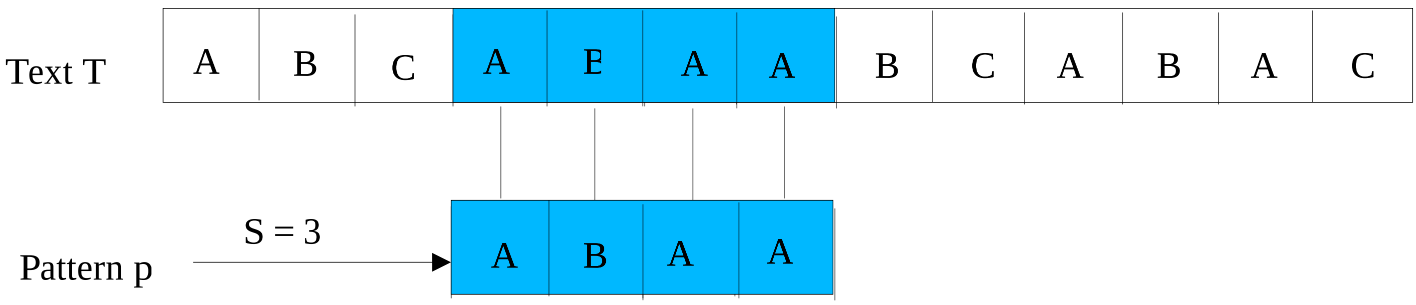
Problems in String Matching and Pattern Recognition

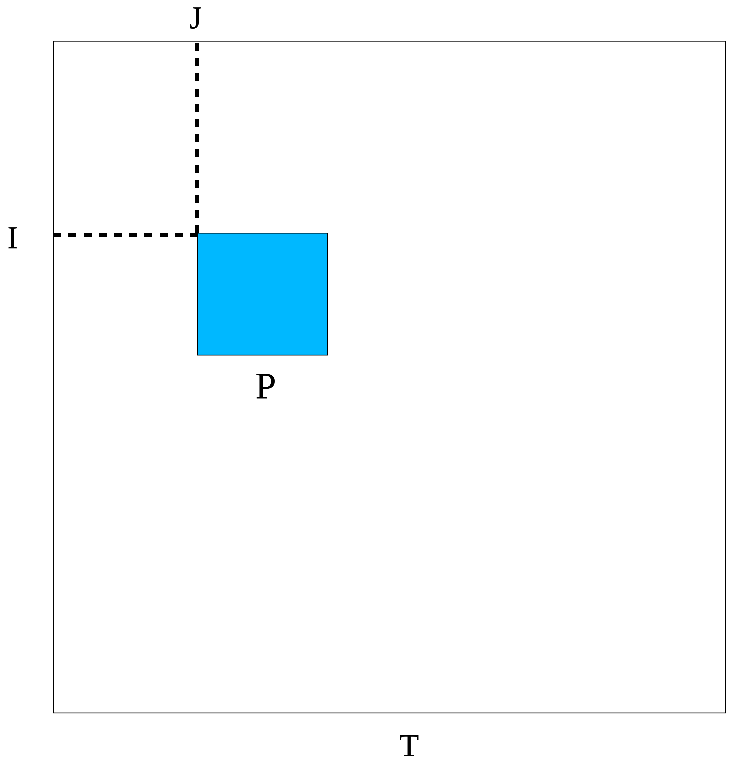
Dr. Gur Saran Adhar

Reference clrs, Chapter 32, Page 906-
Robert Sedgewick, Chapter 19
Udi Manber Page 148-

Problem Statement: String Matching

Given a **text** $T[1 \ldots n]$ of length n and a **Pattern** $P[1 \ldots m]$ of length $m \leq n$, where the elements of T and P are drawn from an alphabet Σ, the **string matching problem** is to find an occurrence of the pattern within the text.


Text T
A
B
C
A
B
A
A
B
C
A
B
A
C
S = 3
Pattern p
A
B
A
A


J
I
P
T

List Of Common String Matching Algorithms

1.	Naive String-Matching Algorithm	clrs 909-
2.	Rabin Karp Algorithm	clrs 911-
3.	String Matching with Finite Automata	clrs 916-
4.	Knuth-Morris-Pratt (KMP) Algorithm	clrs 923-
5.	Boyer-Moore (BM) Algorithm	Sedgewick 28

Notation: String Matching

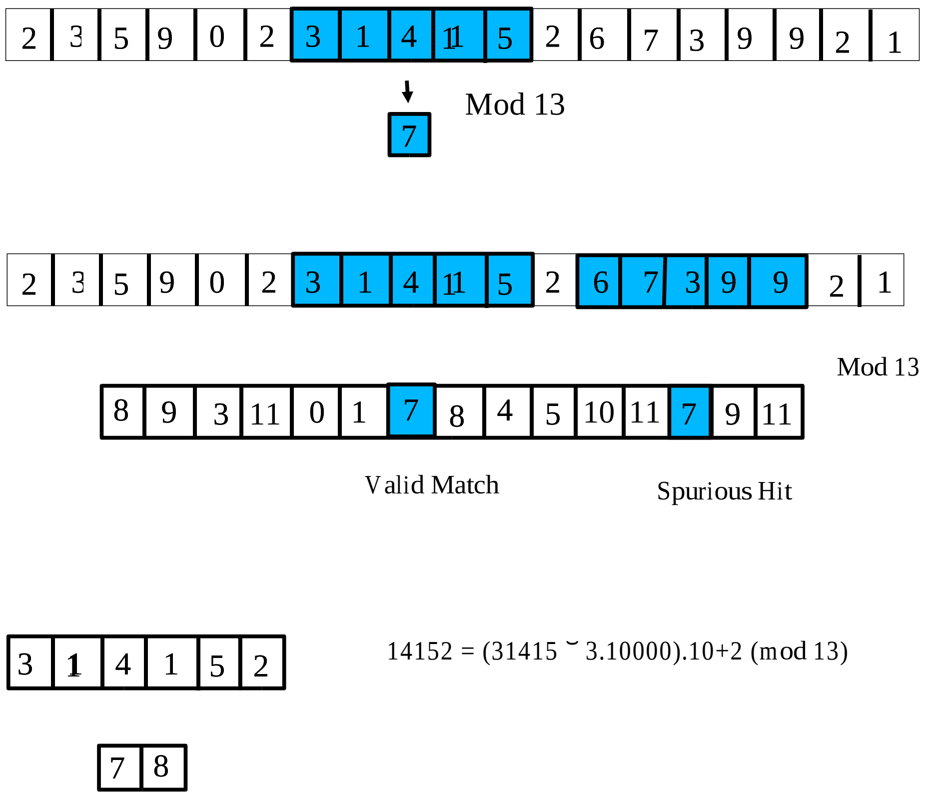
Prefix: w is a **prefix** of string x, denoted $w \sqsubset x$, if $x = wy$ for some string $y \in \Sigma^*$.

Suffix: w is a **suffix** of string x, denoted $w \sqsupset x$, if $x = yw$ for some string $y \in \Sigma^*$.

Naive String Matcher(T, P)

$n \leftarrow length[T]$
$m \leftarrow length[P]$
for $s \leftarrow 0$ to $n - m$
    do if $P[1 \ldots m] = T[s + 1 \ldots s + m]$
        then print "Pattern occurs with shift" s

`Reference clrs 909`

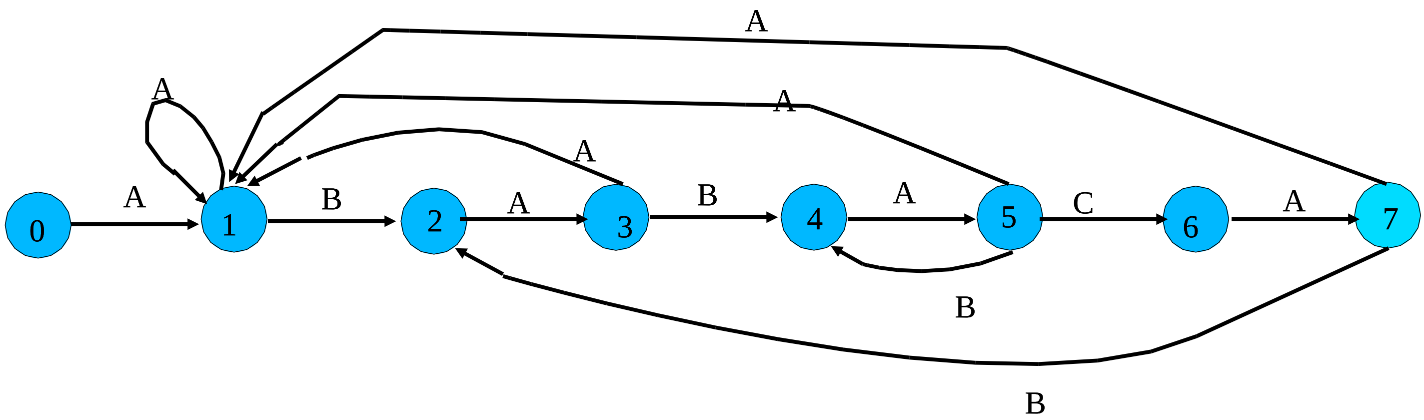

2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1
Mod 13
7
2 3 5 9 0 2 3 1 4 1 5 2 6 7 3 9 9 2 1
Mod 13
8 9 3 11 0 1 7 8 4 5 10 11 7 9 11
Valid Match
Spurious Hit
3 1 4 1 5 2
14152 = (31415 ˘ 3.10000).10+2 (mod 13)
7 8

// p denotes the decimal value of the pattern $P[1 \ldots m]$.
// t_s denotes the decimal value of the m-length substring
// $T[1 + s \ldots s + m]$.

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 ← 1 to m
    do p ← (dp + P[i]) mod q
       t_0 ← (dt_0 + T[i]) mod q
for s ← 0 to n − m
    do if p = t_s
          then if P[1 ... m] = T[s + 1 ... s + m]
                  then print "Pattern occurs with shift" s
       if s < n − m
          then t_{s+1} ← (d(t_s − T[s + 1]h) + T[s + m + 1]) mod q
```

Reference clrs 909



	INPUT A	**B**	C	P -the pattern
0	1	0	0	A
1	1	2	0	B
2	3	0	0	A
3	1	4	0	B
4	5	0	0	A
5	1	4	6	C
6	7	0	0	A
7	1	2	0	

i---		1	2	3	4	5	6	7	8	9	10	11
I[i]---		A	B	A	B	A	B	A	C	A	B	A
State---	0	1	2	3	4	5	4	5	6	7	2	3

Suffix Function: σ for a pattern $P[1 \ldots m]$ is the *length* of the longest prefix of P that is a suffix of x:

$$\sigma(x) = \max\{k : P_k \sqsupset x\}$$

The function σ is a mapping from Σ^* to $\{1, 2, \ldots, m\}$ such that $\sigma(x)$ is the length of the prefix of P which is a suffix of x.

`Observe: In the example Finite Automaton` $\delta(5, b) = 4$, `this transition is made because if the automaton reads a` b `in state` $q = 5$, `then` $P_q b = ababab$, `and the longest` **prefix of** $P = ababab$ **which is also a suffix of** $ababab$ **is** $P_4 = abab$.
The transition function δ is defined by the following equation, for any state q and any character a.

$$\delta(q, a) = \sigma(P_q a)$$

Finite-Automaton-Matcher(T, δ, m)

```
n ← length[T]
q ← 0
for i ← 1 to n
    do q ← δ(q, T[i])
        then if q = m
                then print "Pattern occurs with shift" i − m
```

Reference clrs 919

COMPUTE_TRANSITION_FUNCTION(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
                until P_k ⊐ P_q a
                δ(q, a) ← k
return δ
```

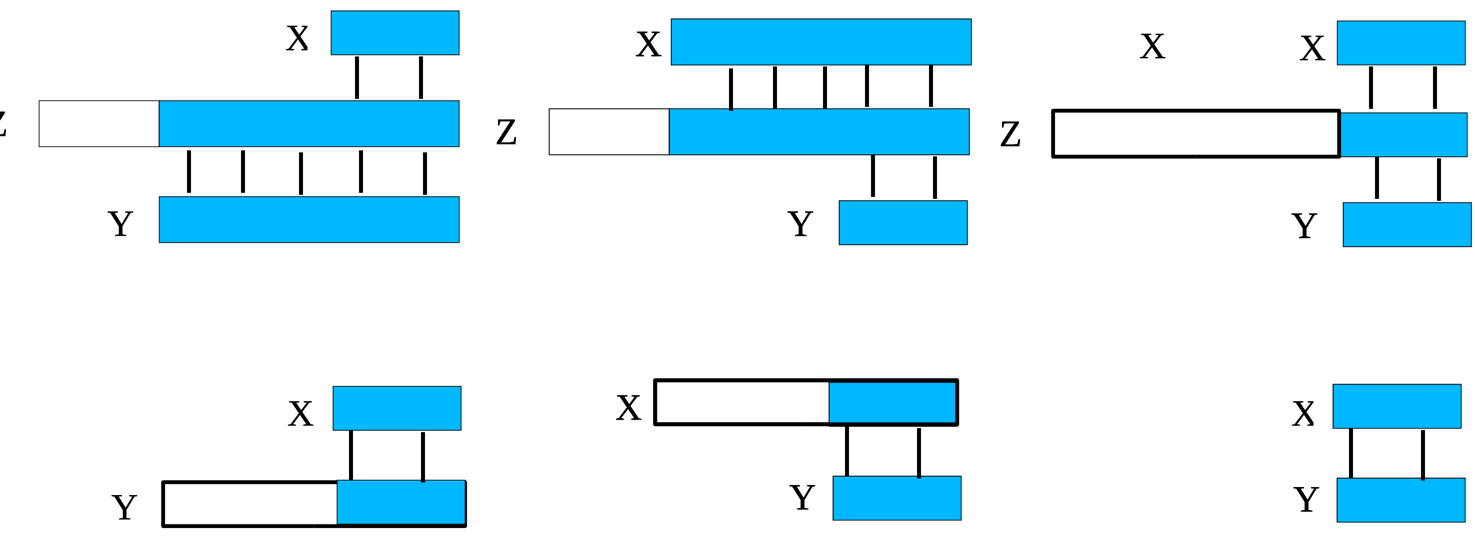
Reference clrs 922

Observation: Overlapping suffixes

Suppose x, y, and z are strings such that, $x \sqsupset z$ and $y \sqsupset z$. That is, both x and y are suffix of z.

If $|\ x\ | \leq |\ y\ |$ then $x \sqsupset y$.
That is, x is suffix of y
If $|\ x\ | \geq |\ y\ |$ then $y \sqsupset x$.
That is, y is suffix of x
If $|\ x\ | = |\ y\ |$ then $x = y$.
That is, x is equal to y

Also note, for any strings x and y and any character a if $x \sqsupset y$ then $xa \sqsupset ya$.
That is if x is suffix of y then so is xa of ya.


X
Z
Y
X
Z
Y
X
X
Z
Y
X
Y
X
Y
X
Y

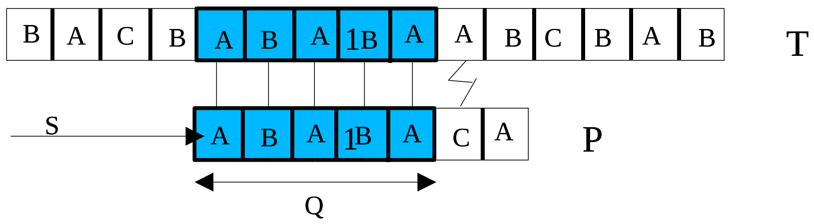
Knuth Morris Pratt (KMP) Algorithm Main Idea

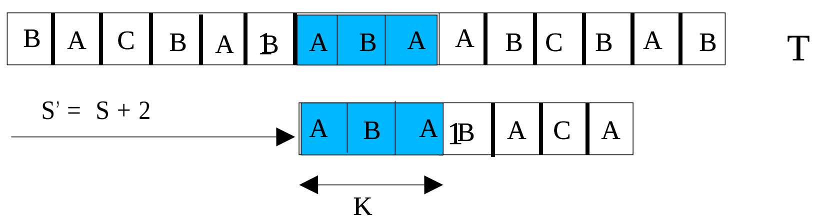
Prefix Function:

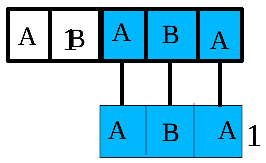
Prefix function π encapsulates the knowledge about how a pattern matches against shifts of itself. This avoids useless shifts in the naive algorithm.

$\pi[5]$ for example gives the *next potentially valid shift* given that five characters have matched (and the sixth does not match).

Reference clrs 923-


B A C B A B A 1B A A B C B A B
T
S
A B A 1B A C A
P
Q


B A C B A 1B A B A A B C B A B
T
S' = S + 2
A B A 1B A C A
K


A 1B A B A
A B A 1

Prefix Function: π for a pattern $P[1 \ldots m]$ is the function $\pi : \{1, 2, \ldots m\} \rightarrow \{0, 1, 2, \ldots m-1\}$ such that:

$$\pi(q) = \text{max}\{k : k < q \text{and} P_k \sqsupset P_q\}$$

That is $\pi[q]$ is the length of the longest prefix of P that is a proper suffix of P_q.

KMP-Matcher(T, P)

```
n ← length[T]
m ← length[P]
π ← COMPUTE_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
        then print "Pattern occurs with shift" i − m
            q ← π[q]
```

Reference clrs 926

COMPUTE_PREFIX_FUNCTION(P)

$m \leftarrow length[P]$
$\pi[1] \leftarrow 0$
$k \leftarrow 0$
for $q \leftarrow 2$ to m
    do while $k > 0$ and $P[k+1] \neq P[q]$
        do $k \leftarrow \pi[k]$
    if $P[k+1] = P[q]$
        then $k \leftarrow k + 1$
    $\pi[q] = k$
return π

Reference clrs 926