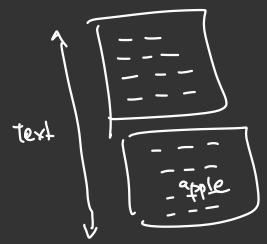
Pattern Matching in Strings

Text Doc



Pattern = "apple"

example – T = adbcd adb - - - - - - - N length

p = bcd

Load, "substring"

Background.

Brute Force Algo

→ Text = a b c x y z d e f a b c

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

len = N

→ Pattern = e f a

len = M

i=0 abc = efa O(1)

i=1 bcx = efa O(1)

i=2 cxy = efa

⋮

for(every i)

[compare the text [i ..... i+M-1] with pattern [0 ..... M-1]

Worst Case

$T = a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ b$

1 2 3 N-3 N-2 N-1

$P = \boxed{a\ a\ b}$

←3→

Ⓜ

Ⓟ

$i=0$ 0 1 2 aaa = aab , Ⓜ

$i=1$ 1 2 3 aaa = aab Ⓜ ←

$i=2$ aaa = aab

⋮

$N-M$ $\boxed{aab} = \boxed{aab}$ Ⓜ

$\boxed{N > M}$

Bigger

$(N-M)M = NM - M^2$

$= O(N \cdot M)$ Quadratic

$\boxed{N * M}$

Goal

Linear

$O(N+M)$

↳ Next class

Concepts :

- → Prefix & Suffix Strings
- → LPS of a given String.
- → LPS[] array , Pattern Matching
- → Problems.

Prefix Strings → substrig start with 0 idx

S = " a b a b "

a
ab
aba
abab

Suffix → ending at n-1 index

S = " a b a b "

b
ab
bab
abab

LPS → Length of Longest Prefix which is also a suffix string except the entire strig

Ex1 → s = a b c a b

Prefix	same len.	Suffix
a		b
ab		ab
abc		cab
abca		bcab
~~abcab~~		~~abcab~~

len = 2

Ex-2 $\quad S = \text{"a"}$

LPS(S) $\quad \cancel{a \longrightarrow a} \quad len = 0$

Ex-3 $\quad S = \text{"a a a a a"}$

a	a
a a	a a
a a a	a a a
→ $\boxed{a\,a\,a\,a}$	$\boxed{a\,a\,a\,a}$
~~a a a a b~~	

$len = 4$

$S_6 = S_0\, S_1\, S_2\, S_3\, S_4\, S_5$ len = N

Time to find LPS

S_0 ------- S_5 1 +

$S_0 S_1$ --- $S_4 S_5$ 2 +

$S_0 S_1 S_2$ --- $S_3 S_4 S_5$ 3 +

$S_0 S_1 S_2 S_3$ --- $S_2 S_3 S_4 S_5$ 4 +

$S_0 S_1 S_2 S_3 S_4$ --- $S_1 S_2 S_3 S_4 S_5$ 5 + .

prefix suffix

$= 1+2+3+ \ldots n-1$

$= \frac{(n-1)(n)}{2}$

$= \boxed{O(N^2)}$ to find out $\boxed{\text{LPS}}$

A B == A B 2 itrs

Given a string s of len N, Return LPS array.

LPS for every prefix string.

LPS[i] = LPS value of substring[0...i]

LPS of prefix string ending at i.

	0	1	2	3	4	5	6
s =	$\boxed{a}$	a	b	a	a	b	a
LPS =	0	1	0	1	2	3	4

→ TC of Building this array.

LPS[0] = "a" = 0

~~a~~ ~~a~~ LPS = 0

LPS[1] = "aa"

a a = 1

~~aa~~ ~~aa~~

LPS[2] = "aab" = 0

a --- b = 0

aa --- ab

~~aab~~ ~~aab~~

LPS[3] = "aaba"

a a = 1

aa ba ×

aab aba ×

LPS[i] = generating all prefix strings & comparing

$= O(N^2)$

LPS[] for every i

$= O(N) \times O(N^2)$

$= O(N^3)$

LPS [4] = "a a b a a"

a a

aa aa = 2

aab baa

aaba abaa

LPS [5] = "a a b a a b"

a b

aa ab

aab aab = 3

aaba baab

aabaa abaab

LPS [6] = 4

there exists an Algorithm which can do this work [in O(N) time.]

[code]

Original Problem

	0	1	2	3	4	5
T =	a	a	b	a	c	d

N = 6

P = a b a c M = 4

Pat len = 4

matches with prefix of len 4 sta

Join P + T

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

LPS[10] = 0, 0, 1, 0, 1, 1, 2, 3, 4, 0

LPS(i)

LPS[i] = N^2 for every i = N^3

But O(N) is possible

Interpret

ab

aba

LPS=8 (abac aaba c)

a c

abac → abac ⇒ 4

Obs if $LPS[i] == M$ → length of Pattern

↳ Pattern is present at text.

Another Example

Count the no of occ of given pattern in Text.

T : d a b c c a b c f

P : a b c

Cnt = 2

Algorithm: P + T

↗ Pattern as prefix in Text

Pattern

$\Rightarrow$ a b c d a b c c a b c f

LPS = [0, 0, 0, 0, 1, 2, 3, 0, 1, 2, $\boxed{3}$, 0] → Linear time

↑ longest match

cnt indexs

where LPS[i] == M

Effective length of P + T = (N + M)

↑ Text ↑ Pattern

Brute Force Code for Pattern Matching

```
s = P + T
LPS = [ ]
for ( every i=0 — s.length-1) {
        s = substring (0---i)
        LPS = getLPS (s);
}
```

$O(N \cdot N^2) = O(N^3)$

$\rightarrow O(N)$

$\leftarrow O(N^2)$

0 1 2 3 LPS

(- a b a c - - - - -) N

i=3

a c

ab a c

abc b a c

$O(N^2)$

```
get LPS ( S){
    ans = 0, n = S.length
    for( j= 0; j< n-1 ; j++){
        PS(j) = S (0 ... j)
        SS(j) = S (n-j-1 --- n-1)
        if ( PS == SS) & ans = j+1 };
}
```

N

Longest

Linear in length

a b a a b

j=0 j=1 j=2

ans = ~~0~~

a == b

a b == a b 2

a b a == a a b -

- - - - - -

a a a a

j=0 j=1

LPS

a == a 1

a a == a a 2

a a a == a a a 3

j=2

⇒ s1.equals(s2)

O(N)

$T =$ a a a a $N = 4$

$P =$ a a $M = 2$

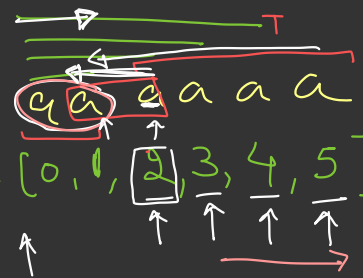
ans = 3

Improved Algo → to count occ

1. $S = P + T$ use special char.
2. Construct LPS[] ← Algo $O(\cancel{N^3})$ $O(N)$
3. iterate over LPS

```
if(LPS[i] == M) {
    cnt = cnt + 1
}
```

4. Return cnt



LPS = [0, 1, 2, 3, 4, 5]

cnt all indices where lps >= 2

ans = 4 − 1 = 3

Fix this issue

P + T ⇒ add some special char in between which

(should not be a input.)

a a $ a a a a

LPS = [0 1 0 1 2 2 2]

cnt - - - - 1+1+1 = 3

ab c a b d

0 0 0 1 2 0

0,1

Q) Given a binary string S, find no of cyclic rotations which are same as original string

S original = 1 0 1 0

r1 = 0 1 0 1

r2 = 1 0 1 0 + 1

r3 = 0 1 0 1

r4 = 1 0 1 0 + 1

cnt = 2

All rotations

T = S S

P' = S

S = a b c d

Ex →

→ r1 = b c d a

→ r2 = c d a b

→ r3 = d a b c

→ r4 = a b c d

Pattern

abcd

abcd

S' = S S

= a b c d a b c d

Claim: [contains every possible rotation]

Algo

1010 $ 1010 1010

n 1 n n

P $ T

S $ SS T

S = "1010"

Counting Algorithm

$\Rightarrow O(3N+1)$

$= O(N)$

LPS → $O(N^3)$

LPS → $O(N)$

LPS → Sliding window $O(N^2)$

Find pattern S in SS

a a a a a b

aab

N, N, N, N $= N \times N$

$= N^2$

Algo for creating LPS O(N)

Reference code for Assignment

```
fn ( str ) {

    LPS[0] = 0

    for (i=1 ; i<N ; i++) {

        x = LPS[i]

        while ( str[i] != str[x] ) {
            if (x==0) {
                x = -1
                break
            }
            x = Lps[x-1]
        }

        Lps[i] = x+1
    }

}
```

Complex.

[Analysis + Explaination]

Next class

Pattern matching

Regex is form of pattern matching

libraries
Java

[Rules]

(abc)

T → a a a a b

P → a * b

one or more occ of prev letter

a a b
a a a a b

any letter

P ⇒ [a-z] * @ gmail com

T

pn @ gmail com

x12 @ gmail com ✗

$\Rightarrow$ regex(P, T) [✓]