Unit : 7 String Matching Algorithm

* Explain Naive String Matching Algorithm.

=> This Algorithm is used to find the Pattern in the string.

Naive String Matching Algorithm working is easy and simple to understand.

Steps to Perform this Algorithm:

1 Take First string as a T and Take Pattern string as a P.

2 Take Ti and Pj as a indices of String and Patterm.

3 Compare Ti and Pj

If $T_i = P_j$ than
do t_{i++} and P_{j++}

else
t_{i++}

Ex. String : ABCDBRAINXYZ
Pattern : BRAIN

=> Step 1 : Take T = ABCDBRAINXYZ
P = BRAIN

Step 2 :

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	B	R	A	I	N	X	Y	Z

↕

B	R	A	I	N
1	2	3	4	5

T[1] ≠ P[1] So, T++

Step : 3 :

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	B	R	A	I	N	X	Y	Z

↕

B	R	A	I	N
1	2	3	4	5

T[2] = P[1] So, T++ and P++

Step : 4 :

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	B	R	A	I	N	X	Y	Z

↕

B	R	A	I	N
1	2	3	4	5

T[3] ≠ P[2] So, T++

Page No.
Date : / /

Step 5:

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	B	R	A	I	N	X	Y	Z

↕

B	R	A	I	N
1	2	3	4	5

∴ T[4] ≠ P[1] So, T++

Step 6:

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	B	R	A	I	N	X	Y	Z

↕

B	R	A	I	N
1	2	3	4	5

∴ T[5] = P[1] So, T++, P++

Step 7:

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	B	R	A	I	N	X	Y	Z

↕

B	R	A	I	N

∴ T[6] = P[2] So, T++, P++

Step 8:

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	B	R	A	I	N	X	Y	Z

↕

B	R	A	I	N

∴ T[7] = P[3] So, T++, P++

Step 9:

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	B	R	A	I	N	X	Y	Z

I

B	R	A	I	N

∴ T[8] = P[4], So, T++, P++

Step 10:

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	B	R	A	I	N	X	Y	Z

I

B	R	A	I	N

∴ T[9] = P[5] So, T++, P++

Here, Match is Found between Index 5 to 8 of Text String.

=> Time Complexity:

- Best Case:

For this algorithm, If match is not found then this condition become best case.

So, Time Complexity is O(n-m)

- Worst Case :

For this algorithm, If match is found at the last position then it is become worst case.

So, Time Complexity O(m*(n-m))

* Explain Rabin-Karp Pattern Searching Algorithm.

=> Using Rabin-Karp algorithm, we can find the Pattern.

In Algorithm, we have to use rolling hash Function For Find the Pattern.

Steps:

1 In this Algorithm, For every text we have to take one hash value.

2 After select hash value, If there is n text in Pattern then we have to add

One by One value in text.

3 We have to Compare n text hash value and n pattern hash value.

4 IF Hash value is match then again compare the string and text.

5 else move to the other index of text string.

Example :

String : a a a a a b
Pattern : a a b

-> We have to take hash value according to own needs.

We can take any hash value of any string text.

For, a - 1
b - 2
c - 3
d - 4
e - 5

Page No.
Date : / /

Here, Pattern String length is Three. So, we have to add First three element hash value.

Step: 1

String : a a a a a b
Pattern : a a b

String :

For three element S[1] = a + a + a
= 1 + 1 + 1
= 3

For Pattern, three element P[1] = a + a + b
= 1 + 1 + 2
= 4

Here, S[1] ≠ P[1],
So, move to the string next element.

Step: 2

String : a a a a a b
Pattern : a a b

For String S[2] = a + a + a
= 1 + 1 + 1 = 3

For Pattern P[2] = a + a + b
= 1 + 1 + 2 = 4

Here, S[2] ≠ P[2], So, Move to the String next element.

Step 3:

String: a a a a a b
Pattern: a a b

For String S[3] = a + a + a = 3

For Pattern B[3] = a + a + b = 4

Here, S[3] ≠ P[3], So, Move to the String next element.

Step 4:

String: a a a a a b
Pattern: a a b

For String S[4] = a + a + b = 4

For Pattern P[4] = a + a + b = 4

Here, S[4] = P[4]
So, we have to compare the both string.

String : a a b
Pattern : a a b

If Match is Found then, we have to take this index.

else, match is not found in this string.

Here, Pattern is Found at String 4 to 6 index.

-> Drawback:

If Both the String and Pattern hash value is match but text is not match, this is become drawback of this algorithm.

-> Time Complexity :

- Best Case :

For this algorithm, if match

is not found, So this is become best case for this algorithm.

So, Time Complexity is O(n+m)

- Worst Case:

For this algorithm, if match is found at last position, so this is become worst case for this algorithm.

So, Time Complexity is O(m*(n-m))

* Explain String Matching with Finite Automata.

=> Finite Automata can generate multiple string pattern.

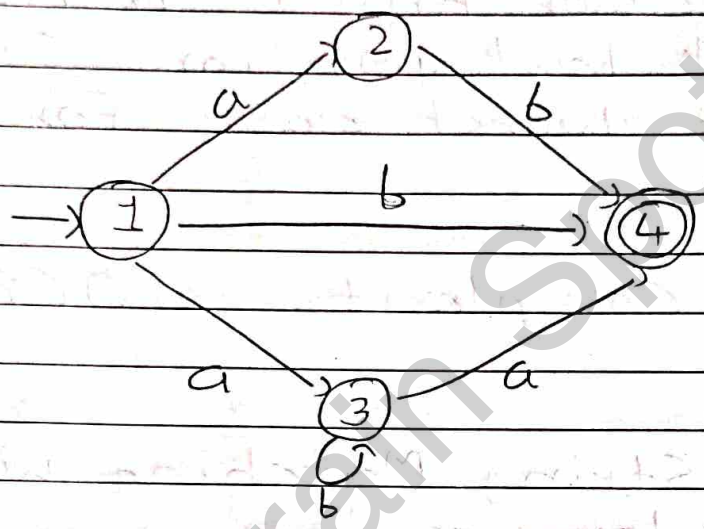
Finite Automata is defined by Five Tuple.

$$M = \{Q, S, q_0, F, d\}$$

Here, Q -> Set of Finite states
S -> Set of input symbol.

q_0 → Initial State
F → Set of Final State
d → Transition Function
$Q \times S \rightarrow Q$

- Example :



Here, $Q = \{1, 2, 3, 4\}$
$S = \{a, b\}$
$q_0 = \{1\}$
$F = \{4\}$

Transition Table :

d	a	b
1	2, 3	4
2	-	4
3	4	3
4	-	-

Here, Time Complexity of Algorithm is O(n).

* Explain Knuth Morris Pattern Matching Algorithm.

=> Knuth Morris Pattern Matching Algorithm is most effective algorithm for Pattern Matching.

In this Method, we have to create Π table For Given Pattern which you have to search in given text or string.

Π table :

For Create Π table, we have Given index value to every Pattern string character.

After this step, we have to match every character.

If character is match then give this character index number

We have to start match From left to Right.

If match is not found then give 0 in π table.

We have start with First character and move with next character.

If character is repeat then given previous match character index value.

Ex.

1	2	3	4	5
a	b	a	b	d

π Table

0	0	1	2	0

Steps To Perform :

1. For Text string take i variable with starting with 1 index and For Pattern string take j variable with 0 index.

2. Create π table For Pattern String.

3. Compare T[i] with P[j+1]

   if match is found then move i and j Right side

If MisMatch then Move J as per Π table location.

4 If J = 0 then move i to the Right side.

Ex. Text : a b a b c a b a b a b d
~~a b a b d~~
Pattern : a b a b d

⇒ Π table For Pattern String :

	0	1	2	3	4	5
		a	b	a	b	d
Π Table :	0	0	1	2	0	

Step 1 : Compare T[i] and P[j+1]

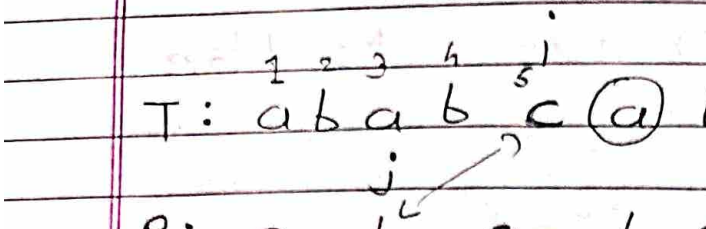
T : a b a b c a b a b a b d
1 2 3 4 5 6 7 8 9 10

P : a b a b d
0 1 2 3 4 5

Here, T[4] and P[4] is match but T[5] ≠ P[5]

So, we have to take P[4] value which is store in Π table.

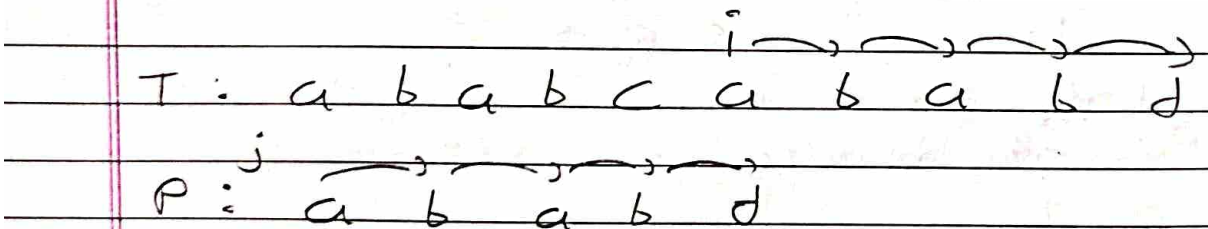
Step 2 : Value of P[4] = 2
So, j is Present at 2 location.



Compare T[5] and P[2]
Again not math So, we have to move P[2] ~~T~~ table location which is zero.
So, we have to move i location by one.

So, location of i is T[6] and location of j is P[0].

Step 3 : Comape T[6] with P[j+1].



Here, Pattern is Found in Text string.

So, Match Pattern Present at location i = 6 to 10