**############################################################**

**1- Check for Binary**

Given a non-empty sequence of characters str, return true if sequence is Binary, else return false

**Example 1:**

**Input:**

str = 101

**Output:**

1

**Explanation:**

Since string contains only 0 and 1, output is 1.

**Example 2:**

**Input:**

str = 75

**Output:**

0

**Explanation:**

Since string contains digits other than 0 and 1, output is 0.

**Your Task:**  
Complete the function **isBinary()**which takes an string str as input parameter and returns 1 if str is binary and returns 0 otherwise.

**Expected Time Complexity:** O(|s|)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 <=T<= 50  
1 <=Length of str<= 10000

bool isBinary(string str)

{ int i=0;

while(i<str.length())

{ if(!(str[i]=='1' || str[i]=='0'))

return false;

i++;

}

return true;

}

**############################################################**

**2- Convert to Roman No**

Given an integer n, your task is to complete the function **convertToRoman** which prints the corresponding roman number of n. Various symbols and their values are given below.

I 1  
V 5  
X 10  
L 50  
C 100  
D 500  
M 1000

**Example 1:**

**Input:**

n = 5

**Output:** V

**Example 2:**

**Input:**

n = 3

**Output:** III

**Your Task:**  
Complete the function **convertToRoman()**which takes an integer N as input parameter and returns the equivalent roman.

**Expected Time Complexity:** O(log10N)  
**Expected Auxiliary Space:** O(log10N \* 10)

**Constraints:**  
1<=**n**<=3999

string convertToRoman(int n)

{

int no[]={1000,900,500,400,100,90,50,40,10,9,5,4,1};

string rom[]={"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};

string str="";

int i=0;

while(n>0)

{ int d=n/no[i];

n=n%no[i];

while(d)

{ str.append(rom[i]);

d--;

}

i++;

}

return str;

}

**############################################################**

**3- Maximum Occuring Character**

Given a string str. The task is to find the maximum occurring character in the string str. If more than one character occurs the maximum number of time then print the lexicographically smaller character.

**Example 1:**

**Input:**

str = testsample

**Output:** e

**Explanation:** e is the character which

is having the highest frequency.

**Example 2:**

**Input:**

str = output

**Output:** t

**Explanation:**  t and u are the characters

with the same frequency, but t is

lexicographically smaller.

**Your Task:**  
The task is to complete the function **getMaxOccuringChar()** which returns the character which is most occurring.

**Expected Time Complexity:** O(N).  
**Expected Auxiliary Space:** O(Number of distinct characters).  
**Note:**N = |s|

**Constraints:**  
1 ≤ |s| ≤ 100

static bool cmp(pair<char,int> &a, pair<char,int> &b)

{ return a.second>b.second;

}

static bool cmpi(pair<char,int> &a, pair<char,int> &b)

{ return a.first<b.first;

}

char getMaxOccuringChar(string str)

{

unordered\_map<char,int> mp;

vector<pair<char,int>> v;

int i=0;

while(i<str.length())

{

mp[str[i]]++;

i++;

}

for(auto i:mp)

v.push\_back(i);

sort(v.begin(),v.end(),cmp);

int f=0,count=0;

for(auto i:v)

{ if(f>i.second)

break;

f=i.second;

count++;

}

v.resize(count);

sort(v.begin(),v.end(),cmpi);

return v[0].first;

}

**############################################################**

**4- Missing Characters in Panagram**

You are given a string s. You need to find the missing characters in the string to make a panagram.  
**Note:**Th**e output characters will be lowercase** and **lexicographically**sorted.

**Example 1:**

**Input:**

s = Abcdefghijklmnopqrstuvwxy

**Output:** z

**Example 2:**

**Input:**

s = Abc

**Output:** defghijklmnopqrstuvwxyz

**Your Task:**

You only need to complete the function **misssingPanagram()** that takes s as parameter and returns -1 if the string is a panagram, else it returns a string that consists missing characters.

**Expected Time Complexity:**O(|S|).  
**Expected Auxiliary Space:**O(1).

**Constraints:**  
1 <= |s| <= 10000

string missingPanagram(string strin)

{

bool cp[26]={false};

bool sm[26]={false};

int i=0;

while(i<strin.length())

{ if(isupper(strin[i]))

cp[strin[i]-'A']=true;

else if(islower(strin[i]))

sm[strin[i]-'a']=true;

i++;

}

/\*

for(int j=0;j<26;j++)

cout<<cp[j]<<" ";

cout<<endl;

for(int j=0;j<26;j++)

cout<<sm[j]<<" ";

cout<<endl;

\*/

bool f=true;

for(int j=0;j<26;j++)

{ if(cp[j]==true || sm[j]==true)

continue;

else

{

f=false;

break;

}

}

if(f==true)

return "-1";

string str="";

i=0;

while(i<26)

{

if(cp[i]==false && sm[i]==false)

{

if(i==0)

str.push\_back('a');

else

str.push\_back('a'+i);

}

i++;

}

return str;

}

**############################################################**

**5- Implement strstr**

Your task is to implement the function **strstr**. The function takes two strings as arguments **(s,x)** and  locates the occurrence of the string**x** in the string **s**. The function returns and integer denoting the **first occurrence**of the string **x** in **s** (0 based indexing).

**Note:**You are not allowed to use inbuilt function.

**Example 1:**

**Input:**

s = GeeksForGeeks, x = Fr

**Output:** -1

**Explanation:** Fr is not present in the

string GeeksForGeeks as substring.

**Example 2:**

**Input:**

s = GeeksForGeeks, x = For

**Output:** 5

**Explanation:** For is present as substring

in GeeksForGeeks from index 5 (0 based

indexing).

**Your Task:**  
You don't have to take any input. Just complete the **strstr()**function which takes two strings **str, target** as an input parameter. The function returns -1 if no match if found else it returns an integer denoting the first occurrence of the x in the string s.

**Expected Time Complexity:** O(|s|\*|x|)  
**Expected Auxiliary Space:** O(1)

**Note**: Try to solve the question in constant space complexity.

**Constraints:**  
1 <= |s|,|x| <= 100

int strstr(string s, string x)

{

int j,n=s.length(),m=x.length();

for(int i=0;i<=n-m;i++)

{

for(j=0;j<m;j++)

if(s[i+j]!=x[j])

break;

if(j==m)

return i;

}

return -1;

}

**############################################################**

**6- Remainder with 7**

Given a number as string(n) , find the remainder of the number whe it is divided by 7  
  
**Example 1:**

**Input:**

5

**Output:**

5

**Example 2:**

**Input:**

8

**Output:**

1

**Your Task:**

You only need to complete the function **remainderwith7()** that takes string n as parameter and returns an integer which denotes the remainder of the number when its divided by 7

**Constraints:**  
1<=length of n <=105

int remainderWith7(string str)

{ int n=str.length();

int i=0;

int no=0;

while(i<n)

{

no=(no\*10)+(int)(str[i]-'0');

no=no%7;

i++;

}

return no;

}

**############################################################**

**7- Check if strings are rotations of each other or not**

Given two strings s1 and s2. The task is to check if s2 is a rotated version of the string s1. The characters in the strings are in lowercase.

**Example 1:**

**Input:**

geeksforgeeks

forgeeksgeeks

**Output:**

1

**Explanation:** s1 is geeksforgeeks, s2 is

forgeeksgeeks. Clearly, s2 is a rotated

version of s1 as s2 can be obtained by

left-rotating s1 by 5 units.

**Example 2:**

**Input:**

mightandmagic

andmagicmigth

**Output:**

0

**Explanation:** Here with any amount of

rotation s2 can't be obtained by s1.

**Your Task:**  
The task is to complete the function **areRotations()**which checks if the two strings are rotations of each other. The function returns true if string 1 can be obtained by rotating string 2, else it returns false.

**Expected Time Complexity:**O(N).  
**Expected Space Complexity:** O(N).  
**Note:** N = |s1|.

**Constraints:**  
1 <= |s1|, |s2| <= 107

bool areRotations(string s1,string s2)

{

if(s1.length()!=s2.length())

return false;

s1.append(s1);

int n=s1.length(),m=s2.length(),j;

for(int i=0;i<n;i++)

{ for(j=0;j<m;j++)

if(s1[i+j]!=s2[j])

break;

if(j==m)

return true;

}

return false;

}

**############################################################**

**8- Isomorphic Strings**

Given two strings '**str1**' and '**str2**', check if these two strings are isomorphic to each other.  
Two strings str1 and str2 are called isomorphic if there is a one to one mapping possible for every character of str1 to every character of str2 while **preserving the order**.  
Note: All occurrences of every character in ‘str1’ should map to the same character in ‘str2’

**Example 1:**

**Input:**

str1 = aab

str2 = xxy

**Output:** 1

**Explanation:** There are two different

charactersin aab and xxy, i.e a and b

with frequency 2and 1 respectively.

**Example 2:**

**Input:**

str1 = aab

str2 = xyz

**Output:**

**Explanation:** There are two different

charactersin aab but there are three

different charactersin xyz. So there

won't be one to one mapping between

str1 and str2.

**Your Task:**  
You don't need to read input or print anything.Your task is to complete the function **areIsomorphic**() which takes the string **str1** and string **str2** as input parameter and  check if two strings are isomorphic. The function returns **true**if strings are isomorphic else it returns **false**.

**Expected Time Complexity:**O(|str1|+|str2|).  
**Expected Auxiliary Space:**O(Number of different characters).  
**Note:** |s| represents the length of string s.

**Constraints:**  
1 <= |str1|, |str2| <= 103

bool areIsomorphic(string str1, string str2)

{ if(str1.length()!=str2.length())

return false;

int n=str1.length();

/\*

map<char,char> mp;

set<char> s;

for(int i=0;i<n;i++)

{

if(mp.count(str1[i])==0 && s.count(str2[i])==0)

{

mp[str1[i]]=str2[i];

s.insert(str2[i]);

}

else if(mp.count(str1[i])==0 && s.count(str2[i])!=0)

return false;

else if(mp[str1[i]]!=str2[i])

return false;

}

return true;

Execution Time:0.22

\*/

bool arr[256]={false};

int rem[256]={-1};

memset(rem,-1,sizeof(rem));

for(int i=0;i<n;i++)

{

if(rem[str1[i]]==-1)

{

if(arr[str2[i]]==true)

return false;

arr[str2[i]]=true;

rem[str1[i]]=str2[i];

}

else if(rem[str1[i]]!=str2[i])

return false;

}

return true;

//Execution Time:0.04

}

**############################################################**

**9- Anagram**

Given two strings **a**and **b** consisting of lowercase characters. The task is to check whether two given strings are an anagram of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different. For example, “act” and “tac” are an anagram of each other.

**Example 1:**

**Input:**

a = geeksforgeeks, b = forgeeksgeeks

**Output:** YES

**Explanation:** Both the string have same

characters with same frequency. So,

both are anagrams.

**Example 2:**

**Input:**

a = allergy, b = allergic

**Output:** NO

**Explanation:**Characters in both the strings

are not same, so they are not anagrams.

**Your Task:**  
You don't need to read input or print anything.Your task is to complete the function **isAnagram()** which takes the string **a** and string **b** as input parameter and check if the two strings are an anagram of each other. The function returns true if the strings are anagram else it returns false.

**Expected Time Complexity:**O(|a|+|b|).  
**Expected Auxiliary Space:** O(Number of distinct characters).

**Note:** |s| represents the length of string s.

**Constraints:**  
1 ≤ |a|,|b| ≤ 105

bool isAnagram(string a, string b)

{

if(a.length()!=b.length())

return false;

int n=a.length();

int arr[26]={0};

for(int i=0;i<n;i++)

arr[a[i]-'a']++;

for(int i=0;i<n;i++)

arr[b[i]-'a']--;

for(int i=0;i<26;i++)

if(arr[i]!=0)

return false;

return true;

}

**############################################################**

**10- Validate an IP Address**

Write a program to Validate an IPv4 Address. According to Wikipedia, [IPv4 addresses](http://en.wikipedia.org/wiki/IP_address)are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots, e.g., 172.16.254.1 . The generalized form of an IPv4 address is**(0-255).(0-255).(0-255).(0-255).**Here we are considering numbers only from 0 to 255 and any additional leading zeroes will be considered invalid.

Your task is  to complete the function**isValid**which returns 1 if the ip address is valid else returns 0. The function takes a string **s**as its only argument .

**Example 1:**

**Input:**

ip = 222.111.111.111

**Output:** 1

**Example 2:**

**Input:**

ip = 5555..555

**Output:** 0

**Explanation:** 5555..555 is not a valid

ip address, as the middle two portions

are missing.

**Your Task:**  
Complete the function **isValid()** which takes the string **s** as an input parameter and returns 1 if this is a valid ip address otherwise returns 0.

**Expected Time Complexity:** O(N), N = length of string.  
**Expected Auxiliary Space:** O(1)  
  
**Constraints:**  
1<=length of string <=50  
  
**Note:**The **Input/Ouput** format and **Example** given are used for system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from stdin/console. The task is to complete the function specified, and not to write the full code.

int isValid(string ip) {

// code here

int len = ip.length();

int count = 0 ;

for(int i=0;i<len;i++)

if(ip[i]=='.')

count++;

if(count!=3)

return false;

string s="";

bool flag=true;

bool zero=false;

for(int i=0;i<len;i++)

{

if(ip[i]=='.')

{

if(flag)

return false;

if(zero)

zero=false;

if(flag==false)

flag = true;

stringstream g(s);

int temp ;

g>>temp;

if(temp>255)

return false;

s="";

}

else

{

if(isdigit(ip[i])==false)

return false;

if(zero)

return false;

if(flag)

{ if(ip[i]=='0')

zero = true;

flag=false;

}

s += ip[i];

}

}

stringstream g(s);

int temp ;

g>>temp;

return temp<256;

}

**############################################################**

**11- Reverse words in a given string**

Given a String S, reverse the string without reversing its individual words. Words are separated by dots.

**Example 1:**

**Input:**

S = i.like.this.program.very.much

**Output:** much.very.program.this.like.i

**Explanation:** After reversing the whole

string(not individual words), the input

string becomes

much.very.program.this.like.i

**Example 2:**

**Input:**

S = pqr.mno

**Output:** mno.pqr

**Explanation:** After reversing the whole

string , the input string becomes

mno.pqr

**Your Task:**  
You dont need to read input or print anything. Complete the function**reverseWords()** which takes string S as input parameter and returns a string containing the words in reversed order. Each word in the returning string should also be separated by '.'

**Expected Time Complexity:**O(|S|)  
**Expected Auxiliary Space:**O(|S|)

**Constraints:**  
1 <= |S| <= 2000

void rever(string &str, int l, int r)

{ while(l<r)

{ swap(str[l],str[r]);

l++;

r--;

}

}

string reverseWords(string s)

{

// code here

int i=0,count,n=s.length();

vector<int> v;

for(int i=0;i<n;i++)

if(s[i]=='.')

v.push\_back(i);

v.push\_back(n);

int l=0;

for(int i=0;i<v.size();i++)

{ rever(s,l,v[i]-1);

l=v[i]+1;

}

rever(s,0,n-1);

return s;

}

**############################################################**

**12- Longest Substring Without Repeating Characters**

Given a string S, find the length of its longest substring that does not have any repeating characters.

**Example 1:**

**Input:**

S = geeksforgeeks

**Output:** 7

**Explanation:** The longest substring

without repeated characters is "ksforge".

**Example 2:**

**Input:**

S = abbcdb

**Output:** 3

**Explanation:** The longest substring is

"bcd". Here "abcd" is not a substring

of the given string.

**Your Task:**  
Complete **SubsequenceLength**function that takes string s as input and returns the length of the longest substring that does not have any repeating characters.

**Expected Time Complexity:** O(N)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
0<= N <= 10^5  
here, N = S.length

int SubsequenceLength (string s)

{ int n = s.length();

int j = 0, ans = 0;

vector<int> v(256,-1);

for (int i = 0; i < n; i++)

{ cout<<i<<" "<<j<<" "<<ans<<" "<<v[s[i]]<<endl;

j = max(j, v[s[i]]+1);

ans = max (ans, i - j + 1);

v[s[i]] = i;

cout<<i<<" "<<j<<" "<<ans<<" "<<v[s[i]]<<endl<<endl;

}

return ans;

}

**############################################################**

**13- Multiply two strings**

Given two numbers as stings s1 and s2. Calculate their Product.

**Example 1:**

**Input:**

**s1 =** "33"

**s2 =** "2"

**Output:**

66

**Example 2:**

**Input:**

**s1 =** "11"

**s2 =** "23"

**Output:**

253

**Your Task:**

You don't need to read input or print anything. Your task is to complete the function **multiplyStrings()** which takes two strings s1 and s2 as input and returns their product as a string.

**Expected Time Complexity:**O(n1\* n2)  
**Expected Auxiliary Space:**O(n1+ n2) ; where n1 and n2 are sizes of strings s1 and s2 respectively.

**Constraints:**  
1 <= length of s1 and s2 <= 103

string multiplyStrings(string s1, string s2)

{ if(s1=="0"||s2=="0")

return to\_string(0);

int sign=1;

if(s1[0]=='-')

{ sign\*=-1;

s1.erase(0,1);

}

if(s2[0]=='-')

{ sign\*=-1;

s2.erase(0,1);

}

int k = s1.length() + s2.length();

int a[k] = {0};

int digit;

for (int i = s1.length() - 1; i >= 0; i--)

{ for (int j = s2.length() - 1; j >= 0; j--)

{ digit = (s1[i] - '0') \* (s2[j] - '0');

a[j + i + 1] += digit;

a[i + j] += a[i + j + 1] / 10;

a[i + j + 1] = a[j + i + 1] % 10;

//cout<<a[i+j]<<" "<<a[i+j+1]<<endl;

}

}

string s = "";

if (sign == -1)

s = "-";

int i=0;

while(a[i]==0)

i++;

for (int j = i; j < k; j++)

s.append(to\_string(a[j]));

return s;

}

**############################################################**

**14- Excel Sheet | Part - 1**

Given a positive integer N, return its corresponding column title as it would appear in an Excel sheet.  
For N =1 we have column A, for 27 we have AA and so on.

**Note:**The alphabets are all in uppercase.

**Example 1:**

**Input:**

N = 51

**Output:** AY

**Your Task:**  
Complete the function **ExcelColumn()** which takes N as input and returns output string.

**Expected Time Complexity:** O(Log(N))  
**Expected Auxiliary Space:** O(Log(N))

**Constraints:**  
1 ≤ N ≤ 107

string ExcelColumn(int n)

{ int r;

string s = "";

while (n > 0)

{

r = n % 26;

n /= 26;

if (r==0)

{ s+='Z';

n--;

}

else

s+=char(r+64);

}

reverse (s.begin(), s.end());

return s;

}