AOA Assignment: String Matching Algorithms

Submitted by:

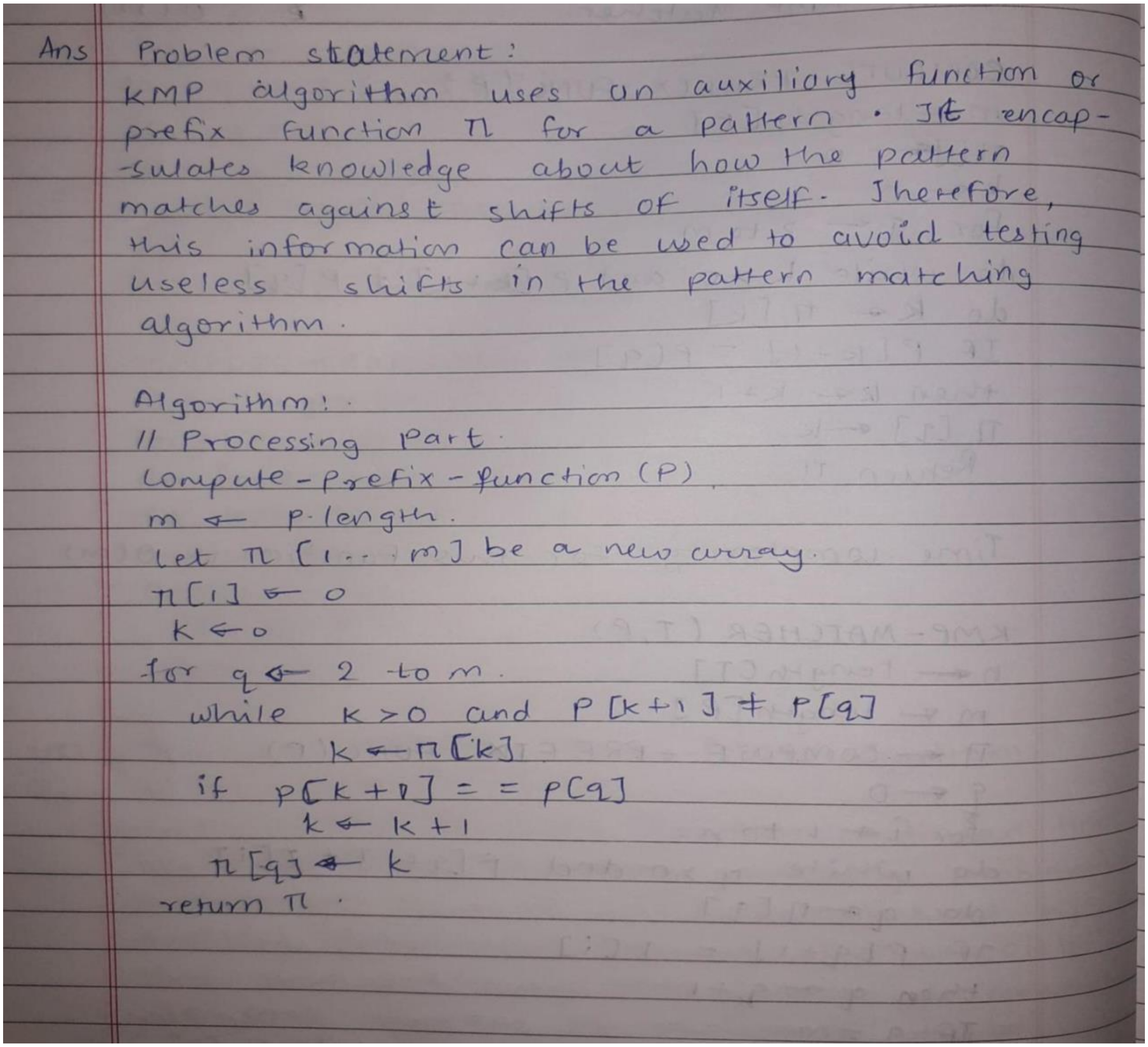
Shawn Louis

Roll no. 31

Batch : B

1.Rewrite KMP algorithm and explain with example

Ans:

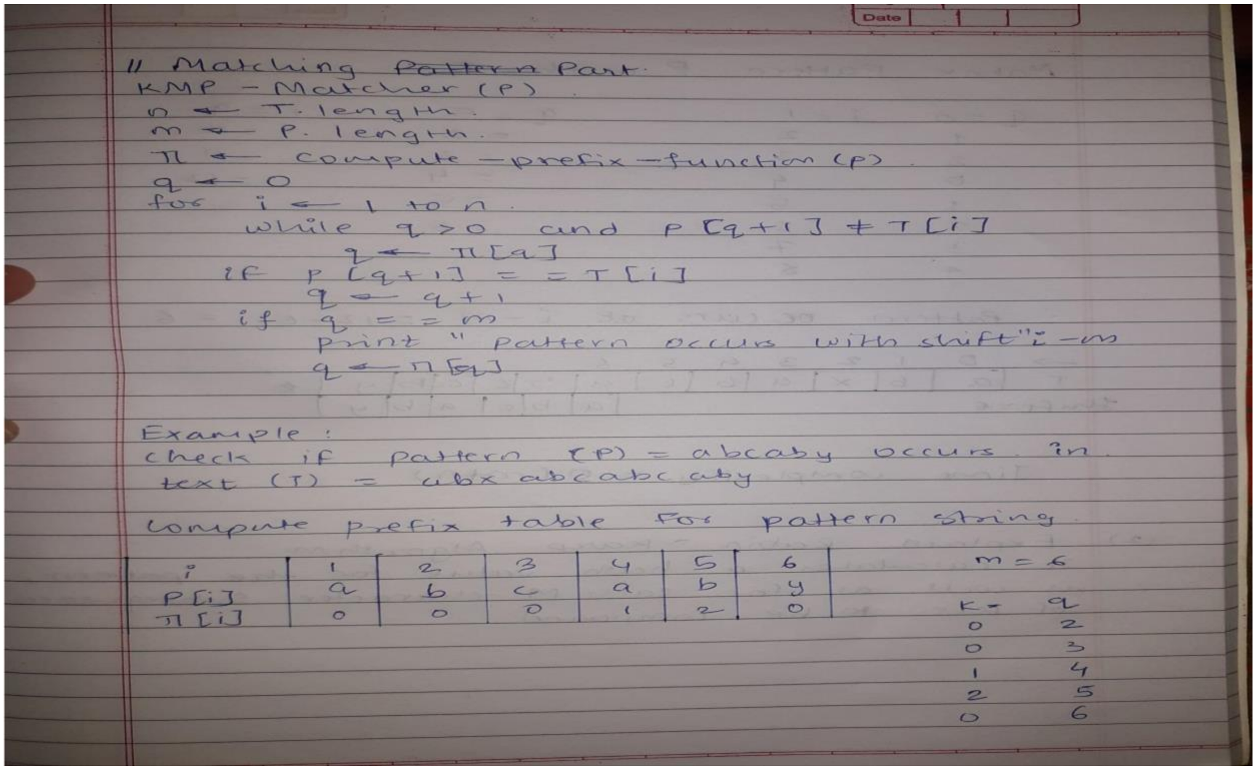


Ans Problem statement:

KMP algorithm uses an auxiliary function or prefix function π for a pattern. It encapsulates knowledge about how the pattern matches against shifts of itself. Therefore, this information can be used to avoid testing useless shifts in the pattern matching algorithm.

Algorithm:

```
// Processing part.
Compute-Prefix-function (P)
m ← p.length.
let $\pi[1..m]$ be a new array.
$\pi[1] \leftarrow 0$
$k \leftarrow 0$
for $q \leftarrow 2$ to m.
    while $k > 0$ and $P[k+1] \neq P[q]$
        $k \leftarrow \pi[k]$
    if $P[k+1] == P[q]$
        $k \leftarrow k+1$
    $\pi[q] \leftarrow k$
return π.
```



Date

// Matching ~~Pattern~~ Part.

```
KMP - Matcher (P).
n ← T. length.
m ← P. length.
π ← compute - prefix - function (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]
```

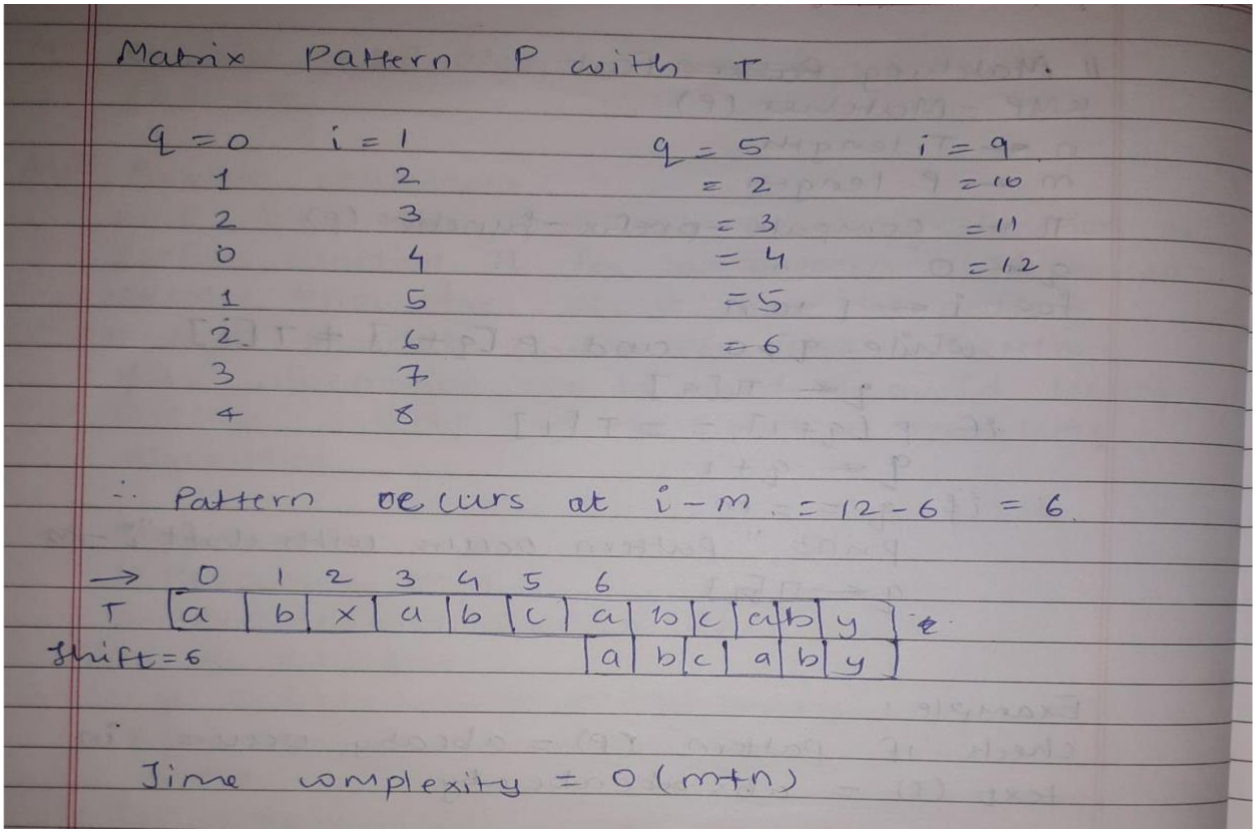
Example :
check if pattern (P) = abcaby occurs in text (T) = abx abcabc aby

compute prefix table for pattern string.

i	1	2	3	4	5	6
P [i]	a	b	c	a	b	y
π [i]	0	0	0	1	2	0

m = 6

k -	q
0	2
0	3
1	4
2	5
0	6



Matrix Pattern P with T.

q = 0	i = 1
1	2
2	3
0	4
1	5
2	6
3	7
4	8

q = 5	i = 9
= 2	= 10
= 3	= 11
= 4	= 12
= 5	
= 6	

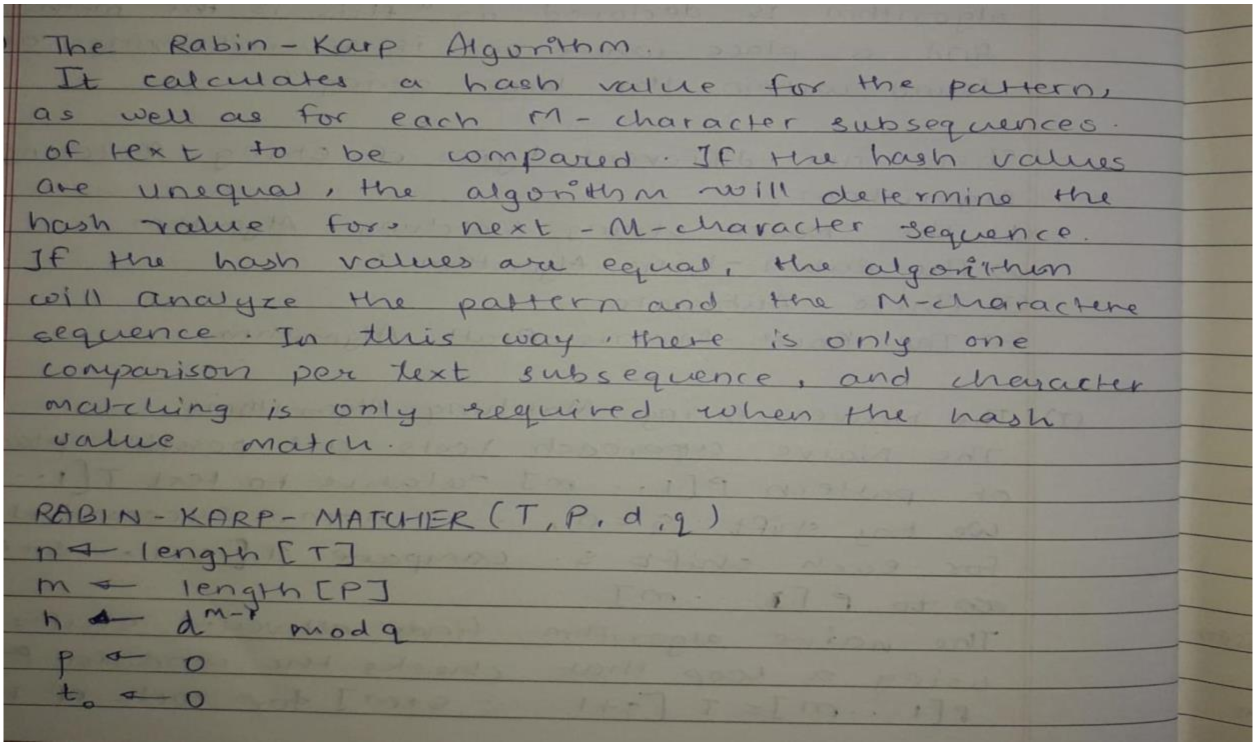
∴ Pattern occurs at i - m = 12 - 6 = 6.

	0	1	2	3	4	5	6					
T	a	b	x	a	b	c	a	b	c	a	b	y
Shift = 6							a	b	c	a	b	y

Time complexity = O (m+n)

2. Explain Rabin- Karp Algorithm

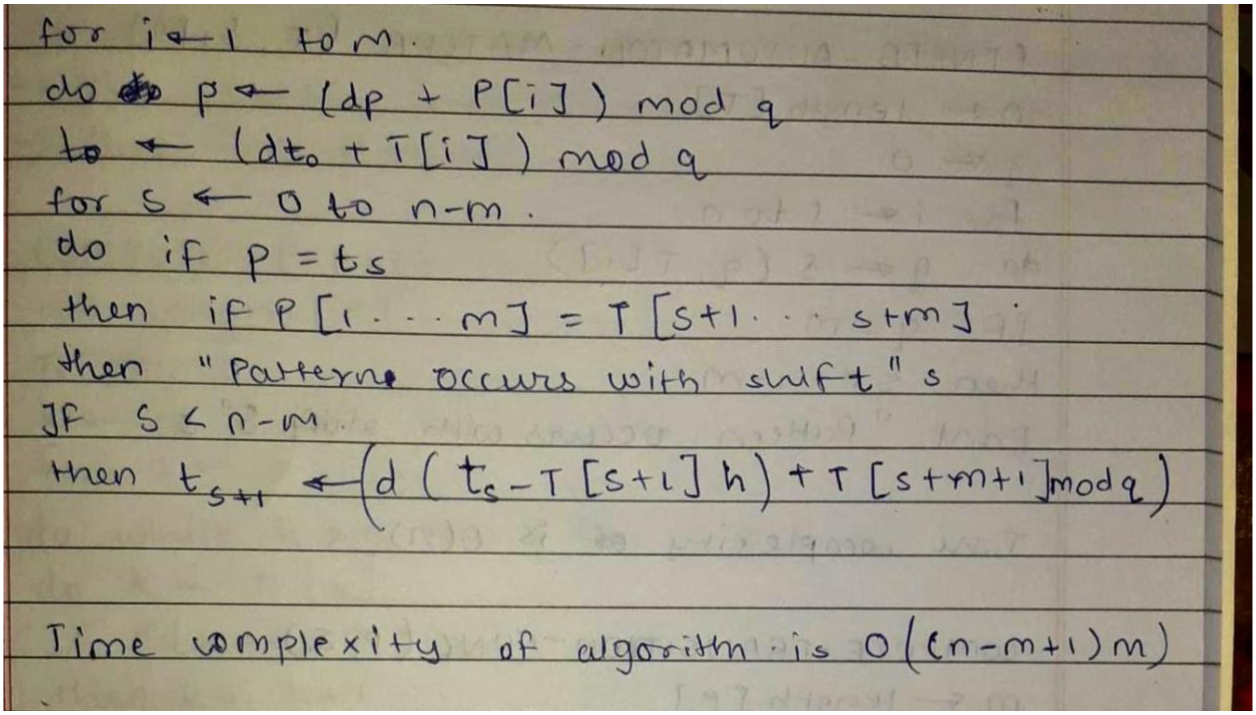
Ans



The Rabin-Karp Algorithm.

It 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 and the M-charactere sequence. In this way, there is only one comparison per text subsequence, and character matching is only required when the hash value match.

```
RABIN-KARP-MATCHER (T, P, d, q)
$n \leftarrow$ length [T]
$m \leftarrow$ length [P]
$h \leftarrow d^{m-1} \bmod q$
$p \leftarrow 0$
$t_0 \leftarrow 0$
```

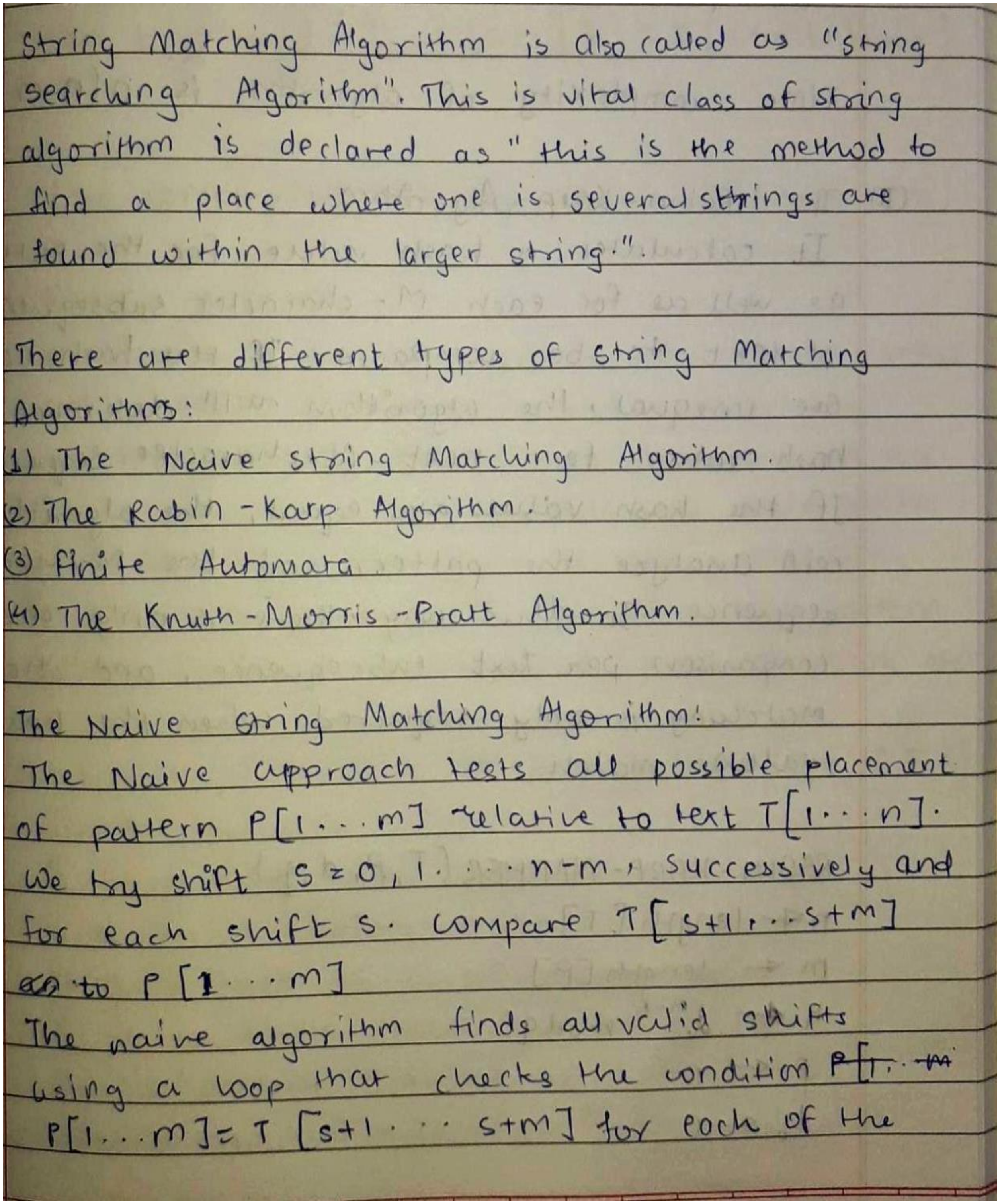


```
for $i \leftarrow 1$ to m.
do $p \leftarrow (dp + P[i]) \bmod q$
 $t_0 \leftarrow (dt_0 + T[i]) \bmod q$
for $s \leftarrow 0$ to $n-m$.
 do if $p = t_s$
  then if $P[1 \ldots m] = T[s+1 \ldots s+m]$
  then "Pattern occurs with shift" s
If $s < n-m$
then $t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1]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

Time complexity of algorithm is $O((n-m+1)m)$

3.Explain Various string Matching algorithms

Ans:



String Matching Algorithm is also called as "string searching Algorithm". This is vital class of string algorithm is declared as " this is the method to find a place where one is several strings are found within the larger string".

There are different types of string Matching Algorithms:

(1) The Naive string Matching Algorithm.

(2) The Rabin - Karp Algorithm.

(3) Finite Automata

(4) The Knuth - Morris - Pratt Algorithm.

The Naive string Matching Algorithm:

The Naive approach tests all possible placement of pattern P[1 . . . m] relative to text T[1 . . . n]. We try shift s = 0, 1 . . . n-m, successively and for each shift s. compare T[s+1 . . . s+m] ~~as~~ to P[1 . . . m]

The naive algorithm finds all valid shifts using a loop that checks the condition ~~P[1 . . m~~ P[1 . . . m] = T [s+1 . . . s+m] for each of the n-m+1 possible values of S.

```
NAIVE-STRING-MATCHER (T, P)
 n ← length[T]
 m ← length [P]
 for s ← 0 to n-m.
 do if P[1...m] = T [s+1...s+m]
 then print "Pattern occurs with shift "s
```

Time complexity of algorithm is O(n-m+1)

The Rabin-Karp Algorithm.

It 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 and the M-charactere sequence. In this way, there is only one comparison per text subsequence, and character matching is only required when the hash value match.

```
RABIN-KARP-MATCHER (T, P, d, q)
 n ← length [T]
 m ← length [P]
 h ← $d^{m-1}$ mod q
 p ← 0
 $t_0$ ← 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 $P[1 \cdots m] = T[s+1 \cdots s+m]$
then "Patterne occurs with shift" s
If $s < n-m$.
then $t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1] \bmod q)$
```

Time complexity of algorithm is $O((n-m+1)m)$

FINITE AUTOMATA.

The string-Matching automaton is very useful tool which is used in string Matching algorithm. It examines every character in the text exactly once and reports all valid shifts in $O(n)$ time. The goal of string matching is to find location of specific text pattern within large text (sentence, paragraph, book etc)

A finite Automaton ~~M~~ M is a 5 tuple $(Q, q_0, A, \Sigma, \delta)$ where

Q is a finite set of states.

$q_0 \in Q$ is starting state.

$A \subseteq Q$ set of accepting states.

Σ is a finite input alphabet.

δ is function defined as $\delta : Q \times \Sigma \rightarrow Q$ called transition function of M.

```
FINITE-AUTOMATON-MATCER (T, δ, M)
n ← length [T]
q ← 0
for i ← 1 to n.
do   q ← δ (q, T[i])
If   q = m.
then   s ← i-m
print "Pattern occurs with shift S" s.
```

Time complexity ~~of~~ is O(n).

```
COMPUTE-TRANSITION-FUNC (P, Σ)
m ← length [P]
for q ← 0 to m.
do for each character a ∈ Σ*.
do   k ← min (m+1, q+2)
repeat k ← k-1
Untill
  δ(q, a) ← k
  Return δ.
```

The (KMP) Knuth-Morris-Pratt Algorithm.

Knuth-Morris & Pratt 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 pattern 'p' to be matched. i.e., backtracking on string 's' neve occurs.

components of KMP Algorithm:	Time complexity
→ The Prefix Function (π):	O (m)
→ The KMP Matcher.	O (n)

```
COMPUTE-PREFIX-FUNC (P)
m ← length [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
```

Time complexity for this Function is O(m).

```
KMP-MATCHER (T, P)
n ← length [T]
m ← length [P]
π ← COMPUTE-PREFIX-FUNC (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.
then print "Pattern Occurs with shift" i-m
q ← π [q]
```