Strings Algorithms

Knuth - Morris - Pratt (KMP) Algorithm :-

Idea: "A B A B C A B A B D"

Pattern: "A B A B D"

instead of starting from $i = 1$ & $j = 0$
start from $i = 4$ & $j = 2$
Because first 2 element of pattern is same as next 2 elements

"A B A B C A B A B D"
already matched

"A B A B D"

For this we need to pre calculate Longest prefix Suffix match in pattern for each index.

pattern: "a a a c a a a a"

	0	1	2	3	4	5	6	7
LPS:	0	1	2	0	1	2	3	3

"aaa"

Dry Run :

a a a c a a a a
i
L = 1

a a a c a a a a
0 1 i
L = 2

a a a c a a a a
0 1 2 i
✗ L = LPS[2-1] = 1
✗ L = LPS[1-1] = 0
L = 0

a a a c a a a a
0 1 2 0 i
L = 1

a a a c a a a a
0 1 2 0 1 i
L = 2

a a a c a a a a
0 1 2 0 1 2 i
L = 3

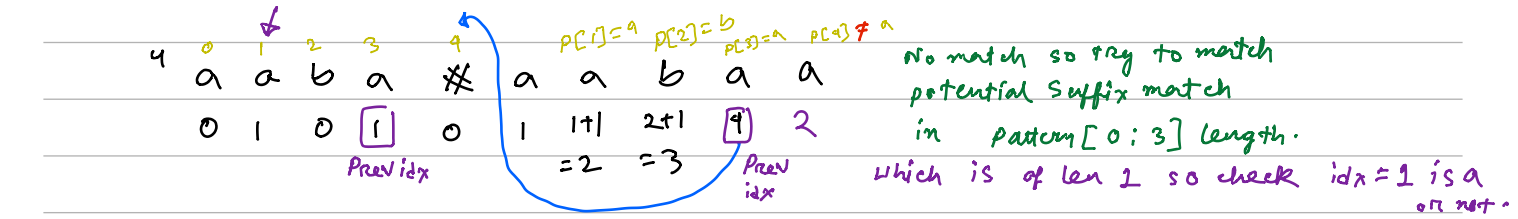
a a a c a a a a
0 1 2 0 1 2 3 i
✗ L = LPS[2] = 2
✓ L = 2+1 = 3

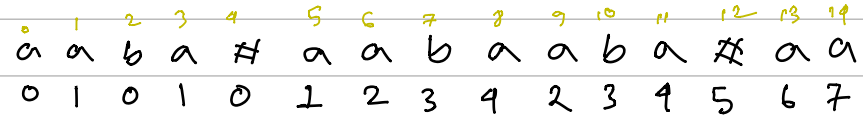
a a a c a a a □
3 match was already there but not c == □
So try if a a a c a a a □
max suffix = 2
so if □ == a
max suff = 3

```
i = 1 , length = 0
while (i < n)
{ if ( pattern[i] == pattern[length] ){
      length++ ;
      LPS [i] = length ; i++ }
  else if ( length != 0 ){
      length = LPS [length - 1] }
  else i++ ;
}
```

KMP Code:

```
while (i < N)
{  if ( text[i] == Pattern[i])
        i++; j++;
   if ( j == lenPattern){
        result.pushback (i-j);
        j = 0                    }
   else if ( text[i] ≠ pattern[j]) {
        if (j ≠ 0)  j = LPS[j-1];
        else  i++;  // j=0  first letter in pattern.  }
}
```





Shorter version :-

```
while (i < n)
    prev_idx = LPS[i-1];
    while (prev_idx > 0  &&  pattern[prev_idx] ≠ pattern[i])
        prev_idx = LPS[prev_idx - 1]   // May get an answer from 0 to prev_idx-1 substring.

    LPS[i] = prev_idx + (pattern[i] == pattern[prev_idx])
```

Trickery :-

#⭐, #Trick

pattern: "aaba" string: "gabaabaabab"

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	a	a	b	a	#	g	a	b	a	a	b	a	a	b	a	b
LPS:	0	1	0	1	0	0	1	0	1	2	3	4	2	3	4	0

actual idx = 11 − 2×4 = 3

actual index = 14 − 8 = 6

> For finding maximum length palindrome from the start:

a c b c a b e # e b a c b c a

Longest Prefix & Suffix

○ check whether a string is made by repeating one substring more than ones;

e.g abab ✓ aba ✗

→ if LPS[n−1] ≠ 0

&

LPS[n−1] % (n − LPS[n−1]) == 0

e.g abeabeabcabe

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

n − LPS[n−1]

LPS[n−1] % (n − LPS[n−1]) ≠ 0

LPS[n−1] = 9 → abcabeabc

n − LPS[n−1] = 3 → abe

	0	1	2	3	4	5	6	7	8	9	10	11
	a	b	c	a	b	c	a	b	e	a	b	e
	0	0	0	1	2	3	1	2	3	4	5	6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	a	b	a	b	a	c	a	b	a	b	a	c	a	b
	0	0	1	1	2	3	0	1	2	3	4	5	6	7

7 % (14 − 7)

6 % (13 − 6)

	0	1	2	3	4	5	6	7	8
	a	b	c	a	b	c	a	b	e
LPS	0	0	0	1	2	3	4	5	6

3 % (9 − 6) = 0

Longest common suffix & prefix is abcabe & abcabe

① Determine whether 2 strings are close or not.

operation 1 : swap any 2 existing character

op 2 : change all occurance of a character into another

return word1 & word2 are interchangable.

soln : ① each character must be present in one another.

② Must have same frequency count not necessarily of same character.

aabbbccccc && bbcccaaaaa

② Orderly Queue : Hard

s: "baaca" k = 2

return lexicographically smallest string possible.

you can take any character among any first K character & push it at back, any no. of times

soln :

$a_1 a_2 a_3 \boxed{X Y} a_4 a_5$ k = 2

$a_2 a_3 X Y a_4 a_5 a_1$

$a_3 X Y a_4 a_5 a_1 a_2$

$X \boxed{Y} a_4 a_5 a_1 a_2 a_3$

$X a_4 a_5 a_1 a_2 a_3 Y$

$a_4 a_5 a_1 a_2 a_3 Y X$

$a_5 a_1 a_2 a_3 Y X a_4$

$a_1 a_2 a_3 \boxed{Y X} a_4 a_5$

with enough swapping with k >= 2 can swap any 2 character.

so ans = sorted string

if k = 1 Brute force

```
string res = s;
for(int i=0; i<s.length(); i++)
{   string temp = s.substr(i) + s.substr(0, i)
    res = min(res, temp)   }

return res;
```

0 String with largest variance :

" a b a a a b b b b " find substring where variance is highest.

unique chars a , b .

a:	a	b	a	a	a	b	b	b	b
freq a	1	1	2	3	4	4	4	4	4
freq b	0	1	1	1	1	2	3	4	5
ans		0	1	2	3	2	1	0	~~+0~~

max(0, fa - fb)

b:	a	b	a	a	a	b	b	b	b
freq b	~~0~~ 0	1	1	~~+0~~ 0	~~0~~ 0	1	2	3	4
freq a	~~+0~~ 0	0	1	~~2~~ 0	~~1~~ 0	0	0	0	0
ans	~~-1~~		0	~~-0~~	~~-0~~	1	2	3	4

max(0, fb - fa)

b a a a b b b b b a a a b b b b .

1 1 1 0 1 2 3 4 0 1 2 3 3 3 3 .

0 1 2 1 1 1 1 1 1 0 0 0 1 2 3 .

0 3 2 1 0

```
for ( ele1 = 0 to 26)
    for (ele2 = 0 to 26 )
        (if ele1 exist & ele2 exist in string)
            {   for (ch : s)
                    if (ch == ele1) firstCount ++.
                    if (ch == ele2) secondCount ++;
                    if (secondCount > 0) ans = max(ans, firstC - secC)
                else if (seenInPast) ans = max(ans, FC-1)
                    if (firstCount < second) fC = 0, sC = 0,
                                                seenInPast = True
```

This code will not work for this : " a b b "

ans = 0

instead 1 need proper Kadane

OR seen in past variable

① Decode Strings :-

"ab22cd3" → ababababcdababababcdababababcd

K = 10

return char at idx = K

⇒ decoded size : 30

start checking from the end.

	a	b	2	2	c	d	3
size:						10	10 ← 30/3
K:						0	10 ← 10 % 30

K == 0 → d (ANS)

	a	b	2	2	c	d	3
		↓ ans			K=0 ↓ ANS	K≠0 ↑ -1	↑ /3
size:		1	2	4	8	9	10
K:		0	2	2	2	2	22

```
while ( j >= 0 )  j--;
    K = K % size;
    if (K == 0  && s[i] is alpha)  ans = s[i]
    else if ( s[i] is alpha)  size--;
    else if ( s[i] is digit)  size  /=  (s[i] - '0') ;
```

0 Minimum Time to Revert word to Initial State :-

a b a d c a b a d

↳

~~a b a d c a b a d~~ K=2

~~a b a d c~~ a b a d K=5

suffix Prefix match.

Actual Problem :- when are you getting a suffix-prefix match earlies while deleting K elements from the front.

			ans
	a b b a x Y a b b a		
✓	~~a b b a x Y~~ a b b a	K = 6	1
x	~~a b b a x Y~~ a b b a	K = 5	2
x	~~a b b a x Y~~ a b b a	K = 4	3
✓	~~a b b a x Y~~ a b b a	K = 3	2
x	~~a b b a x Y~~ a b b a	K = 7	2
x	~~a b b a x Y~~ a b ~~b a~~	K = 8	2
✓	~~a b b a x Y~~ a ~~b b~~ a	K = 9	1

another suffix prefix match.

```
int suffix_match = LPS[n-1]   // assuming the largest.
while ( (remaining % K) ≠ 0 ) && suffix match > 0
    suffix_match = LPS[suffix_match - 1]
    remaining = n - suffix_match;

return ceil( remaining / (double) K )
```

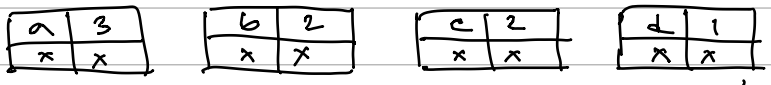
* otherwise floor would be calculated.

Huffman Coding :

String "a a a b c b d c"

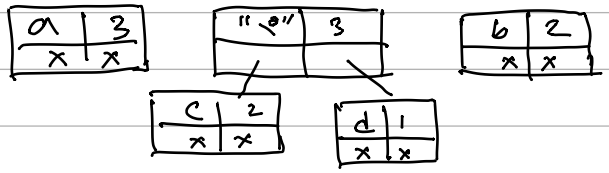
a : 3
b : 2
c : 2
d : 1

Step 1 : Build Tree

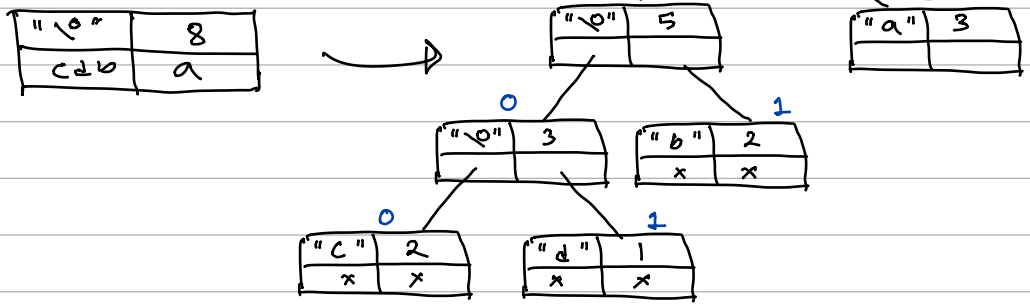
pq : 

Struct{ char int left right }

Pop smallest 2 & make a new combine & push it back.

pq : 

pq : 

pq : 

Step 2 : Build Hash map

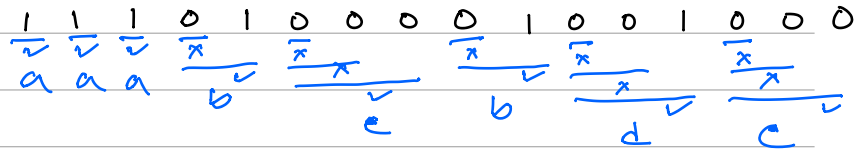
encode

char	binary
a	1
b	01
c	000
d	001

decode

1	a
01	b
000	c
001	d

"a a a b c b d c"
→ 1 1 1 0 1 0 0 0 0 1 0 0 1 0 0 0



* No character's encoding is prefix of another character.