponential time problem:

- Try to find similarity b/w the different EXPONENTIAL time problems so that if we find in future the solution of one problem in polynomial time, then we may solve the other problem also in similar ways. ⇒ Try to write non-deterministic algorithms for them.

- To write such algorithms three new functions are used..

1. Choice (s): It arbitrarily choose one element from the set S.
2. Failure: It signals an unsuccessful solution.
3. success : It signals a successful solution.



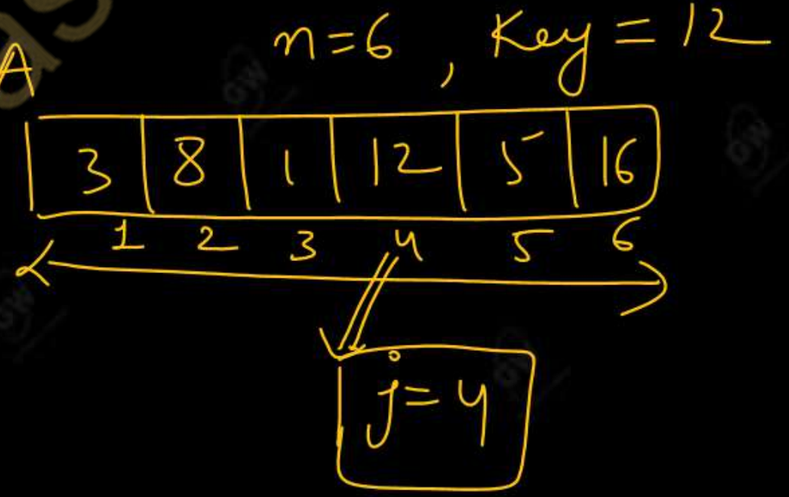
Example problem – Searching x in A[1: n], n ≥ 1

On success if A [j] = x it returns j.

Non-Deterministic Algo

or return 0 otherwise.

```
Algorithm N Search (A, n, key)
{
    j= choice( );        ⟹ unknown.
    if (key = = A [j] )
    {
        Write (j);
        Success ( );     ⟹ unknown
    }
    Write (0);
    failure ( );         ⟹ unknown
}
```



First time choice ( ) is called and it is giving us the index of key element and we check that if that key element is present at position j then success otherwise failure.Overall algo takes constant time O(1)

So, in this way non – deterministic polynomial time algorithm for these exponential time problems are written.

NP-class- Algorithms to solve these are Non- deterministic but takes polynomial time to run.

In other words -

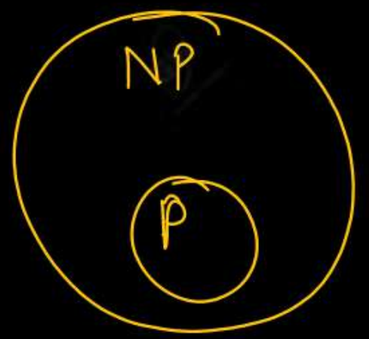
YES/NO

- NP is a set of decision problems that can be solved by Non-deterministic turning machine in polynomial time. P is a subset of NP, so any problem that can be solved by deterministic turning machine in polynomial time, can be solved by non-deterministic TM in polynomial time.
- NP can also be thought of as class of problems "Whose solutions can be verified in polynomial time"

1. Boolean Satisfiability Problem (SAT)
2. Hamiltonian Path Problem
3. Graph coloring

Relation b/w P-class and NP-class

$P \subseteq NP$



To relate unsolved problems, we must take some base problem into consideration.

Example. CNF – Satisfiability Problem

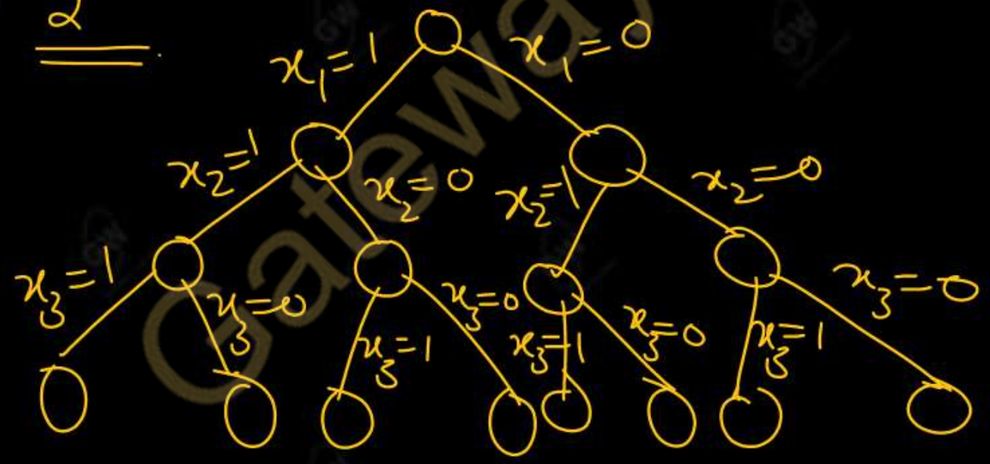
$x_i \begin{cases} 0 \\ 1 \end{cases}$

$x_i = \{x_1, x_2, x_3\}$

$\text{CNF} = \underbrace{(x_1 \vee \bar{x}_2 \vee x_3)}_{C_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee \bar{x}_3)}_{C_2}$

For three variables :- 2^3 combinations are possible.

For 'n' variables :- 2^n



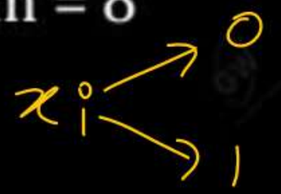
x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

0/1 knapsack problem

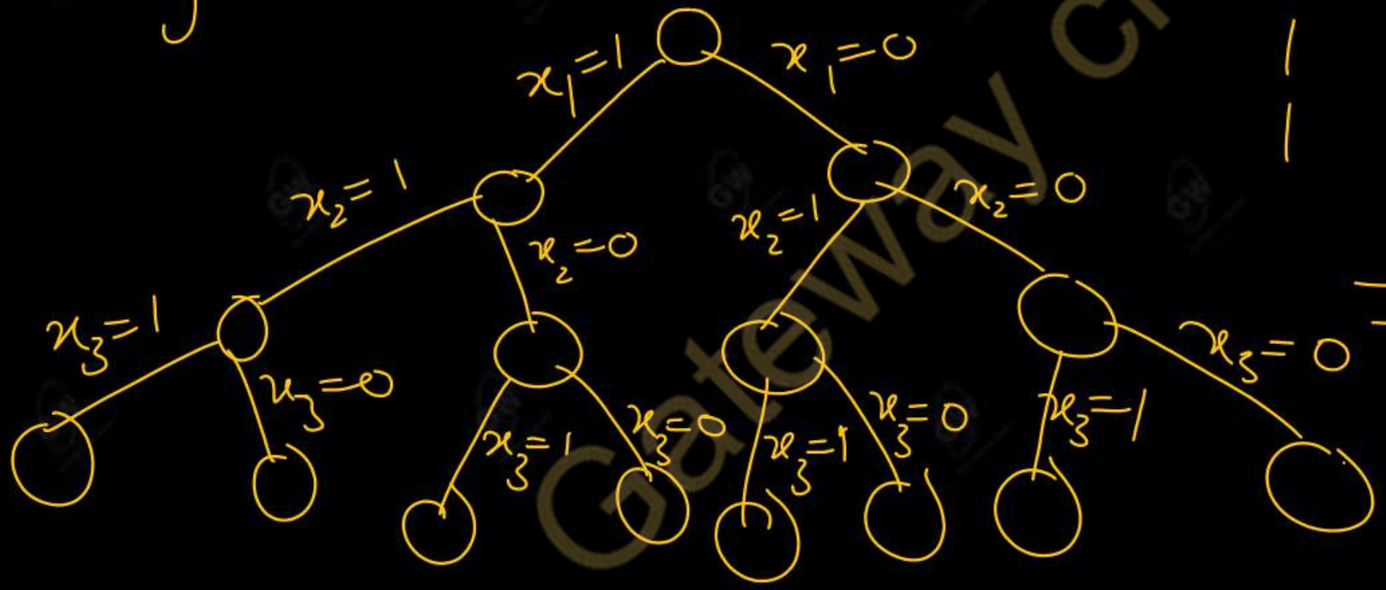
P = { 10, 8, 12} n = 3

W = { 5, 4, 3} m = 8

x_1 = { _ , _ , _ }



for n objects ⇒ 2^n



x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

2^8 combinations

⟹ Similar to CNF-satisfiability problem.

satisfiability $\propto$ 0/1 Knaps

Reduction

Suppose we have two decision problems P_1 and P_2

I_1 is the input for P_1

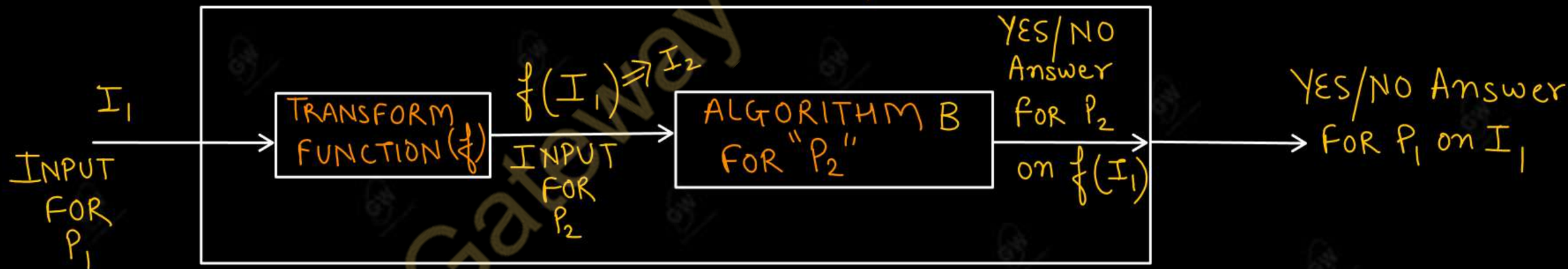
And we have to write algorithm A to solve this.

I_2 is the input for P_2

And Algorithm B solves the problem in polynomial time

P_1 is reducible to P_2

$P_1 \propto P_2$



The idea is to find a transformation from P1 to **P2** so that algorithm **B** can be part of algorithm **A** to solve **P1**.

For example, if we have library functions to solve certain problems and if we can reduce a new problem to one of the solved problems, we save a lot of time.

P_1 is reducible to problem P_2 if there is a function which convert input of problem P_1 into input instance of problem P_2 and solution of that instance provides the solution to problem.

$$(p_1 \propto p_2)$$

NP-hard class

An NP-hard problem is at least as hard as the hardest problem in NP

It is a class of problems such that every problem in NP reduces to NP-hard.

Features:

- All NP-hard problems are not in NP.
- If a solution for an NP-hard problem is given then it takes a long time to check whether it is right or not.
- A problem A is NP-hard if, for every problem L in NP, there exists a polynomial-time reduction from L to A.

Examples of NP-hard problems:

1. **Halting problem.**
2. **Qualified Boolean formulas.**
3. **No Hamiltonian cycle.**

NP Complete class

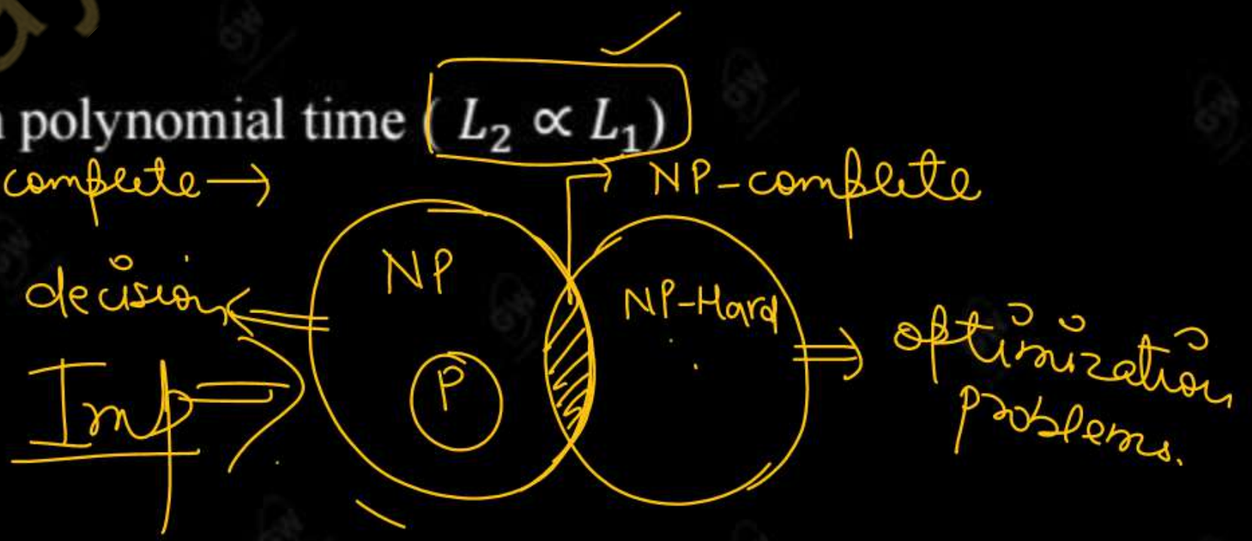
- A problem is NP-complete if it is both NP and NP-hard.
- A problem L in NP is NP-complete if all other problems in NP can be reduced to L in polynomial time.
- If any NP-complete problem can be solved in polynomial time, then every problem in NP can be solved in polynomial time. NP-complete problems are the hardest problems in the NP set.

A decision problem L_1 is NP-complete if it follows the below two properties –

1) L_1 is in NP

2) Every problem L_2 in NP is reducible to L_1 in polynomial time ($L_2 \propto L_1$)

Relation b/w P, NP, NP-Hard and NP-complete →



AKTU PYQs

1. Explain P, NP, NP Hard and NP complete classes with example.(AKTU 2022-23, 2023-24)
2. Write short note on NP Complete and NP Hard problems. (AKTU 2020-21, 2021-22)

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-5 : Lecture- 4

Today's Target

- Approximation algorithms
  - Vertex Cover Problem
  - Set Cover Problem
  - Traveling Salesman Problem
- AKTU PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

Approximation Algorithms // heuristic Algorithm

These algorithms are ways to deal with NP-completeness for an optimization problem.

- Does not guarantee the best solution
- The goal is to come as close as possible to the optimal solution in polynomial time.
- Also called as heuristic algorithms.

Features of Approximation Algorithm

•An approximation algorithm guarantees to run in polynomial time though it does not guarantee the most effective solution.

•Approximation algorithms are used to get an answer near the (optimal) solution of an optimization problem in polynomial time

Performance Ratios (Approximation ratio)

- Idea is to find how close the approximate solution is to the optimal solution.
- Represented by P(n), where n is the input size of algorithm.
- Algorithm has approximate ratio of p(n) iff –

$$\max\left\{\frac{c}{c^*}, \frac{c^*}{c}\right\} \leq p(n)$$

c → It is the cost of near optimal solution given by the approx^n algo.

c^* → optimal cost

The algorithm is then called a p(n) approximation algorithm. It can be applied on minimization or maximization problems.

For Minimization Problems: P(n) is calculated by $\frac{c}{c^*}$

since $0 \leq c^* \leq c$

$$\frac{c}{c^*} \leq p(n)$$

For Maximization Problems: P(n) is calculated by $\frac{c^*}{c}$

since $0 \leq c \leq c^*$

$$\frac{c^*}{c} \leq p(n)$$

We assume that all c, c^* are positive.

So, p(n) is well defined and $p(n) \geq 1$.

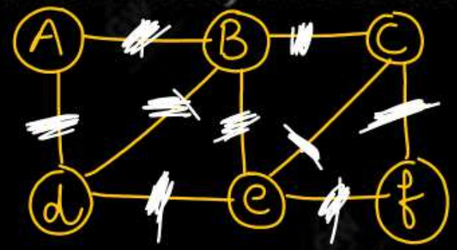
If p(n) = 1 means approximate algorithm generates the exact optimal solution.

Larger p(n) means worst algorithm.

Vertex cover Problem

- A vertex cover of an undirected graph is a subset of its vertices such that for every edge (u, v) of the graph, either 'u' or 'v' is in the vertex cover.
- The set covers all edges of the given graph.
- The vertex cover problem is to find minimum size vertex cover.
- In the vertex cover problem, the optimization problem is to find the vertex cover with the **fewest vertices**, and the approximation problem is to find the vertex cover with **few vertices**.

Example:1



$VC_1 = \{A, B, e, C\}$

$VC_2 = \{B, e, d, f\}$

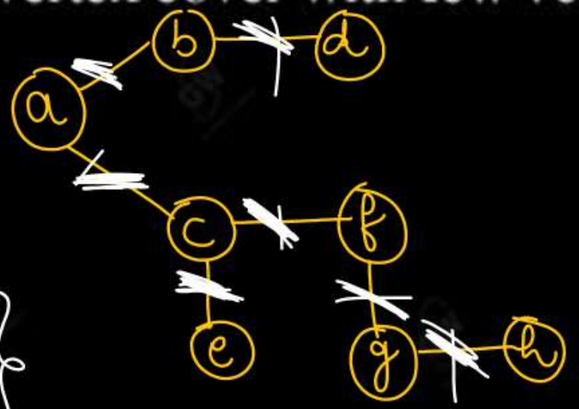
near optimal

Example 2-

$VC_1 = \{a, c, g, b\}$

optimal →

$VC_2 = \{c, g, b\}$



Pseudocode

Approx – vertex – cover

$C \leftarrow \phi$ → vertex cover initially Empty

$E' \leftarrow E$ → Set of all edges of the graph

While $E' \neq \phi$ do

Let (u, v) be an arbitrary edge of E'

$C \leftarrow C \cup \{(u, v)\}$

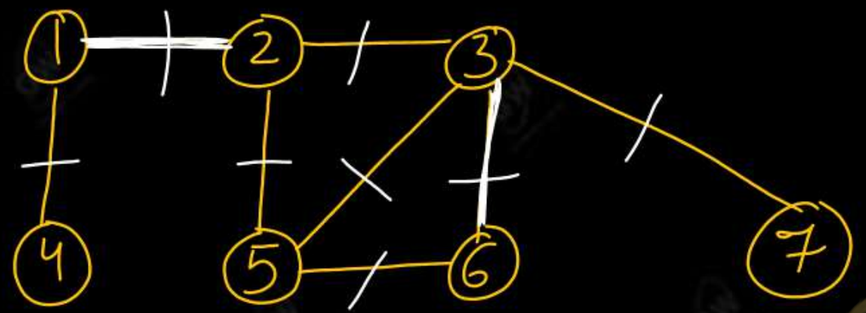
Remove from E' all edges incident on either u or v.

End while

return C.

Example

Apply vertex-cover approximation algo on the given graph.



$C = \{\phi\}$

$E' = \{(1,2)\ (1,4)\ (2,5)\ (2,3)\ (3,6)\ (3,5)\ (3,7)\ (5,6)\}$

(1,2) ⇓ selected

$C = \{\phi, 1, 2\}$

$E' = \{(3,6)\ (3,5)\ (3,7)\ (5,6)\}$

(3,6) ⇓ selected

$E' = \{\phi\}$, $\boxed{C = \{\phi, 1, 2, 3, 6\}}$

C (near optimal solution)

$\boxed{C = 4}$

C^* (optimal solution)

$\boxed{C^* = 3}$

optimal Vertex cover $= \{3, 5, 1\}$

As this is a minimization problem

$$\frac{C}{C^*} \leq P(n) \Rightarrow P(n) \geq \frac{4}{3}$$

$$\Rightarrow \boxed{P(n) \geq 1}$$

Set Cover Problem

A set cover of a set T is any collection of subsets of T whose union is T.

Problem: Given a weight for each subset, find the set cover which minimizes the total weight.

Ex:

Set T = {1, 2, 3, 4, 5, 6, 7, 8}

Available Subsets:

(1) $S_1 = \{1, 2, 3\}$ $w_1 = 1$

(2) $S_2 = \{2, 7, 8\}$ $w_2 = 2$

(3) $S_3 = \{4, 5, 6, 7\}$ $w_3 = 3$

(4) $S_4 = \{4, 5, 6, 8\}$ $w_4 = 4$

Problem

- An Instance (X,F) of the set covering problem consists of a finite set X and a family F of subsets of X, such that every element of X belongs to at least 1 one subset in F.
- The problem is to find a minimum-size subset cover $C \subseteq F$ whose members cover all of X.

```
Greedy-Set-Cover (X, F)
U = X
C = ∅
while U ! = ∅ do
    select an S ∈ F that maximizes |S ∩ U|
    U = U - S
    C = C ∪ {S}
return C
```

F → family of subsets.

X → Given set

U = X → set of all the elements in X

C = ∅ → Set cover is initially empty.

Example → set $X = \{1, 2, 3, 4, 5, 6, 7, 8\}$

F ⇐

$S_1 = \{1, 2, 3\}$ $w_1 = 1$

$S_2 = \{2, 7, 8\}$ $w_2 = 2$

$S_3 = \{4, 5, 6, 7\}$ $w_3 = 3$

$S_4 = \{4, 5, 6, 8\}$ $w_4 = 4$

$C = \phi$, $U = \{1, 2, 3, \cancel{4}, \cancel{5}, \cancel{6}, 7, \cancel{8}\}$

Suppose S_4 is chosen as it fulfils the condition of Max $|S \cap U|$

⇓

$U = \{1, 2, 3, \cancel{7}\}$, $C = \{\phi, S_4\}$

Now S_3 is chosen as it satisfy Max $|S \cap U|$

$U = \{1, 2, 3\}$

$C = \{\phi, S_4, S_3\}$

Now S_1 is chosen

$U = \{\phi\}$

near optimal set cover

$C = \{\phi, S_4, S_3, S_1\}$

c (near optimal cost) $= 1+3+4 \Rightarrow 8$

c^* (optimal cost) $= 1+2+3 \Rightarrow 6$

For min^n problem

$\frac{c}{c^*} \leq P(n) \Rightarrow \frac{8}{6} \leq P(n)$

$P(n) \geq 1$

optimal Set cover

$C = \{S_1, S_2, S_3\}$

Applications of Set Cover Problem

(1) Assign fire stations in a city

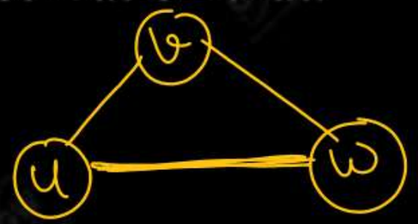
(2) Allocate sale branch offices for a company

(3) To create smallest team of members with vehicles for every required skill, there is a member in the team having that skill.

Traveling Salesman Problem (Approximate solution)

Since finding the shortest tour for TSP requires so much computation, we may consider to find a tour that is almost as short as the shortest. That is, it may be possible to find near optimal solution.

Example: We can use the approximation algorithm for Hamiltonian cycle problem. It is relatively easy to find a tour that is longer by at most a factor of two than the optimal tour. The method is based on the observation that:



$$C(u, w) \leq C(u, v) + C(v, w)$$

Approx -TSP-TOUR

//Computes a near-optimal tour of an undirected graph G.

Procedure APPROX-TSP-TOUR (G, c):

Begin:

Select a vertex r ∈ V(G) to be the "root" vertex.

Grow a minimum spanning tree T for G from root r, using Prim's algorithm.

Apply a preorder walk of T and let L be the list of the vertices visited in the walk.

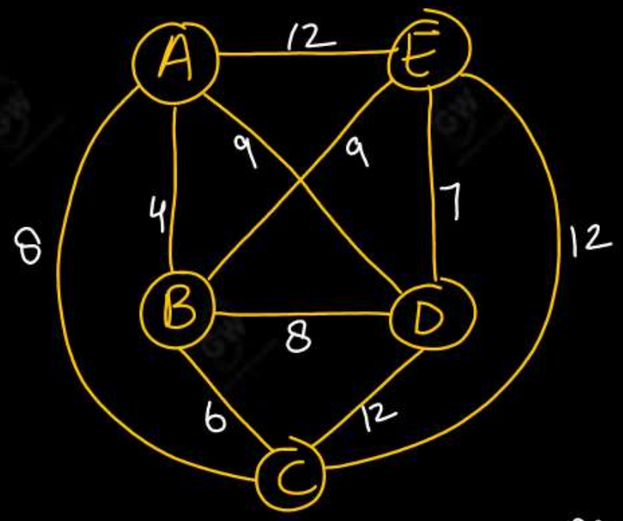
Form the Hamiltonian cycle H that visits the vertices in order L.

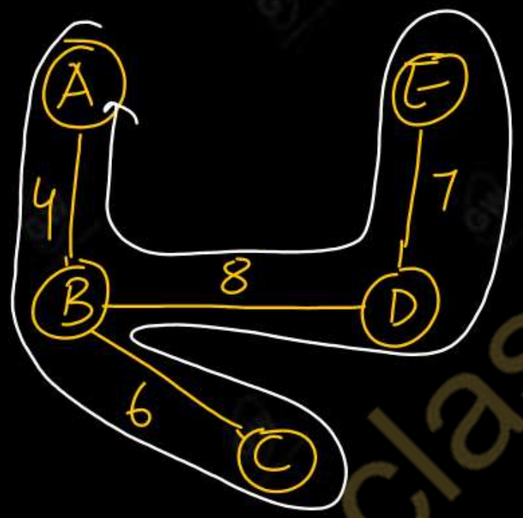
/* H is the result to return. */

End.

Note — A pre-order tree walk recursively visits every vertex in the tree, listing a vertex when it is first encountered, before any of it's children are visited

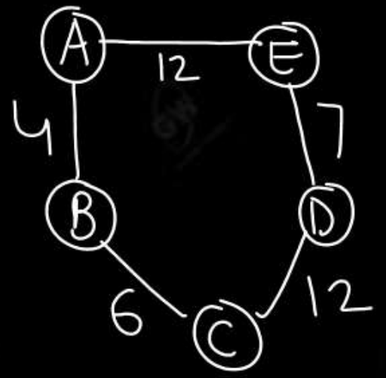
Problem : Solve following instance of Traveling Salesman Problem using Twice-around-tree Algorithm





pre-order walk — $A \to B \to C \to \overset{X}{B} \to D \to E \to \overset{X}{D} \to \overset{X}{B} \to A$

$A \to B \to C \to D \to E \to A$



cost = 41

near optimal soln

C = 41

AKTU PYQs

1. Write short notes on Approximation algorithm (AKTU 2020-21,2021-22)
2. Explain Approximation algorithms.(AKTU 2020-21)
3. What are approximation algorithms ? What is meant by P (n) approximation algorithms? Discuss approximation algo for traveling salesman problem(AKTU 2019-20, 2015-16)
4. Write and explain the algorithm to solve vertex cover problem using approximation algorithm.(AKTU 2021-22)
5. Explain approximation algorithm. Explore set cover problem using approximation algorithm. (AKTU 2020-21 2022-23)

AKTU

B.Tech 5th Sem

GW GATEWAY CLASSES

CS IT & CS Allied

DAA : Design & Analysis Of Algorithm

Unit-5 : Lecture-5

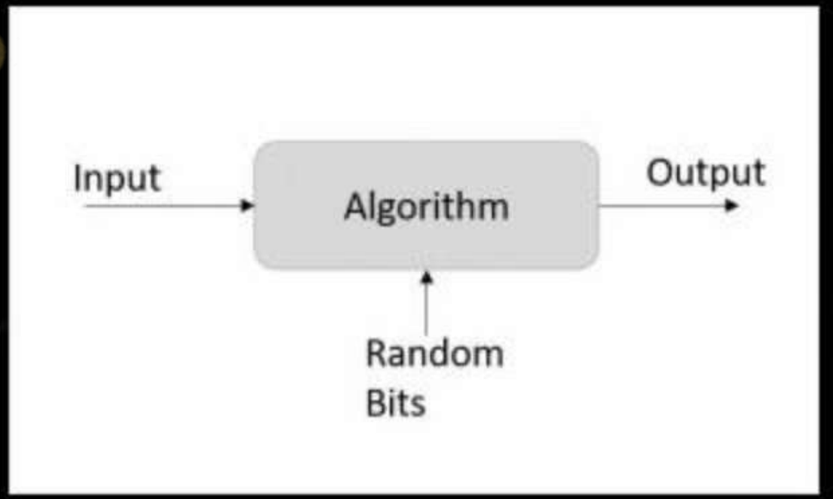
Today's Target

- Randomized Algorithms
- Algebraic Computation
- Fast Fourier Transform
- AKTU PYQs

By Dr. Nidhi Parashar Ma'am
- M.Tech Gold Medalist
- Net Qualified

Randomized algorithm

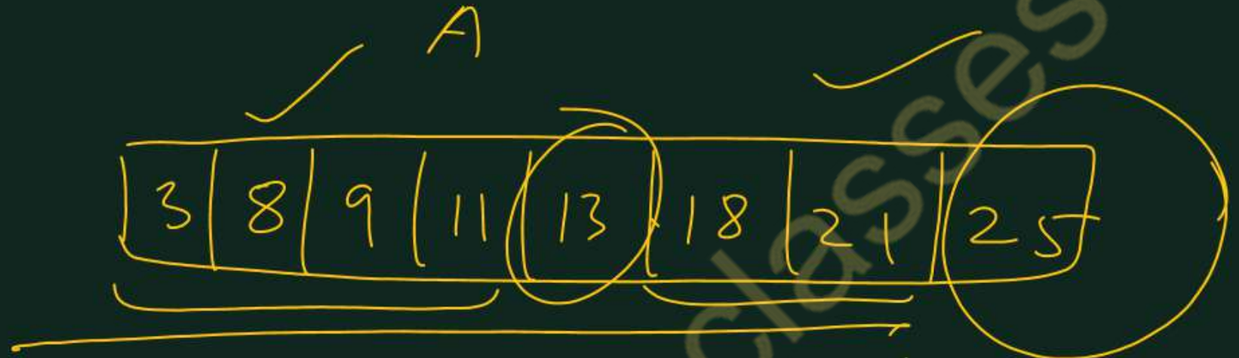
- It is a design approach where few random bits are added to a part of their logic.
- Randomized algorithms are non-deterministic so, they produce a different output every time they're executed.
- It is not the input that is randomized, but the logic of the standard algorithm. It makes some random choices in some steps of the algorithms.
- Output and/or running time of the algorithm may depend on the random choices made, so, If you run the algorithm more than once on the same input data, the results may differ depending on the random choice.



Need for Randomized Algorithms

- This approach is usually adopted to reduce the time complexity and space complexity.
- In randomized quick sort algorithm, randomizing the pivot selection would execute the algorithm faster than the worst time complexity. However, even if the algorithm chose the first element as pivot randomly and obtains the worst time complexity, executing it another time with the same input will solve the problem since it chooses another pivot this time.
- On the other hand, for algorithms like merge sort the time complexity does not depend on the input; even if the algorithm is randomized the time complexity will always remain the same. Hence, **randomization is only applied on algorithms whose complexity depends on the input**.

Quicksort



$T(n) = O(n^2)$

$O(n \log n)$

Types of randomized algorithms

(1) Las Vegas Algorithm (2)Monte Carlo algorithm

Las Vegas Algorithm

1. These algorithms never gives incorrect results.
2. Makes the time constraint as random variable.
3. The solution to a problem can be found in finite time but No. of possible solutions are limited.
4. Time complexity is evaluated as expected value. It may vary depending upon the random value

   "Time complexity is probabilistic but correctness is deterministic"
5. In string matching algorithms, once Las Vegas algorithm encounters an error, they start from the beginning. Thereby increases the probability of correctness.

Example:- Randomized Quicksort.

In randomized quicksort we choose the pivot element randomly. But the result is always a sorted array.

Las Vegas Algorithm to identity repeated number in Array

```
// finds repeated elements from A[1---n]
Repeat Element (A, n)
{
        While (true) do
        {
            i = Random ( ) mod (n+1)
            j = Random ( ) mod (n+1)
            //i and j are random numbers in range [1---n]
            If ((i≠j) and (A[i] = A(j) )) then
                return i
        }
}
```

n → size of array

10

Monte- Carlo Algorithms

1. This method focuses on finishing the execution within the given time constraint.
2. Therefore, the running time of this method is deterministic. It always has the same time complexity .
3. For example, in string matching, if monte Carlo algorithm encounters an error, it restarts the algorithm from the same point. Thus, saving time.
4. Sometimes it may produce incorrect outputs depending upon the random choices made.
5. **"Time Complexity is deterministic but correctness is probabilistic"**
6. Used in finding randomized median.

Monte Carlo Algorithm to search an element in array

```
Algorithm Monte Carlo Search (A, a, x)
{
    for (i = 0; i <= x; i++)
    {
        j = random ( ) mod (n+1);
        If (A[j] == a)
            return (True);
    }
}
```

x → no. of iterations

a → element to be searched

x = 3 , a = 4

0	1	2	3	4	5
8	1	3	4	6	9

i = 0	j = 1	X
i = 1	j = 5	X
i = 2	j = 1	X

i = 3	j = 4	X

It may provide correct result in x iterations or may not provide.

Algebraic Computation

- Computer Algebra or Symbolic Computation **refers to the study and development of algorithms and software for manipulating mathematical expressions** and other mathematical objects.
- Although computer algebra could be considered a subfield of scientific computing, they are generally considered as distinct fields because scientific computing is usually based on numerical computation with approximate floating-point numbers.
- While, computer algebra emphasizes exact computation with expressions containing variables that have no given value and are manipulated as symbols. It is widely used to experiment in mathematics and to design the formulas that are used in numerical programs.

It is also used for complete scientific computations, when purely numerical methods fail as in public key cryptography or for some linear problems..

- S/w apps that perform symbolic calculation are called computer algebra systems (includes at least a method to represent mathematical data in a computer, a user programming language usually different from PL used for implement) a dedicated memory manager, user Interface for I/P and O/P of mathematical ~~expense~~. expressions
- A large set of routines to perform usual operations like-
  1. Simplification of expressions
  2. Differentiation using chain-rule
  3. Polynomial factorization
  4. Indefinite Integration etc.

Things to be focused on algebraic computation

(1)→ Matrix operations (Strassen's Algo)

(2)→ Computational Geometry (Convex hull Problem)

3) Euclid Algorithm to compute gcd:

```
gcd(a, b)
{  if (b == 0)
      return a;
   else
      gcd(b, a%b)
}
```

Suppose a = 25, b = 15

gcd(25, 15)

→ gcd(15, 25%15) — 25%15 = 10

→ gcd(10, 15%10) — 15%10 = 5

→ gcd(5, 10%5) — 10%5 = 0

Any gcd = 5

Fast Fourier Transform

- A Fourier transform converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies. This operation is is often too slow to be practical. $T(n) = O(n^2)$
- Fast Fourier Transform (FFT) is an algorithm that computes a Discrete Fourier Transform (DFT) of an n-length vector in O(n log n) time.
- In the FFT algorithm, we apply the divide and conquer approach to polynomial evaluation by observing that if n is even, we can divide a degree (n-1) polynomial.

$$A(x) = a_0 + a_1x + a_2x^2 + \cdots \ldots + a_{n-1}x^{n-1}$$

into two polynomials of degree $\left(\frac{n}{2} - 1\right)$.

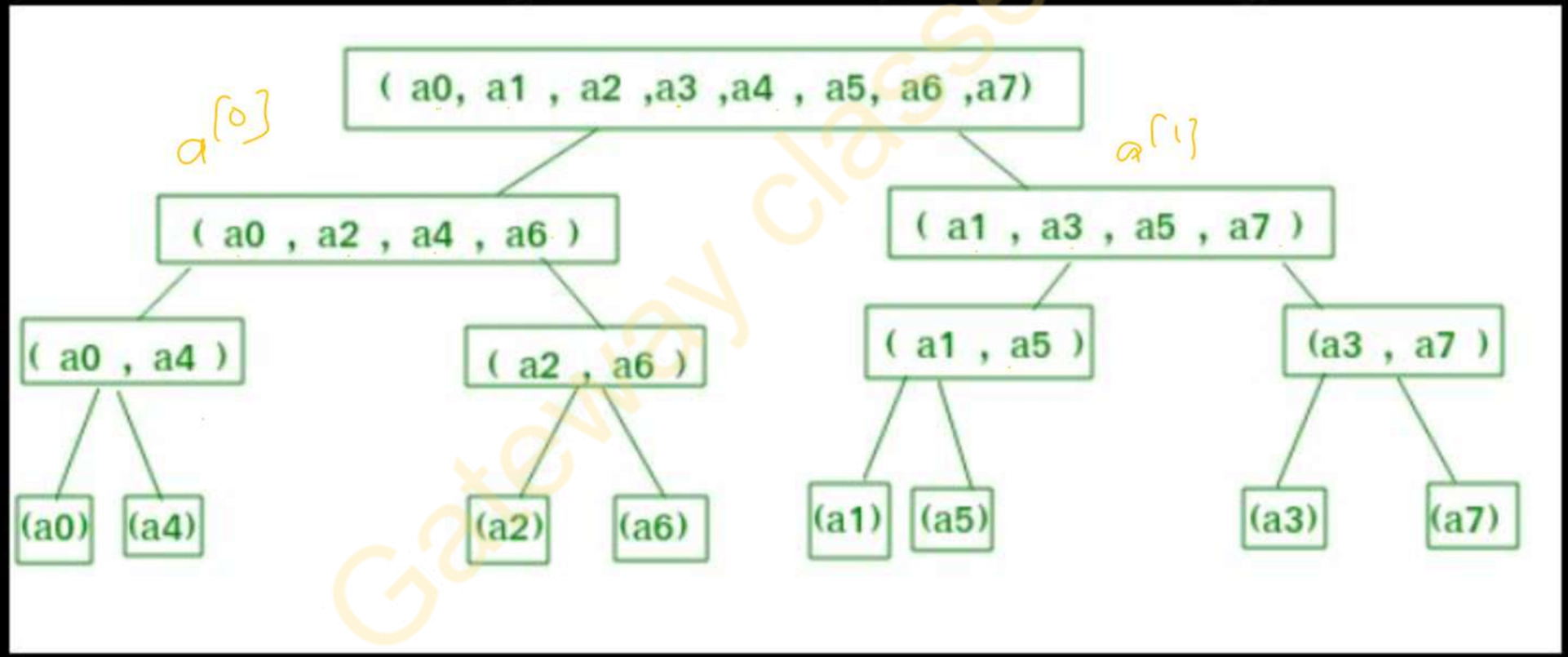
```
FFT (a):  // a is the input coefficient vector
n ← length [a]          //n is a power of 2.
if n == 1
    return a
ω_n ← e^(2πi/n)         // ω_n is the principle complex nth root of unity
ω ← 1
a^[0] ← (a_0, a_2, ... a_(n-2))
a^[1] ← (a_1, a_3, ... a_(n-1))
y^[0] ← FFT (a^[0])
y^[1] ← FFT (a^[1])     // y^[0] and y^[1] are local arrays

For k ← 0 to (n/2) − 1
    y_k ← y^[0]_k + ω y^[1]_k          // y array stores the value of DFT of the given polynomial.
    y_(k+(n/2)) ← y^[0]_k − ω y^[1]_k
    ω ← ω ω_n
return y
```

Divide phase

Merging

Subproblems have exactly the same form as the original problem, but are half the size. So recurrence formed is $T(n) = 2T(n/2) + O(n)$, i.e complexity $O(n\log n)$.



Application of Fast Fourier Transform:

1. Signal processing
2. Image processing
3. Fast multiplication of large integers
4. Solving Poisson's equation nearly optimally

AKTU PYQs

1. What is the application of Fast Fourier Transform (FFT)? Also write the recursive algorithm for FFT.(AKTU 2018-19)
2. Write short note on Fast Fourier transform.(AKTU 2023-24)
3. Explain randomized algorithms.(AKTU 2020-21, 2021-22, 2022-23)