Strings

String Compression

Chars[]={‘a’ , ‘a’ , ‘b’ , ‘b’, ‘c’ , ‘c’ , ‘c’ } output={‘a’ , ‘2’ , ‘b’ , ‘2’, ‘c’ , ‘3’ } return length of modified array

Approach 1 :- for each char[i] find its count in chars and put the char[i] with count in new char array.

Approach 2-Use a ordered map qnd then store in array

Approach 3:- keep a char to store current_char as arr[i] , two var index and i , i to iterate and count each char , Index to replace that char and its count in the array(modifying array)

If arr[i]==current_char count++, i++ else means different then modify the array by putting current_char at arr[index] then do index++ and now its count at arr[index] again index++, curr_char=arr[i] cnt=0;

At the end when i>n return the index (length of the modified array)

Special case when count reaches more than 1 digit like A14B11 how to accommodate count , so use count as string by string_cnt=to_string(count) then put each char 1 and 4 1 and 1.

Check string is panagram (a to z letters)

Approach 1 put all char in a map cnt frequencies and at the end check if any char has frequency =0 or not

Approach 2 Rather than using map use an array of size 26 C :- O(2N) SC:- O(1)

If arr[str[i]-‘a’]==0 make it 1 this way add all chars to array and at the end check if any index has 0 or not

Also rather than traversing in the end count if you have done 26 unique changes from 0 to 1

Count and say

We have to tell ans for n=3 or n=4 and so on

For n=3 21 is the ans

We have to just do processing on what the previous call returned so that processing can return further

Approach 1 :-

Integer to ROMAN

Given these

Approach 1

Store the value corresponding to symbols in two vectors of length 13

To make 58 see the greatest symbol just less than or equal to 58 i.e L and see how many times this L is needed (as in case of XXX or III type of numbers a symbol is needed more than once)

Int num=58 greatest num with (defined symbol) just less than or = 58 is 50(from val arr)

Int times=58/50=1 hence one times L (from symbol arr) result “L” remainder 58%50=8

Num=8 greatest number with (defined symbol) just less than or = 8 is 5(from val arr)

Int times=8/5=1 hence one times V(from symbol arr) result “L”+”V” remainder 8%5=3

Num=3 greatest number with (defined symbol) just less than or = 3 is 1(from val arr)

Int times=3/1=3 hence three I result “LV”+”I” +”I”+”I” remainder 3%1=0

END

Result=”LVIII”

To find the greatest number (from val arr) just less than num simply start from 0th index in val and divide it by num , the val[i] if greater than num will give num/val[i]=0 hence we have to add them 0 times means nothing to add.

So the code will handle this case just write it for the case where we have to add the number for some number of times begging from the starting of val will automatically gives us greatest number just less than num.

Check if two String arrays are equivalent

Approach 1:Concat strings from arr1 and arr2 and check is the resultant string is same or not return true or false

Approach 2 : Compare character by character each string(word)inside word1 with that in word2.Let string arrays be of length m and n.

While(w1i < m && w2i <n){

if(words1[w1i][i]!=words2[w2i][j])

return false;

i++;

j++;

w1i represents i’th word of word1 , i is to iterate all chars of all words of word1 ,

when one word of word1 is finished then

reset i to 0 as now we are counting chars of new word inside word1.

Also do w1i++ as we are now on another word inside word1

Similarly for j and word2

Orderly Queue

Given a string and an integer K , we can put any of the first k chars at the end and go on repeating the same to return the lexicographically smallest string.

(at a time we are picking any one elements(any from 1 to K) and put it in the last

Approach 1 :- try all possible combos for all the first k elements by putting each at last and then see the smallest lexicographical string

Approach 2 :- The output will always be a sorted string except for K=1.

How always sorted string

If in a string we can swap any two elements , means the string can be sorted.

So the main logic is to check that keeping the constraint of shifting any char from first k chars , we can still swap any two elements.As if we can still swap any two elements with that constraint means we can get sorted string and means we can therefore get lexically smallest string

If we can prove that for k=2 we can swap any two elements hence we can sort the string and get smallest lexico

So means still we can swap any two elements with even the given constraint

But for K=1 we need bruteforce as only one place no two element swap constraint

Move each element to end and check smallest

For string=”ceabd”

```
For(int i=1;i<=n-1;i++){
  s+=s.substr(i)+s.substr(0,i)
}
```

Two Strings are Close or Not

Doing the following 2 operation tell if word1 can become word2

i)swap any two char

ii)change the occurrence of any char with some other char abccd -> cbaad

operations can be done any number of time

Approach 1 Intuition or conditions to identify

No. of characters in word1 and word2 should be same.

Characters in word1 and word2 should be same(means no char should be only in one word.Each char should be in word1 and word2 both)

Frequencies(regardless of which characters frequency) should be same in both words.

For cabbba a=2 , b=3 , c=1 and abbccc b=2, c=3 , a=1 frequencies are same that is 231 and 231 (regardless of char)

Regardless of char because we can transform one char into another so each char frequency in both words can be matched no issue.

If these conditions are true means we can transform word1 to word2.

This frequency array can help us store freq and also tell if the chars occurring in both words are same or not.So both conditions can be checked

To check if both words have same chars

To check if both words have same frequency sort freq1 and freq2

Good Word or not

Approach 1 Check all conditions as per saidTo check 3rd condition just check word.substr(1) for checking all small

Approach 2 Check the count of only capital letters

If cnt=0 means all small , if cnt==n means all capital if cnt=1 means third

Concatenated Words

Given an array of strings find which strings are concatenation of two or more of the strings present in array.

Approach 1 :- try finding all possible concatenations.We can do it by including the string or not including the string.At the end we will get all possible concats and now put it in set and check if the words of string are present in the set or not

It takes 2^n

Approach 2:- A concatenation is formed by prefix+suffix string.Put all strings in a set and then for all strings in array break each string into prefix and suffix(length by length).

Now Check if the prefix is there in set or not

if no then increase length of prefix as no concat possible with this prefix

if yes then there are two possibilities

that suffix will be present in set as it is.

that suffix can also be broken into prefix-suffix present in the set.

1st case 2nd case

Find index of first occurrence in a String

Approach1 :- Keep i to to traverse S1 and j for s2 , compare s1[i] with s2[j]

If same means this i can be useful(as it can be the first occurrence) so now we have to check if others char of s2 match with s1 string starting from i'th place.so j++

And to save i we check it by usinig s1[i+j]==s2[j] or not

If at any place they don't match so this i is not needed and restart j from 0

When the j has reached n-1 means we got the occurence

For slight opt run I from 0 to m-n as below see I should go till 4 only

Repeated Substring pattern

Approach :- try finding al the substrings of length l of substring should be div by length of orginal string and no of times to append = n/l

Also L at max can be n/2

Optimisation for a case , here if we begin from length 1 then we will append it 18 times (expensive)

So start from l=n/2

Is Subsequence

Approach :- Traverse s using i and t using j when i is out of bound means we found when j will get out of bound without i getting to out of bound means we couldn’t find it
If s[i]==t[j] increment I and j else increment only j

Follow up question we are given a lot of s we need to tell which all are the substrings of t. Checking for each s is substring of t is not efficient

Approach :Put each char along with an array(for the indexes at which the char occur in t) in a hash map

Then start checking the way we checked in above approach
We found s[i]==a we check if a is present in hashmap and if yes at which index ist {0,2}
So we will take 0 the first index , now we store the 0 in a variable called prev to keep a check that now for finding any char of S it should be present in map with index greater than prev as 0th index vala char(In t) we have found and looking for further indexes.

Again now s[i]==a again so we check a in hashmap and find it is at 0 and 2 from its array.We now have to choose 2 so we apply upperbound on array corresponding to a in hashmap upperbound(vector.begin(),vector.end() , prev)

If in any case we are not able to find an index greater than the previous variable in our hashmap means substring cannot be made.

Remove Duplicate letters from string

Remove duplicates in such a way that res is lexicographically smallest not sorted its smallest

Approach :- Make a result string , a bool istaken array for 26 chars and traverse the given string whichever char is in S add it to result and mark it true in istaken array.When again any char that is marked true in istaken comes we will skip it so we can avoid duplicates.

Now main thing store the last indexes of each char in S (at which index it occurs last) so that when putting elements in res we come to elem(e) smaller than the last elem in res , we will know if it is occurring at any index after the (e) .This we would know by the last index of its occurrence in S.

Now initially b and c were at right order so we put it into result and marked it true in istaken.But then came ‘a’ it was less than last element of res ‘c’ and we know c is coming at 4th index also so we removed ‘c’ from result and marked it false in istaken.Then again last element of res was ‘b’ so we checked if ‘b’ is coming after wards or not we can see yes it comes so we will remove it from res and mark false and now put ‘a’ in res and mark true.

Next b comes it is greater than a so now worries, then c comes greater than b no worries put it in res and return res.

And if say'b' was not present after the index of a we would then not remove it 'a' will come after it.

So we will only remove if res.back()> elem and last index of res.back()>index of elem

To check if some char exists after that or not we will make an array that will store the last indexes of each char in S (at which index it occurs last)

This is actually a question of Monotonic stack

{<->} is given below

Reverse words in a sentence

Approach 1:-Traverse the sentence keep reversing each word

Use two loops i marks the beginning of a word and j goes from i till a blank space means j-1 gives last letter of word

Reverse function accepts from where to start reversing , till where to reverse+1

i.e why we have passes s.begin()+j it will reverse till j-1

Approach 2:-Using tokenizer based upon spaces (by default it uses space to make tokens) stringstream ss(s);

Break to tokens and then reverse each and add to result with a space along with it

But at last we will have one extra space at the ending of result so we will return res.substr(0,res.length()-1)

Backspace string compare

means delete like typing then # means hit backspace

Approach1:Traverse S and adding chars in temp when encounter hash do TEMP.POP_BACK() do the same on T.

At the end compare temp with T if equal true else false. This uses an extra space O(m+n)

Approach2: Temp ch add krde aa chars fer agge jake # aan te ohi chars del krde aa so

baar baar add hi kyu krna ulta traverse krroo

Traverse both strings from back compare if s[i] and t[j] are equal or not

if equal i-- , j-- else return false

When you encounter hash we have to skip chars but first count how many hashes are present maybe two or more hashes are present simultaneously we will count them and then when we will come to normal char we will skip to other postions by doing -- as per the count of hashes,after this again we will compare s[i] and t[j] if same then again i-- , j-- and so on.

If any of i or j becomes less than 0 then we will end while loop

Count Number of Homogeneous Strings

Approach: Intution is that any homogenous string of length n will have substrings=sum of 1 to n and each these substrings will also be homogenous.

for aaaa no of homogenous substrings=10 i.e 1+2+3+4=10

for aaa no of homogenous substrings=6 i.e 1+2+3=6

for aa no of homogenous substrings=3 i.e 1+2=3

but to implement this we will not first find a longest poss homogenous substr of n

length , then cal 1 to n sum,this is not good approach

consider abbcccaa we have a length variable that is used to store the length of a homogenous string incrementally and a result variable to count the no of total homogenous substr

like on traversing we found ‘a’ so length=1 res=1

next we get b which is different char than ‘a’ so again length=1 res= 1 + 1

next we get b again same char so length=2 res=1+1+2 this 3 came from bb

next we get c which is different char than ‘b’ so again length=1 res= 1+1+2+1

next we get c again same char so length=2 res=1+1+2+1+2

next we get c again same char so length=3 res=1+1+2+1+2+3 this 6 came from ccc

next we get a which is different char than ‘c’ so again length=1 res=1+1+2+1+2+3 + 1

next we get a again same char so length=2 res=1+1+2+1+2+3+1+2 this 3 is from aa

string is over return res

Sort Vowels

Sorting is done based upon ASCII

Approach1: Make a separate string put all vowels occuring in S into it.Sort this string and now put these in places of vowels in S.

Nlogn(n)

Approach2:Avoid sorting we know that vowels are limited make a string vowel

Vowel=”AEIOUaeiou”

Add all the vowels and there count in a map.Now traverse the string S using i see where we get vowel “l” is not vowel remains as it is now i++ i is at E is vowel so now we go to vowel string and traverse it using j and j is at A see if A has cnt >0 , no move j forward to E it has cnt=1 write E ith index of S (i.e E in S replaced by E from vowel) and make count of E in map cnt-- and j++

Now in S do i++ we get ‘e’ it is a vowel so come to vowel now j is now at ‘I’ its count is also zero next to O its count is 1 so write O in ith index and make cnt-- and j++

Now i++ its t so i++ its c again i++ now its O means vowel so again go to vowel and

KMP Algorithm

To find if a string s appears in other string t , the bruteforce is to use two loops outer loop for every i in t start the substring and go comparing it with j in s till they both are same increment i and j.Where they are not same start j again from 0 and i from i+1

For every i we check that starting from i the s can be generated or not

TC:- O(n*m)

The main flaw is that where everytime t[i]!=s[j] j reintialised to 0 and I starts from i+1

Now t[i]!=s[j] so instead of reintialising j to 0 and I from i+1 , j was started from j went 2 places back from where it didn't match with i and i remains where it was

This is because j went back by length of a longest prefix from 0 to j-1 in S where prefix=suffix .See ABAB , the underlined AB(longest prefix which is equal to suffix) is already matched with AB in txt so why check it again

So we can say till index j-1 the longest prefix(equal to suffix) is of length 2.

Point is that at j where it didn't match with i , the non underlined AB(suffix) has matched with underlined AB in t that is why I reached to A.

Now we know in t if the underlined AB matched with non underlined AB in s means it would match with underlined AB in S also which is prefix of non underlined AB.

So that is why we don't need to make j to 0 coz we already have something usefull matching in t and s.

Now start matching from here

To task is it calculate LPS(longest length prefix=suffix) for every j LPS[j]