6. String Matching

Q.1 Explain naive approach

→ i) It is simplest method which uses brute force approach.

ii) It is straight forward approach for solving problems.

iii) It compares the first character of pattern with searchable text.

iv) If match is found, pointers in both string are advanced.

v) If match not found, pointer of text is incremented and pointer of pattern is reset.

vi) This process is repeated till the end of text.

vii) It does not require any pre-processing. It directly starts comparing both string characters by character.

viii) The time complexity is = $O(m*(n-m))$

where

n = length of text

m = length of pattern.

ix) Algorithm:

```
NAIVE (T, P)
{
    for i ← 0 to n-m do
        if P[1....m] == T[i+1----i+m] then
            print "Match Found"
}
```

For e.g.,

T = ABCABA

P = CAB

1 2 3 4 5 6

1) T: | A | B | C | A | B | A |

P: | C | A | B |

T[1] ≠ P[1]

t_i++;

2) T: | A | B | C | A | B | A |

P: | C | A | B |

T[2] ≠ P[1]

t_i++;

3) T: | A | B | C | A | B | A |

| C | A | B |

T[3] = P[1]

t_i++, P_i++;

4) T: | A | B | C | A | B | A |

| C | A | B |

T[4] = P[2]

t_i++, P_i++;

5) T: | A | B | C | A | B | A |

| C | A | B | Match Found

T[5] = P[3]

Q.2 Rabin Karp Algorithm
→ i) Comparing numbers is easier and cheaper than comparing strings.
ii) Rabin karp represent string in numbers
iii) It is based on hashing technique.
iv) It first computes the hash values of pattern and text.
v) If the hash values are same, i.e. if hash (p) = hash(t), we check each character if characters are same pattern is found.
vi) If has value are not same no need of compairing.
vii) Strings are compared using brute force approach. If the patter is found, then it is called hit. Otherwise it called spurious hit.
viii) Time complexity : O(mn).
ix) Algorithm :

```
RABIN (T, P)
{
    n = T.length
    m = P.length.
    hp = Hash (P)
    ht = Hash(T)

    for   s=0  to  n-m  do
        if (hp == ht) then
            if (P (0....m-1) == T(0---.m-1))
                print "Match Found"
```

if $s < n-m$

$h_t = Hash(s+1 \ldots s+m-1)$

}

For eg.,

$T = 3145926535$

$P = 59$, $q = 11$

$\Rightarrow$

$P \bmod q = 59 \bmod 11 = 4$

i) $31 \bmod 11 = 9 \neq 4$

ii) $14 \bmod 11 = 3 \neq 4$

iii) $45 \bmod 11 = 9 \neq 4$

iv) $59 \bmod 11 = 4 = 4$ $\Rightarrow$ Exact Match Hit

KMP

8.3 ⇒ i) This is the first linear time algorithm for string matching.

ii) It utilizes the concept of naive approach in some different way.

iii) This approach keeps the track of matched part of pattern.

iv) Main idea of this algorithm is to avoid computation of transition function δ and reducing useless shif performed in Naive approach.

v) This algorithm builds a prefix array. This array is also called as π array.

vi) This algorithm acheives the efficiency of O(m+n) which is optimal in worst case.

• **Algorithm** :

```
KMP(T, P)
{
    n ← length of text
    m ← length of pattern
    π ← PREFIX-FUNCTION(P)
    q ← 0

    for i ← 1 to n do
        while q > 0 AND P[q+1] ≠ T[i] do
            q ← π[q]

        if P[q+1] == T[i] then
            q ← q+1
```

```
if q == m the
    print " Pattern Found "

    q ← π [q]
}

PREFIX (P)
{
    π [1] ← 0
    k ← 0

    for q ← 2 to m do
        while k > 0 AND P[k+1] ≠ P[q] do
            k ← π [k]

        if P[k+1] == P[q] then
            k ← k+1

    π [q] ← k

    return π
}
```