(VI) Reverse word in a string

I/P hello.i.like.string
O/P string.like.i.hello

a) abc.def.gh ——(reverse word by word)——> cba.fed.hg ——(reverse string)——> gh.def.abc

```
string f(string s)
{
    int j=0;
    for(int i=0; i<s.length(); i++)
    {
        if(s[i]=='.')
        {   reverse(s.begin()+j, s.begin()+i);
            j=i+1;
        }
    }
    reverse(s.begin()+j, s.end());
    reverse(s.begin(), s.end());
    return s;
}
```

b) Without reversing char words by words
TC- O(|s|) SC- O(|s|)

```
string f(string s)
{
    vector<string> v;
    string st="";
    for(int i=0; i<s.length(); i++)
    {
```

```
if(s[i]=='.')
{
      v.push_back(s1);
      s1="";      // s1 को empty कर के फिर continue
      continue;
}

   s1+=s[i];
}

v.push_back(s1);   // आखरी word push
string ans="";
for(int i=v.size()-1; i>=0; i--)
{

      ans+=v[i];
      if(i!=0)
         ans+='.';
}

return ans;
}
```

→ Pattern Searching

txt (text) → "geeksforgeeks"
pat (pattern) → "eks
   o/p → 2 10 (eks present at index 2 & 10)

txt → "aaaaaa"
pat → "aaa"
   o/p → 0 1 2 (all valid index)

Uses

- when we search word in file (ctrl+F)
- google search is a pattern searching problem
- Regular expression search
- DNA matching
-

Overview

m → pattern length
n → text length
$1 \le m \le n$

① Naive $O((n-m+1)*m)$

Naive when all characters of pattern are distinct $O(n)$

No preprocessing

② Rabin Karp: $O((n-m+1)*m)$
better than naive on Average

Preprocess pattern

- uses idea of rolling hash to it optimize naive algo
- We compute hash function, if hf matching then we match individual character.
- in worst case hf matching every time. ∴ it might perform poor than naive

③ KMP $O(n)$ | preprocess pattern

④ Suffix Tree $O(m)$: preprocess txt then $O(n)$ time to search
↓ it is Data structure

⑤ Other Algos Z-Algorithm, Boyer Moore Algo $O(n)$ both
↓ simple than KMP

LPS : longest proper prefix which is also suffix

- suffix tree is build using idea of Trie.
- If txt is fixed then suffix tree search later

(i) Naive Pattern Searching : $O((n-m+1)*m)$

txt = "ABABABCD"
pat = "ABAB"
logic = two for loop में i के लिए pattern को check करो

```
void f(string txt, string pat)
{
    int n = t.length;
    int m = p.length;
    for(int i=0; i<=n-m; i++)
    {
        for(int j=0; j<m; j++)
            if( p[j] != t[i+j])
                break;
        if (j==m)
            cout<<i<<" ";
    }
}
```

- Improved Naive for Distinct character in Pattern

txt ABCEABEFABCD $O(n)$
pat ABCD o/p - 8

```
void f (txt, pat)
{   int n= txt.length(), m = pat.length();
    for(int i=0; i<=n-m; )
    {   int j;
        for(j=0; j<m; j++)
            if (pat[j] != txt[i+j])
                break;
        if(j==m)
            print (i+" ");
        if( j==0)
            i++;
        else
            i = i+j;
    }
}
```

→ Rabin Karp Algorithm

- Like Naive, in RK also we slide the pattern over text one by one.
- We use hashing
- We compute hash value of pattern (which is integer value) and hash value of every window of the text
- If these two integer values match then only we compare the pattern & text.
  individual characters of
- RK decreases no. of comparisons.

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

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

p: hash value of pattern
t = ——— current window of text

$p = (1+2+3) = 6$

Hash = sum of values

$i=0 \quad t = (1+2+4) = 7$
$i=1 \quad t = (2+4+1) = 7$
$i=2 \quad t = (4+1+2) = 7$
$i=3 \quad t = (1+2+3) = 6$ (Match)
$i=4 \quad t = (2+3+2) = 7$
$i=5 \quad t = (3+2+1) = 6$ (Spurious hit)
$i=6 \quad t = (2+1+2) = 5$
$i=7 \quad t = (1+2+3) = 6$ (Match)

★ Rolling Hash: we calculate next hash value by using previous hash value

$$t_{i+1} = t_i + txt[i+m] - txt[i]$$

m = length of pattern

"abc" 1 2 3

* Since "abc" has total 6 permutation with hash value same as 6, Using the above Hash function we will check and compare all these permutation if they exist in txt

∴ Improved Hash:

d → no. of distinct character in txt

$h("abc") = 1 \times d^2 + 2 \times d^1 + 3 \times d^0$

$= 1 \times 5^2 + 2 \times 5^1 + 3 \times 5^0 = 38$

$h("dab") = 4 \times 5^2 + 1 \times 5^1 + 2 \times 5^0 = 107$

Rolling hash for this:

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

(h = d^{m-1})

Eg: pat = "abc"

$p = 1 \times 5^2 + 2 \times 5 + 3 \times 1$

$= 38$

txt = "abdabcbabc"

$t_0 = 1 \times 5^2 + 2 \times 5^1 + 4 \times 5^0 = 39$

$t_1 = 5 \times (t_0 - 1 \times 5^2) + 1 = 71$

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

$t_3 = 5 \times (t_2 - 4 \times 5^2) + 3 = 38$

→ Worst Case of RK → $T((n-m+1) \times m)$

txt: "AAAAAA"

pat: "AAA"

then we calculate hash value & also do comparison for all window.

q → large prime no
if q small say (13) then window value will have only 0 to -12.

```
#define d 256
void f(string pat, string txt, int q)
{
    int n = txt.length(), m = pat.length();
    int h=1, p=0, t=0;
    for(int i=1; i<=m-1; i++)
        h = (h*d) % q
```

→ $(d^{m-1}) \% q == pow(d, m-1)$

```
    for(int i=0; i<m; i++)
    {
        p = (p*d + pat[i]) % q;
        t = (t*d + txt[i]) % q;
    }

    for(int i=0; i<=n-m; i++)
    {
        if(p==t)
        {
            bool flag = true;
            for(int j=0; j<m; j++)
                if(txt[i+j] != pat[j])
                {
                    flag = false;
                    break;
                }
            if(flag == true)
                print(i);
        }
        if(i<n-m)
        {
            t = (d*(t - txt[i]*h) + txt[i+m]) % q
            if(t<0)
                t = t+q;
        }
    }
}
```

check for match & spurious hit

computing t_{i+1} using t_i

KMP Algo

- We construct lps (longest proper prefix which is also suffix) for preprocessing

Proper prefix of "abc"	Suffix of "abc"
"", "a", "ab"	"", "c", "bc", "abc"

"abacabad" lps[] = {0, 0, 1, 0, 1, 2, 3, 0}	"aaaa" lps[] = {0, 1, 2, 3}
"abbabb" lps[] = {0, 0, 0, 1, 2, 3}	"abcd" lps[] = {0, 0, 0, 0}

(i) Naive $O(n^3)$

"ababacab"
(i = 3)

when i = 3

abab   if aba == bab   lps[i] = 3 ✗

-1 → abab   if ab == ab   lps[i] = 2 ✓

Similarly we calculate for every i

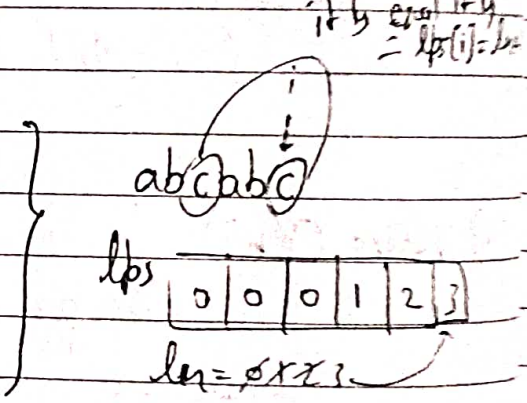
एक for loop string को check करने के लिए
इसके अंदर दो for loop, जो हर i length का lps(i) निकालेगा।

```
int longPropPreSuff(str, n)
{
    for(int len=n-1; len>0; len--)
    {   bool flag = true;
        for(int i=0; i<len; i++)
            if(str[i] != str[n-len+i])
                flag = false;
        if(flag == true)
            return len;
    }
    return 0;
}
```

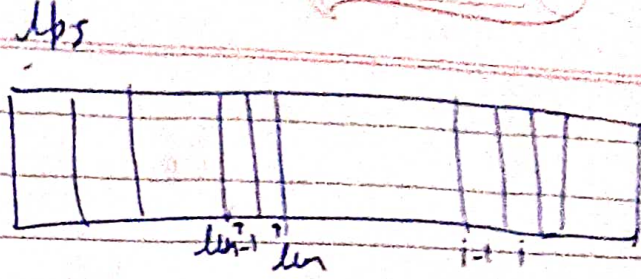
```
void fillLPS (str, lps[])
{
    for(int i=0; i< str.length; i++)
        lps[i] = longProPrefix(str, i+1);
}
```

(ii) O(n) → preprocessing

```
void fillLPS( string p, int m, vector<int> &lps)
{
    lps[0] = 0;
    int i=1, len=0;
    while (i<m)
    {
        if(p[i] == p[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            if(len==0)
            {
                lps[i]=0;
                i++;
            }
            else
                len = lps[len-1];
        }
    }
}
```



Case (I) If $len = lps[i-1]$ & $p[i] = p[len]$ then $lps[i] = len + 1$



Case (II) a) If $p[i]$ & $p[len]$ not same & $len == 0$ then $lps[i] = 0$

b) we recursively apply $len = lps[len-1]$ then compare $p[i]$ & $p[len]$.

lps: 0 0 1 0 0 1 2

↓

~~abc~~ ~~a~~ cecabce

```
vector<int> Search(string pat, string txt)
{
    int m = pat.length(), n = txt.length();
    vector<int> lps(m);
    fillLPS(pat, m, lps);
    vector<int> ans;
    int i=0, j=0;
    while (i<n)
    {
        if(txt[i]==pat[j])
        {
            i++; j++;
            if(j==m)
            {
                ans.push_back(i-j);
                j = lps[j-1];
            }
```

```
else
{
    if(j==0)
        itt;
    else
        j = abs(j-1);
}
}return ans;
}
```

```
else
{
    if(j==0)
        i++;
    else
        j = lps[j-1];
}
}return ans;
}
```

(VI) String Rotation वाला Question

(i) Given 2 string s1 & s2, the task is to check if s2 is a rotated version of the string s1.

eg: "geeksforgeeks"
"forgeeksgeeks"

geeksforgeeksgeeksforgeeks

```
string s= s1+s1;
if(s.find(s2)!=-1)
    return 1;
return 0;
```

```
if(s.find(s2)!= string::npos)
    return 1;
return 0;
```

npos → special constant

(ii) Check if string is rotated by exactly 2 places

→ the task is to find if the string 'b' can be obtained by rotating another string 'a' by exactly 2 places.

eg: "amazon"
"azonam" ⇒ amazonamazon i=2

"azonam"
"amazon" ⇒ azonamazonam i=n-2

```
string if((s1+s1).find(s2)!=-1)
                return
n=s1.length();
int ind=(s1+s1).find(s2);
if(ind==2 || ind==(n-2))
    return 1;
return 0;
```

(VII) Count Occurrence of Anagram

return the count of occurrences of anagram of the pat in the txt.

txt = "forxxorfxdofr" pat = "for"

O/P → 3

```
int f( string pat, string txt)
{   int n= txt.length(), m= pat.length();
    vector<int> v1(26,0), v2(26,0);
    for(int i=0 ; i<m; i++)
    {       v1[pat[i]-'a']++;
            v2[txt[i]-'a']++;
    }
    int ans=0;
    for(int i=0 ; i<=n-m; i++)
    {
            if(issame(v1,v2))
                ans++;
            v2[ txt[i+m]-'a'] ++;
            v2[ txt[i]-'a'] --;
    }
    return ans;
}
bool issame(vector<int> v1, vector<int> v2)
{
        for(int i=0; i<v1.size(); i++)
            if (v1[i] != v2[i])
                return 0;
        return 1;
}
```

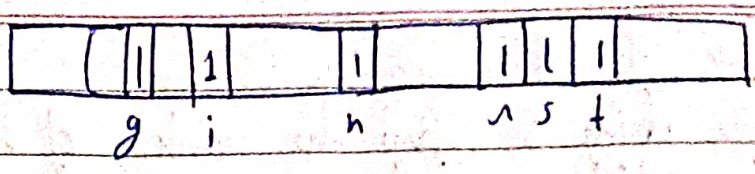
(XIII) Rank The Permutation

- given string S, find the rank of the string among all its permutation sorted lexographically.
- return 0 if character repeat in string

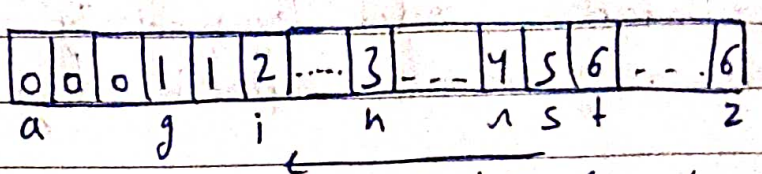
```
int rank(string s)
{
    long long int ans=1, rs=0;
    int n = s.length();
    for(int i=1; i<=n; i++)                    ] calculating n!
        ans = ans * i;
    ~~long long int~~
    vector<int> v(26,0);                       ] char array
    for(int i=0; i<n; i++)
    {   v[s[i]-'a']++;
        if(v[s[i]-'a']>1)
            return 0;
    }
    for(int i=1; i<26; i++)                    ] ~~prefix sum~~
        v[i] += v[i-1];

    for(int i=0; i<n-1; i++)                   ] iterate over s
    {                                            s[i] ~~th~~ ~~left~~ ~~right~~ element
        ans = ans/(n-i);                         ~~left~~ ~~right~~, we
        rs += v[s[i]-'a'-1] * ans;               calculate using char
        for(int j=s[i]-'a'; j<26; j++)           prefix sum array
            v[j]--;                              & multiply by
    }                                            ans.
    return (rs%1000003)+1;
}
```

logic : "string"



char array initialize



char array

s के right में s से छोटे characters 4 हैं, char array में लिखेंगे

g _____

n _____

i _____

^ _____

s से छोटे elements g, n, i, ^ हैं which is v[s[i]-'a'-1]

480 ← (4 × 5!) + ...

similarly others

480 string होते हैं s_____ से से

अब s fix हो गया

अब t फिर ^,i,n,g को fix करेंगे rank निकल जाएगा।

481 rank है s_____ से

after t fix st____

~~4×5!+t~~ t के right में 4 characters होते हैं A value char array में लिखेंगे

$4 \times 5! + 4 \times 4! + 3 \times 3! + 1 \times 2! + 1 \times 1!$

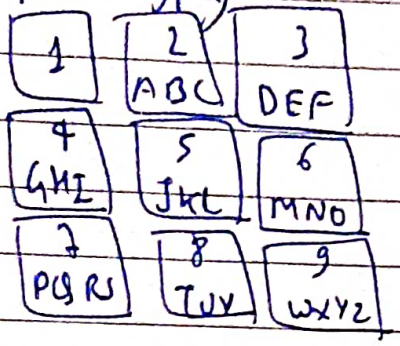
s_____ st____ str___ shi__ strin_

(IX) खुद का string कराके compare :

a) Given a string check if it is Panagram or not, A Panagram is a sentence containing every letter in the English Alphabet.

logic) s1 = "Bawds jog, flick quartz, vex nymph"
o/p → 1
make own string s2 = "abcde - - - - - xyz";
→ Now iterate over s2 & find(s2[i]).

b) Keypad Typing



s1 = "amazon"
o/p → 262966

make string s2 = "22233344455566677778889999";
Now iterate over s1 & find value corresponding to character.

- Isomorphic Strings:
2 strings s1 & s2 are IS if there is one to one mapping for every character

s1 = aab	s1 = aab	aba
s2 = xxy	s2 = xyz	pap
o/p → 1	0	

use visited array concept vector<bool> v(26,0);
& use map to iterate over s1