Pattern Matching

1. i/p → Geeks for Geeks , eks
   Big string — pattern is small string.

   o/p → 2 , 10

2. i/p → AAAAA , AAA

   o/p → 0, 1, 2

3. ABC ABCD

   ABD.

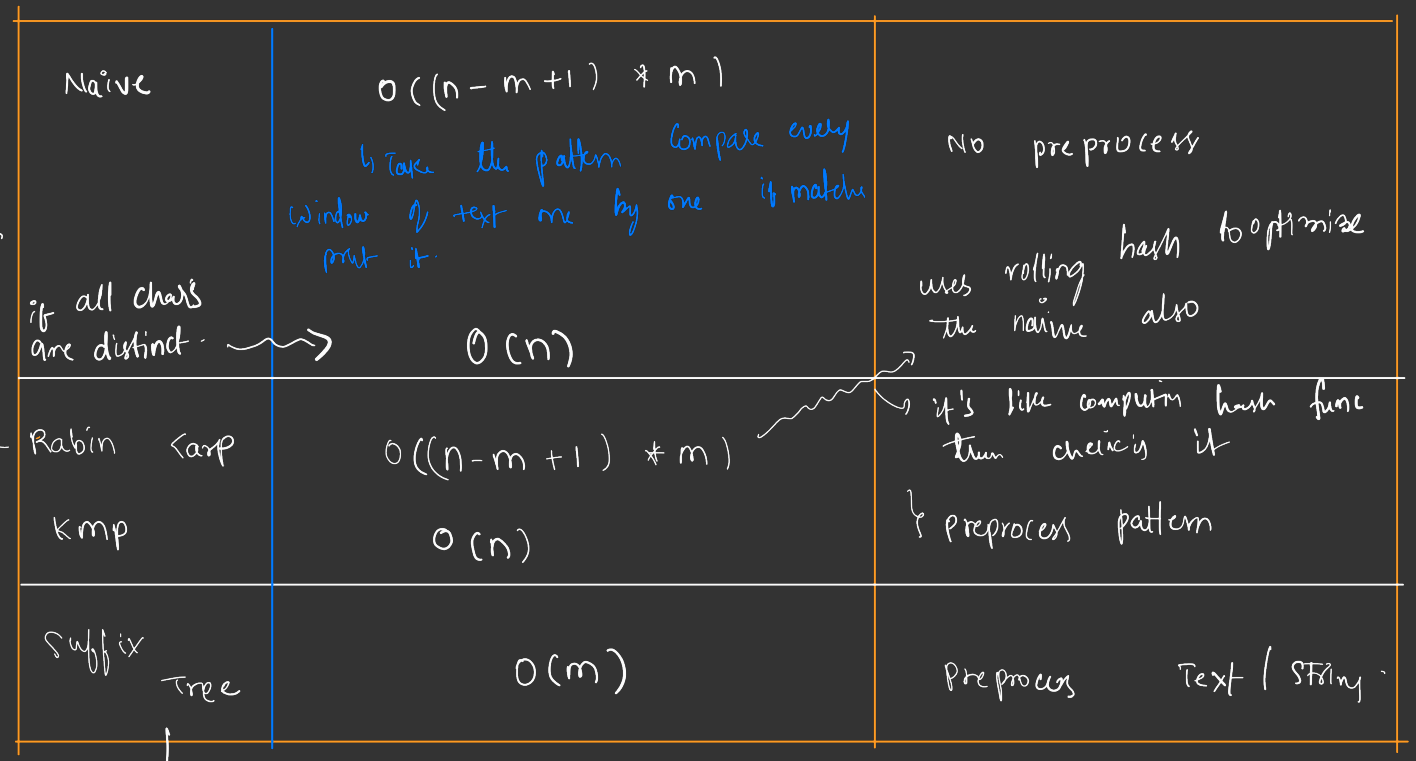
   o/p → -1.

↳ it's like finding the pattern in the given string.

Overview of pattern Searching :

m → pattern length

n → String length.



→ Naive.

↳ We need to Find & return all indexes where we we find the pattern.

① txt = ABABABCD ⇒ 0, 2

ptrn = ABAB.

② AAAAA ⇒ 0, 1, 2.

AAA

AAAAA ⇒ 5

AAA. ⇒ 3

↳ We need to check for 3 times i.e. $i \to [0, n-m]$ inclusive.

① check for next 3 chars = m.

```
for i → 0 → n-m+1 :
    j = 0
    for j in → 0 → m :
        (  )
    if j == m :
```

(n-m+1) * m

pattern[j] == string[i+j]

↳ j=0

j=1

j=2

AABAAA
A AA
A A AAA
A A A
AAA AA
AA A

↳ Incremented 'j' As we found match.

if j == m :

print (i)

index.

Improved Naive Algo For Distinct.

↳ in pattern.

0 1 2 3 4 5 6 7 8 9 10 11

A B C E A B E F A B C D

n = 12

A B C D

m = 4

i = 0 : i += 3 & j

i = 3 :

i = 4 : i += 2

i = 6 :

i = 7 :

i = 8 :

if we go in T loop we (skip the outer loop and vice versa)

O(n) ?

2 matches move ahead of 2

A B C E A B A B C D

A B C D

As we have covered till C, we dont need to check again for that as we know that the chars are distinct

-> we can skip & move ahead.

```
i = 0
while  i <= n - m + 1 :

    J = 0
    while  j != m :
        if  str[i+j]  !=  pattern[j]:
            break
        J++;
    if  j == m :                    # pattern found.
        print (i)

    if  J == 0 :                    # if no match.
        i += 1
    else :                          # if we matched something
        i = i + j
```

Rabin Karp Algo. :- rolling Hash func.

→ like naive algo, slides the pattern one by one in str / text.

→ Compare hash values of pattern & current text window. If hash values match then only Compare invidual chars.

a : 1
b : 2
c : 3
d : 4
e : 5

0 1 2 3 4 5 6 7 8 9
txt = a b d a b c b a b c
pat = a b c
0 1 2

9 - 2 = 7

p = 1+2+3 = 6 = Computation of hash val of pattern

i=0	t = 1+2+4 = 7
=1	t = 2+4+1 = 7
2	t = 4+1+2 = 7
3	t = 1+2+3 = 6 (match)
4	t = 2+3+2 = 7
5	t = 3+2+1 = 6 (spurious Hit)
6	t = 2+1+2 = 5
7	t = 1+2+3 = 6 (match)

p = Hash val of pattern

t = Hash val of current window of text.

→ Simple Hash : Sum of values

→ problem : Spurious hits.

god dog ⇒ both give same value.

5 - 2
c 3+3
str[i] ∑
str[i+m]
6+3 = 9

We can go with ascii vals also, we will get the spurious hits... in - that case also.

How to Calculate hash value??

↳ Compute pattern hash.

↳ Compute Cure Hash For First 3 Chars = m.
(pattern length.)

↳ for i → n-m+1 :

if CureHash == P :

[ ] → check if it's spurious hit or not

if not print it..

if i < n-m :

cure Hash = Cure Hash − ord(str[i]) + ord(str[i+m]).

0 1 2 3 4 5 6 7

ABAB ABCD

AB.

8 - 2 = 6

p = 3

t = 3

i = 0

i = 1

i = 2

i = 3

i = 4

i = 5

i = 6

curr = ord(str(i)) + ord(str(i+m)) / 2

next hash =>

t = 3 => ✓

t = 3 - 1 + 1 = 3 => ✗

t = 3 - 2 + 2 = 3 => ✓

t = 3 - 2 + 2 => ✗

3 - 2 + 2 => ✓

3 - 1 + 3 => 5

5 - 2 + 4 => 7

$t_{i+1} = t_i + str[i+m] - str(i)$.

Rolling Hash.

improved hash.

d = 5 (we compute powers of them). (weighted sum):

a : 1
b : 2
c : 3
d : 4
e : 5

$h(abc) = 1 \times d^2 + 2 \times d + 3 \times d^0 = 25 + 10 + 3 = 38$

$h(dab) = 4 \times d^2 + 1 \times d + 2 \times d^0 = 107.$

Rolling hash func:

$t_{i+1} = d(t_i - t \times txt[i] \times d^{m-1}) + txt[i+m].$

	0	1	2	3	4	5	6	7	8	9
txt =	a	b	d	a	b	c	b	a	b	c

pat = abc
$= 1 \times 5^2 + 2 \times 5 + 3$
$= 38.$

$t_0 = 1 \times 5^2 + 2 \times 5 + 4 = 39$

$t_1 = 5[t_0 - 1 \times 5^2] + 1 = 71$

↳ $5(39 - 25) + 1$

$t_2 = 5(t_1 - 2 \times 5^2) + 2 = 107$

$txt = 132456$

$d = 10, \quad m = 4$

$t_0 = 1324$

$t_1 = \underbrace{324 \times 10 + 5}$ $\Rightarrow$ $3240 + 5 = 3245$

$= (1324 - 1 \times 10^3) \times 10 + 5$

$= (324 \times 10) + 5$

$= 3245$

KMP -> Knuth - Morris - pratt.

↳ Constructing Lps (longest proper prefix Suffix Array).

I/p - ababc

O/p - lps[] = [0, 0, 1, 2, 0]

- Proper prefixes of abcd
  - ↳ '', a, ab, abc
- Suffixes of abcd
  - ↳ '', d, cd, bcd, abcd.

a
'' , a

common ones are '' length = 0

ab
'' , a
'' , b, ab
=> 0

aba
'' , a , ab
'' , a , ba, aba
=> 1

abab
'' , a, ab, aba
'' , b, ab, bab, abab
longest one is 2

ababc
'' , a, ab, aba, abab
'' , c, bc, abc, babc, ababc
=> 0.

=> [0, 0, 1, 2, 0]

'aaaa' [0,1,2,3]

'abcd' [0,0,0,0]

⇒ abac abad

[0, 0, 1, 0, 1, 2, 3, 0]

0 1 2 3 4

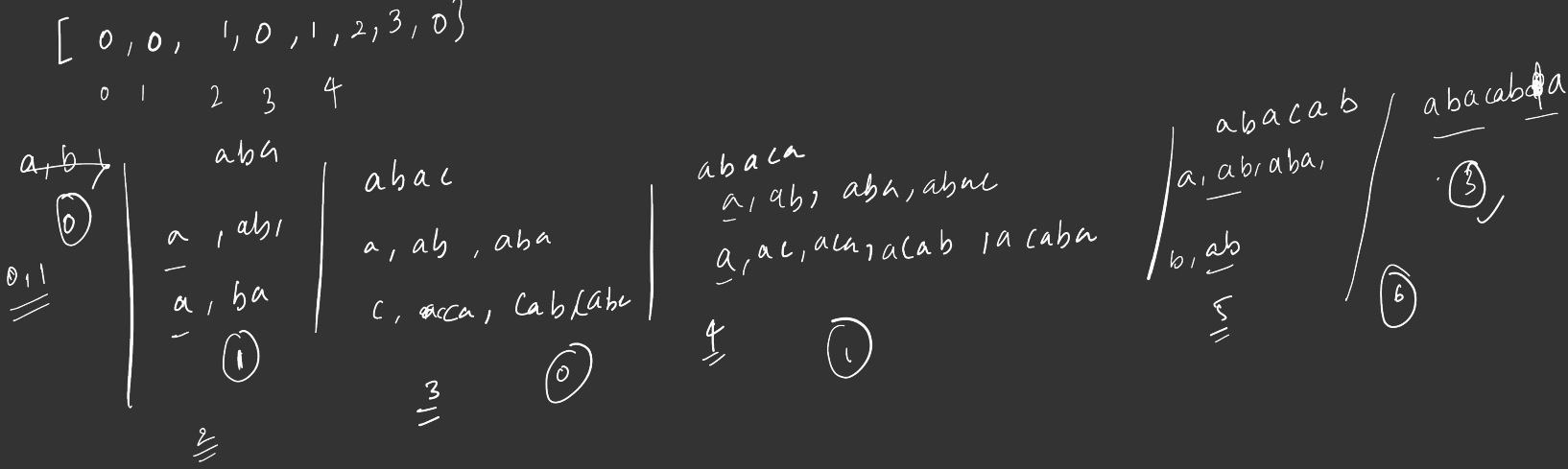


abb abb.