Unit-8 :- String matching

DOMS | Page No. 1
Date / /

* Topics :-

- Introduction
- The naive string matching algorithm
- The Rabin-Karp algorithm
- String matching with finite automata
- The Knuth-Morris-Pratt algorithm.

* Introduction :-

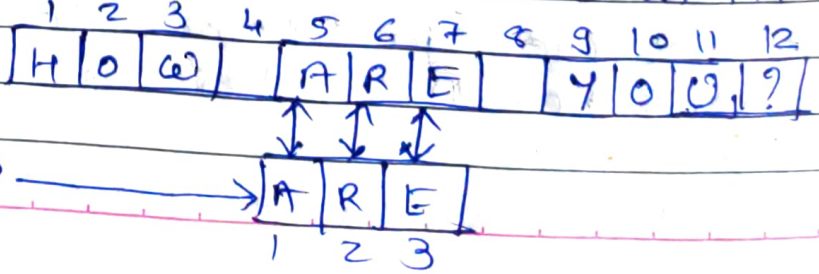
- String matching operation is core part in many text processing applications.

→ Objective of string matching algorithm is to find pattern P from given text T.

- Typically $|P| << |T|$.

- In design of compilers and text editors, string matching operation is crucial. So locating P in T efficiently is very important.

→ Problem :- Given some text string T[1...n] of size n, find all occurrences of pattern P[1...m] of size m in T.



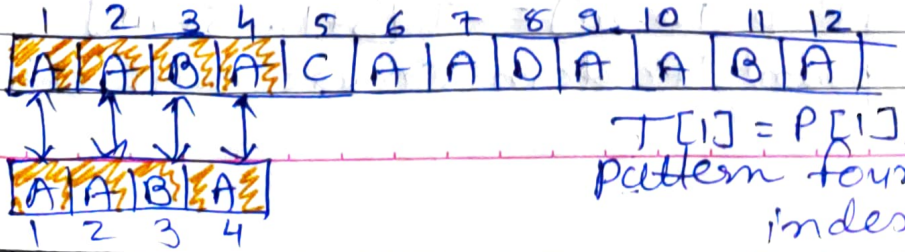
* Naive String Matching Algorithm :-

- This is simple and inefficient brute force approach.
- It compares first character of pattern with searchable text. If match is found, pointers in both strings are advanced.
- If match is not found, pointer of text is incremented and pointer of pattern is reset. This process is repeated till the end of text.
- Naive approach does not require any pre-processing.
- Given text T and pattern P, it directly starts comparing both strings character by character.
- After each comparison, it shifts pattern string one position to the right.

⇒ Example :-

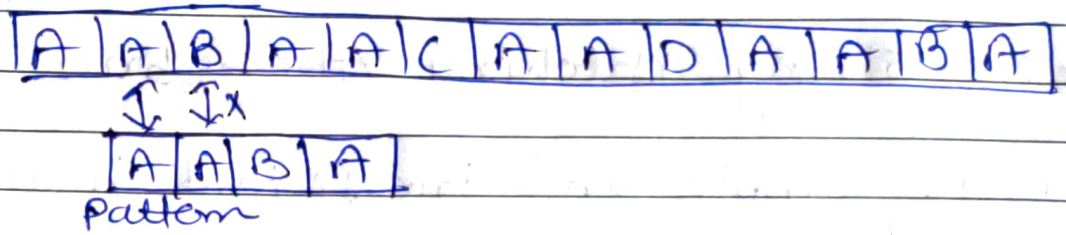
Text : A A B A A C A A D A A B A
Pattern : A A B A

⇒



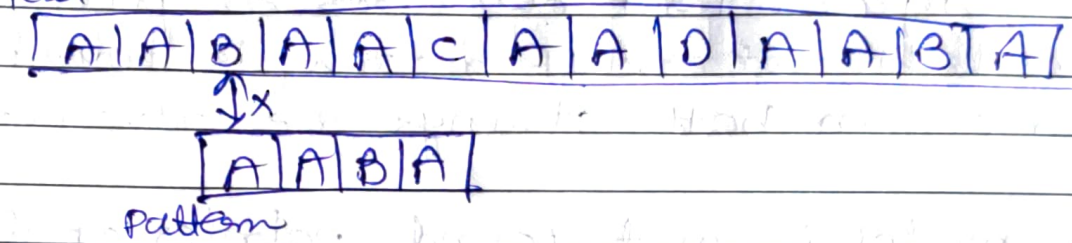
T[1] = P[1], T[2] = P[2], T[3] = P[3]
pattern found at index 1 T[4] = P[4]

Text

- 

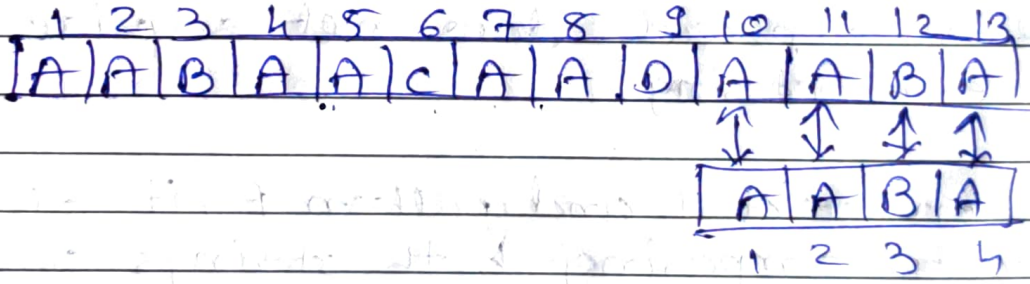
then shift one position right.

Text

- 

- we shift one position right till end.

- At last,



- So, pattern found at position 8 to 10.

⇒ Algorithm :-

```
Naive-string Matching (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
```

⇒ Time Complexity :- $O((n-m+1)m)$

* Rabin-Karp Algorithm :-

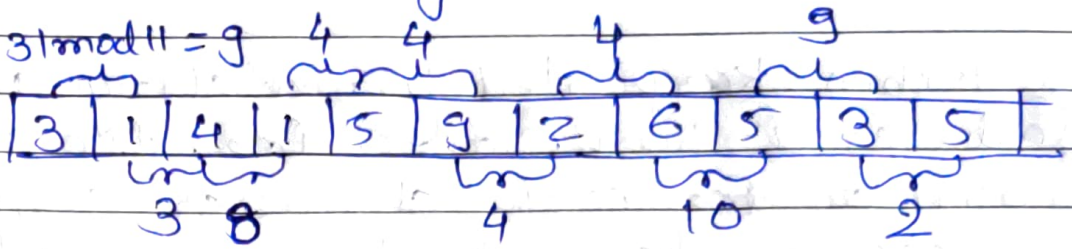
- The Rabin-Karp string matching algo. calculates a hash value for the pattern, as well as for each character sub-sequences of text to be compared.
- If the hash values are unequal, the algorithm will determine the hash value for next character sequence.
- If the hash values are equal, the algorithm will analyze the pattern and the character sequence.
- In this way, there is only one comparison per text subsequence, and character matching is only required when the hash values match.
- If pattern is found then it is called hit. otherwise it is called spurious hit.

⇒ Example :-

T = 3141592653 5
P = 26
9 = 11

- First find P mod q = 26 mod 11
  = 4

- Next finding hash values of Text



- 15 mod 11 = 4 equal to 4
  is spurious hit
  but string is not match
  15 ≠ 26

- 59 mod 11 = 4 equal to 4
  is spurious hit
  but string is not match
  59 ≠ 26

- 92 mod 11 = 4 equal to 4
  but string is not match
  92 ≠ 26
  is spurious hit

- 26 mod 11 = 4 equal to 4
  exact match 26 = 26
  (text) (pattern)
  is called hit.

→ Pattern found at position 7.

⇒ Algorithm :-

```
Rabin-Karp matching (T, P, d, q)

n ← T.length;
m ← P.length;
h ← d^(m-1) mod q;
p ← 0;
to ← 0;

for i ← 1 to m do
    p ← (dp + P[i]) mod q
    to ← (dto + T[i]) mod q

for s ← 0 to n-m do
    if p == ts then
        if P[1...m] == T[s+1...s+m] then
            print "pattern occurs with
                                shift" s

    if s < n-m then
        ts+1 ← (d(ts - T[s+1]h) + T[s+m+1]) mod q
```

* String matching with Finite Automata :-

- In this approach, build finite automata to scan text T for finding all occurrences of pattern P.

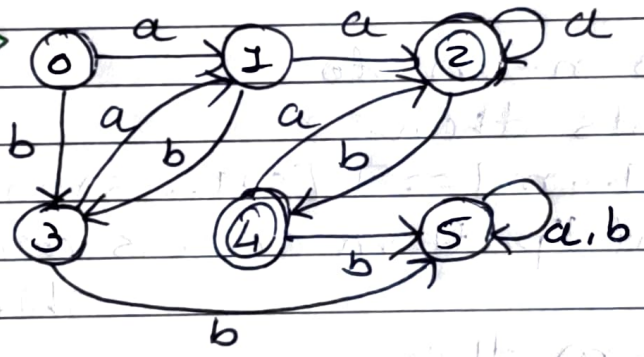
- This approach examines each character of text exactly once to find the pattern.

- Thus it takes linear time for matching but preprocessing time may be large.
- finite automata is defined by triple,

$$M = \{Q, \Sigma, q_0, F, \delta\}$$

where, Q = Set of states in finite automata
Σ = set of input symbols
q_0 = Initial state
F = Set of final states
δ = Transition function

Example:



$Q = \{0, 1, 2, 3, 4, 5\}$
$q_0 = 0$
$F = \{2, 3\}$
$\Sigma = \{a, b\}$

Transition function:

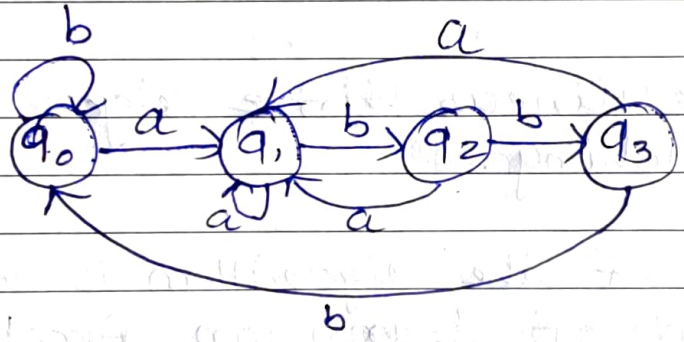
δ	a	b
0	1	3
1	2	3
2	2	4
3	1	5
4	2	5
5	5	5

⇒ Algorithm :-

```
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
```

⇒ Example :-

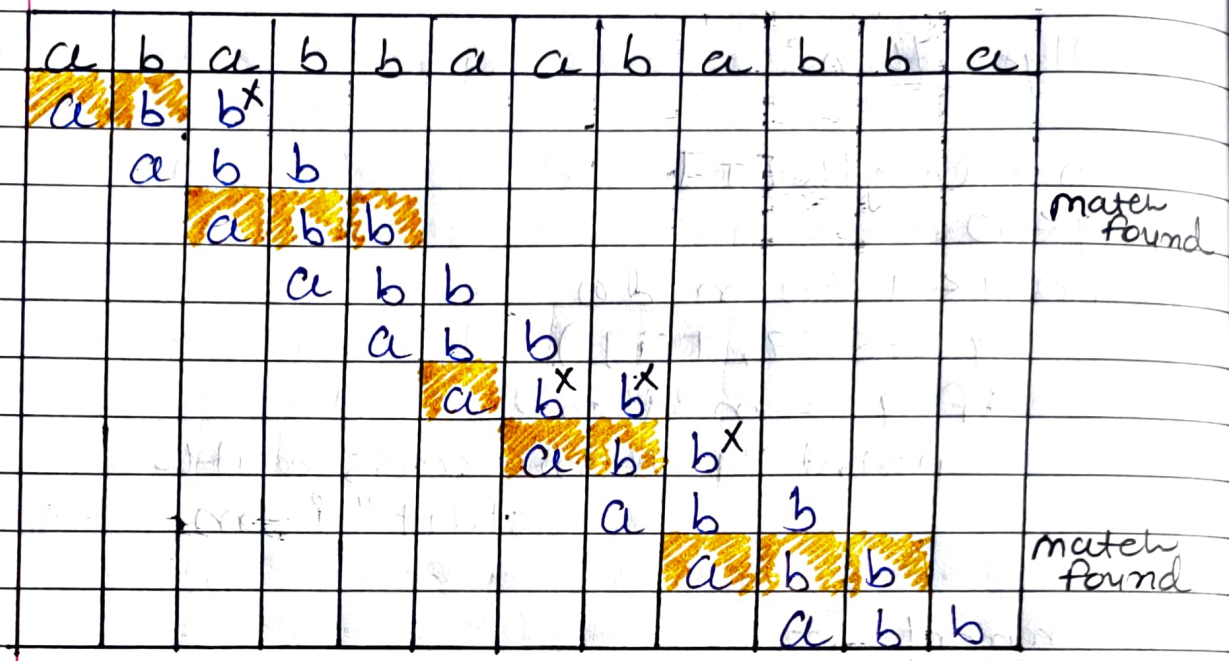
text T = ababbaabbabba
pattern P = abb



$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$
q_0 → starting state
q_3 → End state

Transition table (δ) :-

δ	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_1	q_0



* The Knuth-Morris-Pratt Algorithm :- (KMP Algo.)

- KMP is the linear time algorithm for string matching.

→ Main idea of the algorithm is to avoid computation of transition function δ and reducing useless shifts performed in naive approach.

- This improvement is achieved by using auxiliary function Π.

- Function Π is very useful, it computes the numbes of shifts of pattern by finding pattern matches within itself.

⇒ Example :-

~~text T = bacbcabcabcaceb~~

text T = bacbcabcabcaceb.
pattern P = abcabcacca

→ First find out prefix table,

↳ Consider string "a"

0	1	2	3	4	5	6
a	b	a	b	a	c	a
0						

prefix = ∈
suffix = ∈

↳ Consider string "ab"

0	1	2	3	4	5	6
a	b	a	b	a	c	a
0	0					

Prefix = ^, a
Suffix = ^, b

↳ Consider string "aba"

0	1	2	3	4	5	6
a	b	a	b	a	c	a
0	0	1				

prefix = ^, a, ab
suffix = ^, a, ba

↳ Consider string "abcab"

0	1	2	3	4	5	6
a	b	a	b	a	c	a
0	0	1	2			

prefix = ^, a, ab, abca
suffix = ^, b, ab, abca

↳ Consider string "ababa"

prefix = ^, a, ab, aba, abab
suffix = ^, a, ba, aba, abab

0	1	2	3	4	5	6
a	b	a	b	a	c	a
0	0	1	2	3		

↳ Considers string "ababac"

prefix = ^, a, ab, aba, abab, ababa
suffix = ^, c, ac, bac, abac, babac

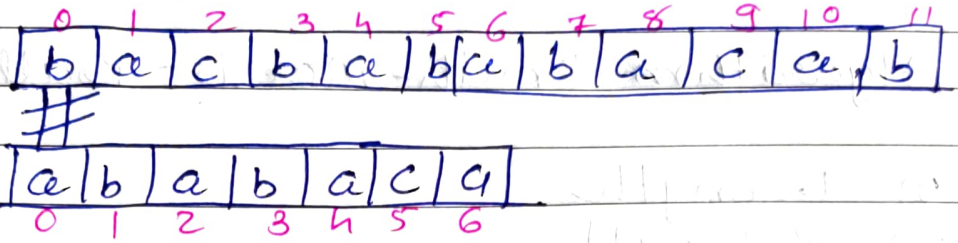
0	1	2	3	4	5	6
a	b	a	b	a	c	a
0	0	1	2	3	0	1

↳ Consider string "ababacca"

prefix = ^, a, ab, aba, abab, ababa, ababac
suffix = ^, a, ca, cca, bacca, abacca, babacca

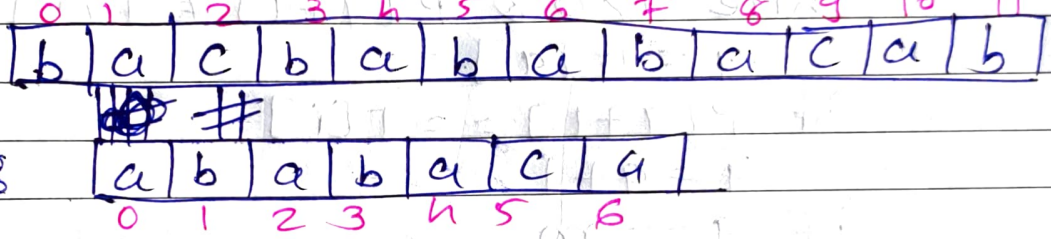
0	1	2	3	4	5	6
a	b	a	b	a	c	a
0	0	1	2	3	0	1

- Now compute T[0] with P[0]



here $a \neq b \Rightarrow$ check prefix table at $\pi[0] = 0$; take P[0] compute T[1] with P[0]

- T:

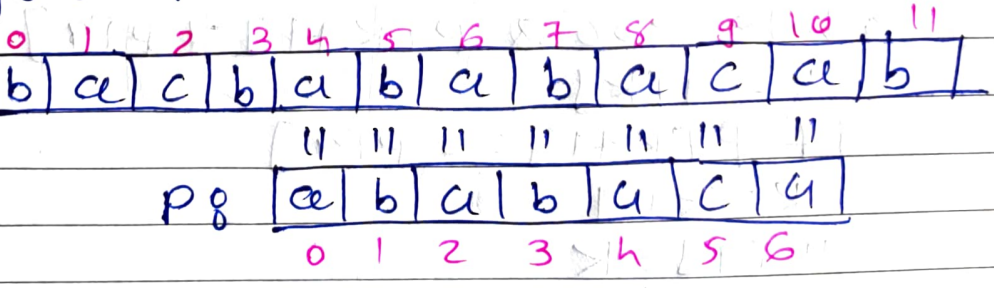


Compare T[2] with P[1]
No match $c \neq b$

- Now, P[0] with T[2] compare, No match,

- Now, compare T[3] with P[0] $\Rightarrow$ No match

- Compare P[0] with T[4]



- Pattern found at T[4].

⇒ Algorithm :-

```
Algorithm Knuth-Morris-Pratt (T, P)

n = T.length
m = P.length
π = Compute-prefix-function (P)
q = 0
for i = 1 to n
    while q > 0 & P[q+1] ≠ T[i]
        q = π[q]
    if P[q+1] == T[i]
        q = q+1
    if q == m
        print "occurs shift" i - m
        q = π[q]

Compute-Prefix (P)

m = P.length
let π[1...m] be a new array
π[1] = 0
K = 0
for q = 2 to m
    while K > 0 & P[K+1] ≠ P[q]
        K = π[K]
    if P[K+1] == P[q]
        K = K+1
    π[q] = K
return π
```