UNIT – V

NP HARD & NP – COMPLETE :-

We can categorize the problems as

1. P – class :-

→ The class P consist of those problems that are solvable in polynomial time. $O(n^k)$ → worstcase.

→ These problems are called tractable.

→ Formally an algorithm is polynomial time algorithm, if there exist a polynomial $p(n)$ such that the algorithm can solve any instance of size n in a time $O(p(n))$.

2. NP – class :-

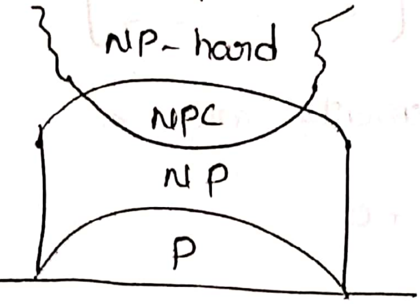
→ The class NP consist of those problems that are verifiable in polynomial time.

→ NP is the class of decision problems for which it is easy to check the correctness of claimed answer, with the aid of little extra information.

→ Hence we are not asking for a way to find a solution but only to verify that an alleged solution is really correct.

Definition of NP-class Problem:- The set of all decision-based problems came into the division of NP Problems, who can't be solved an output within polynomial time but verified in the polynomial time. NP class contains P class as a subset.

Definition of P-class Problem:- The set of decision-based problems come into the division of P Problems who can be solved (or) produced output within polynomial time.



* string matching:-

→ A string matching automation is a very useful tool which is used in string matching algorithm.

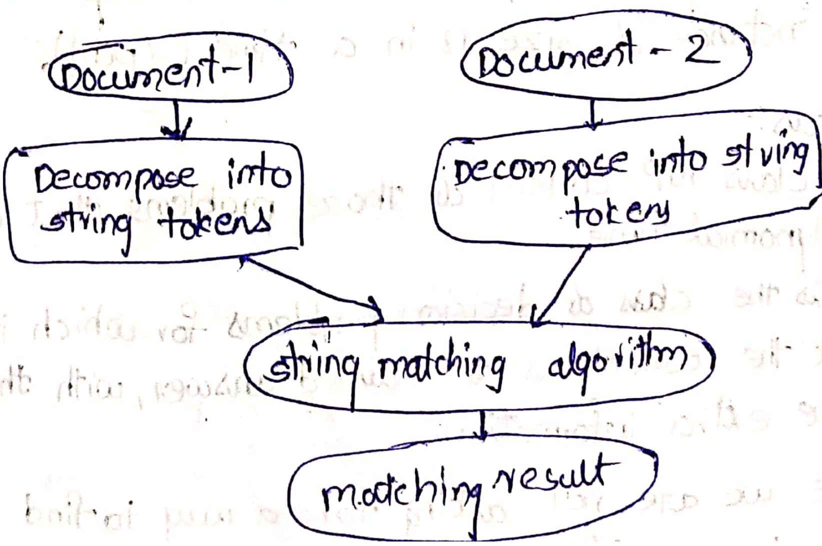
→ It examines every character in the text exactly once & reports all the valid shifts in O(n) time.

→ The ~~good~~ string matching helps in performing time-efficient tasks in multiple domains.

Applications of string matching algorithm:-

→ Plagiarism Detection:-

→ The documents to be compared are decomposed into string tokens & compared using string matching algorithm. It is used to find similarities b/w them.



→ Bioinformatics & DNA sequencing:-

Bioinformatics involves applying information technology & computer science to problems involving genetic sequences to find DNA patterns. string matching algorithms and DNA analysis are both collectively used for finding the occurance of the pattern set

→ Digital Forensics:-

STA are used to locate specific text strings of interest in the digital forensic text.

→ spelling checker:-

Trie is built based on a predefined set of pattern. Then this trie is used for string matching

→ spam filters:-

Spam filters use string matching to discard the spam.

Naive - String - Matching:-

The naive approach tests all the possible placement of pattern P[1.. m] relative to text T[1... n], we try shifts s = 0, 1, ... n−m, successively & for each shift s. Compare T[s+1.....s+m] to P[1. ..m].

Algorithm.

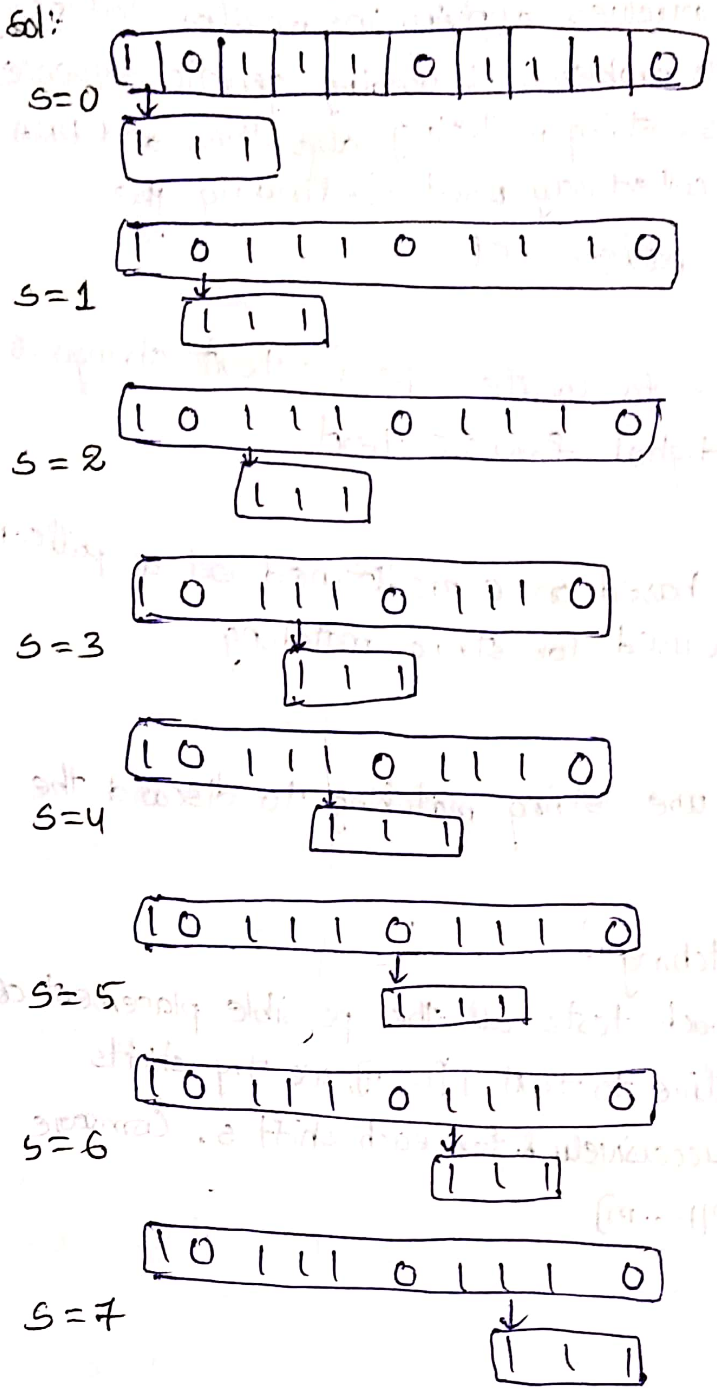
1. n ← length[T]
2. m ← length[P]
3. for s ← 0 to n − m
4. do if P[1... m] = T [s+1 ..... s+m]
5. then print "Pattern occurs with shift" s.

Analysis:-

∴ Total complexity is O(n − m+1).

Example:- Text - 1011101110 P- 111

Sol:-



∴ s2 & s6 are valid shift.

* The Rabin-karp Algorithm:-

→ The rabin karp string matching algorithm calculates a hash value for the pattern, as well as for each m-character subsequences of text to be compared.

→ If the hash values are unequal the algorithm will determine the hash value for next m-character sequence.

→ If the hash values are equal the algorithm will analyze the pattern & the m-character sequence

→ In this way, there is only one comparison per text subsequence & character matching is only required when the hash values match.

Algorithm:

$n \leftarrow length[T]$
$m \leftarrow length[P]$
$h \leftarrow d^{m-1} \bmod q$
$p \leftarrow 0$
$t_0 \leftarrow 0$
for $i \leftarrow 1$ to m
do $p \leftarrow (dp + P[i]) \bmod q$
$t_0 \leftarrow (dt_0 + T[i]) \bmod q$
for $s \leftarrow 0$ to $n-m$
do if $p = t_s$
then if $P[1 \ldots m] = T[s+1 \ldots s+m]$
then "Pattern occurs with shift" s
If $s < n-m$
then $t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$

Example:

$T = 31415926535\ldots$ $P = 26$, T.length = 11 ⇒ Q = 11

$P \bmod Q = 26 \bmod 11 = 4$

Solution:

T → 3 1 4 1 5 9 2 6 5 3 5

P → 2 6

S1: | 3 | 1 | 4 1 5 9 2 6 5 3 5 |

31 mod 11 = 9 ≠ 4

| 3 | 1 | 4 | 1 5 9 2 6 5 3 5 |

14 mod 11 = 3 ≠ 4

| 3 1 | 4 | 1 | 5 9 2 6 5 3 5 |

41 mod 11 = 8 ≠ 4

| 3 1 4 | 1 | 5 | 9 2 6 5 3 5 |

15 mod 11 = 4 = 4 SPURIOUS HIT

3 1 4 1 | 5 | 9 | 2 6 5 3 5

59 mod 11 = 4 = 4 SPURIOUS HIT

3 1 4 1 5 | 9 | 2 | 6 5 3 5

92 mod 11 = 4

3 1 4 1 5 9 | 2 | 6 | 5 3 5

26 mod 11 = 4 Exact match

3 1 4 1 5 9 2 | 6 | 5 | 3 5

65 mod 11 = 10 ≠ 4

3 1 4 1 5 9 2 6 | 5 | 3 | 5

53 mod 11 = 9 ≠ 4

3 1 4 1 5 9 2 6 5 | 3 | 5

35 mod 11 = 2 ≠ 4

∴ The pattern occurs with shift 6.

Analysis:-

for small problems $O(1)$

large " $O(n+m)$

worst case $O((n-m+1)m)$

The Knuth-Morris-Pratt Algorithm:-

→ It introduce a linear time algorithm for the string matching problem.

→ A matching time of $O(n)$ is achieved by avoiding comparison with an element of 's' that have previously been involved in comparison with some element of the pattern 'p' to be matched.

→ components:-

The prefix function (π):-

→ It encapsulates knowledge about how the pattern matches against the shift of itself.

→ This info can be used to avoid a useless shift of the pattern p.

Algorithm:-

The algo placed on proper prefix & suffix

```
m ← length[P]
π[1] ← 0
k ← 0
for q ← 2 to m
do while k > 0 & P[k+1] ≠ P[q]
do k ← π[k]
If P[k+1] = P[q]
then k ← k+1
π[q] ← k
Return π.
```

The KMP matcher:-

with string 's', pattern 'p' & prefix function π as inputs find the occurance of 'p' in 's' & returns the number of shifts of 'p' after which occurances are found.

Time complexity:

$O(m)$ for prefix function $m \to$ times of execution

$O(n)$ for kmp matcher $n \to$ runs

Ex: Compute π for the 'p' below

P: a b a b a c a

Sol: Initially m = length[P] = 7

π[1] = 0 & k = 0

q = 2, k = 0, π[2] = 0

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

q = 3 k = 0 π[3] = 1

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

q = 4 k = 1 π[4] = 2

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

q = 5 k = 2 π[5] = 3

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

q = 6, k = 3, π[6] = 0

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

Scanned with CamScanner

$q = 7 \quad k = 1 \;,\; \pi[7] = 1$

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	0	1

The prefix function computation is comple.

Text b a c b a b a b a b a c a c a

P a b a b a c a

Prefix function

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	0	1

→ Initially n = size of T = 15

m = size of p = 7

S1:- i = 1, q = 0, comparing P[1] with T[1]

T b a c b a b a b a b a c a c a

p a b a b a c a

P[1] does not match with T[1] 'p' will be shifted one position to right.

S2:- T b a c b a b a b a b a c a c a

P a b a b a c a

S:2:- i = 2, q = 0, comparing P[1] with T[2].

T b a c b a b a b a b a c a c a

P a b a b a c a

P[1] matches T[2]. since there is a match p is not shifted.

$s = 4 \quad i = 3, \quad q = 1$

comparing P[2] with T[3], P[2] doesn't match with T[3]

T b a c b a b a b a b a c a c a

P a b a b a c a

Back tracking on p, comparing P[1] and T[3]

$s = 4 \quad i = 4 \quad q = 0$

comparing P[1] with T[4], P[1] doesn't match with T[4]

T b a c b a b a b a b a c a c a

b a b a b a c a

$s = 5 \quad i = 5 \quad q = 0$

comparing P[1] with T[5], P[1] match with T[5]

T b a c b a b a b a b a c a c a

b a b a b a c a

$s = 5 \quad i = 6 \quad q = 1$

comparing P[2] with T[6], P[2] matches with T[6]

T b a c b $\boxed{a}$ b a b a b a c a c a

p $\boxed{a}$ b a b a c a

$s = 5 \quad i = 7 \quad q = 2$

comparing P[3] with T[7], P[3] matches with T[7]

T b a c b $\boxed{a\ b}$ a b a b a c a c a

P $\boxed{a\ b}$ a b a c a

$s = 5 \quad i = 8 \quad q = 3$

comparing P[4] with T[8], P[4] matches with T[8]

T b a c b $\boxed{a\ b\ a}$ b a b a c a c a

P $\boxed{a\ b\ a}$ b a c a

$i = 9, \quad q = 4$

comparing P[5] with T[9], P[5] matches with T[9].

Scanned with CamScanner

step9:- i=9 q=4

comparing P[5] with T[9], P[5] matches with T[9]

T b a c b a b a b a b a c a a b

P a b a b a c a

step10:- i=10, q=5

comparing P[6] with T[10], P[6] doesn't match with T[10]

T:- b a c b a b a b a b a c a a b

P:- a b a b a c a

step11:- i=11, q=4

comparing P[5] with T[11], P[5] match with T[11]

T: b a c b a b a b a b a c a a b

P:- ~~a~~ a b a b a c a

step12: i=12, q=5

comparing P[6] with T[12], P[6] matches with T[12]

T: b a c b a b a b a b a c a a b

P:- a b a b a c a

step 13:- i=3 q=6

comparing P[7] with T[13], P[7] matches with T[13]

T:- b a c b a b a b a b a c a a b

P:- a b a b a c a

Pattern 'P' has been found to complexity occur in a string 'T'. The total number of shifts that took place for the match to be found is $i-m = 13-7 = 6$ shifts