{ Pattern matching }

string Matching Algorithm

we were given text (string) of length 'n'

T = | a | b | a | b | . . . |

have to match a pattern

P = | a | b | Length can be of 'm' size

* so pattern p is a subset of T
* characters of text & P are derived from finite alphabate Σ (sigma)

$\Sigma = (0,1)$

$\Sigma = (a, b \ldots z)$

shift

example से समझते हैं।

T | a | b | c | a | b | a | a | b | c . .

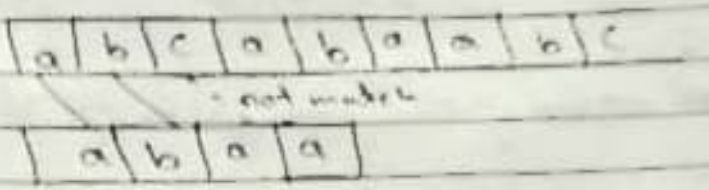
Match the pattern 'P'

P | a | b | a | a |

concept of shift is we start matching string from 0

Brief - 2.

s = 0



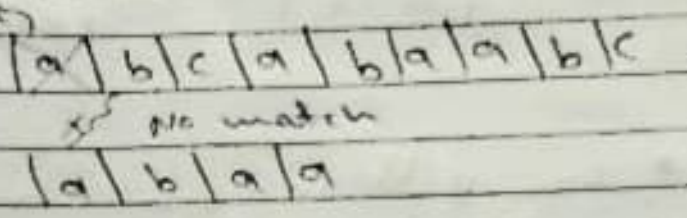
not match then s = s+1

s = 1 now

which means we not consider the 1st element from text (string)

we consider

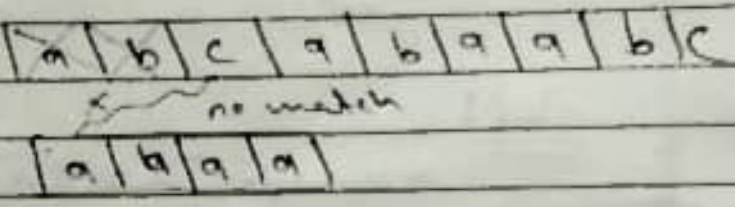
s = 1



s = s+1

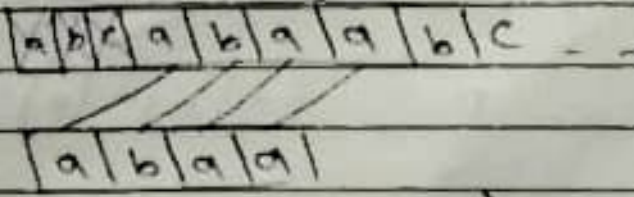
s = 1+1 = 2

s = 2



s = s+1

s = 3



pattern matched at s = 3

boundaries of s $0 \le s \le n-m$

(2)

Naive string matching Algorithm

* simplest method for pattern matching

$T[s+1 \ldots s+m] = P[1 \ldots m]$

$T[s+j] = P[j]$ $\boxed{1 \le j \le m}$

Example

Algorithm

```
n = T.length
m = P.length
for s = 0 to n-m
    if P[1..m] = T[s+1 .. s+m]
        print("found");
```

Eg. 1

T: | a | c | a | a | b | c |

P: | a | a | b |

s = 0

a c a a b c
| ✗ no match s++
a a b

s = 1

a | c a a b c
✗ s++
a a c

s = 2

a|c| a a b c

a a b found printed

KNUTH - MORRIES - PRATT Algorithm

* It works on proper prefix and proper suffix
* In this text में backtrac नहीं होता

proper suffix - start from rhs.

proper prefix - L.H.S. से start

ex. abcd ⟶ P.S.
↓
P.P.

P.P. = { a, ab, abc }

P.S. = { d, cd, bcd }

* Let say Text length = m
  pattern length is = n
  complexity होती है O(m+n)

* If the begining part of pattern is appearing anywhere in pattern or not that is to be observed.

find π table / Longest Proper Prefix
→ to avoid useless shift we use it
only pattern is considered to generate π table.

	1	2	3	4
P_1:	a	b	a	b
	0	0	1	2

S1 - write index

	1	2	3	4	5	6	7	8	9	10
P_2	a	b	c	d	a	b	e	a	b	f
	0	0	0	0	1	2	0	1	2	0

P_3: a a a a b a a c d   S2 - repeat है तो
0 1 2 3 0 1 2 0 0   location लिखें

example to solve KMP Algo

	i														
T	a	b	a	b	c	a	b	c	a	b	a	b	a	b	d
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

j

	0	1	2	3	4	5
P		a	b	a	b	d
	starting point	0	0	1	2	0

S1

i = 1
j = 0   if (T[i] == P(j+1))
i++;
j++;

a b a b c a b c a b a b a b d
j Index,
0 a b a b d

move j as well as i ;

i

a b a b c a b c a b a b a b d

j

a b a b d

here s(i) = b
p(j+1) = b match

move i & j

(i)

a b a b c a b c a b a b a b d

a b a b d

(j)

here s(i) = a
p(j+1) = a match i & j++

(i)

a b a b c a b c a b a b a b d

a b a b d

(j)

s(i) = b
p(j+1) = b match i & j++

i

a b a b c a b c a b a b d

a b a b d

j

s(i) = c
p(j+1) = b not match

Now move j to its π index which is 2

(i)

a b a b c a b c a b a b a b d
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

(ii)

a b a b d,
c

i = c
j+1 = a not match

a b ab repeating
so maybe we have to start from next

i

a b a b c a b c a b a b a b d

a b a b d

(iii)

s(i) = c
p(j+1) = a
no match
increase i

(iv)

a b a b c a b c a b a b a b d

a b a b d

s(i) = a
p(j+1) = a match
i++; j++;

i

a b a b c a b c a b a b a b d

a b a b d
j

s(i) = b
p(j+1) = b match