1. **Reverse words in a given string**

Given a String of length **S**, reverse the whole string without reversing the individual words in it. Words are separated by **dots**.

**Input:**  
The first line contains **T** denoting the number of testcases. T testcases follow. Each case contains a string S containing characters.

**Output:**  
For each test case, in a new line, output a single line containing the reversed String.

**Constraints:**  
1 <= T <= 100  
1 <= |S| <= 2000

**Example:**  
**Input:**  
2  
i.like.this.program.very.much  
pqr.mno

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

**Code:**

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

string s;

cin>>s;

vector<string> v;

string c="";

for(int i=0;i<s.length();i++)

{

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

{

v.push\_back(c);

c="";

}

else

c+=s[i];

// cout<<c<<"c"<<endl;

}

v.push\_back(c);

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

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

{

if(i!=v.size()-1)

cout<<v[i]<<".";

else

cout<<v[i];

}

v.clear();

cout<<endl;

}

return 0;

}

2. **Permutations of a given string**

Given a string S. The task is to print all permutations of a given string.

**Input:**  
The first line of input contains an integer **T,** denoting the number of test cases. Each test case contains a single string **S**in capital letter.

**Output:**  
For each test case, print all permutations of a given string **S**with single space and all permutations should be in lexicographically increasing order.

**Constraints:**  
1 ≤ T ≤ 10  
1 ≤ size of string ≤ 5

**Example:**  
**Input:**  
2  
ABC  
ABSG

**Output:**  
ABC ACB BAC BCA CAB CBA   
ABGS ABSG AGBS AGSB ASBG ASGB BAGS BASG BGAS BGSA BSAG BSGA GABS GASB GBAS GBSA GSAB GSBA SABG SAGB SBAG SBGA SGAB SGBA

**Code:**

// using namespace std;

// int main()

// {

// int t;

// cin>>t;

// while(t--)

// {

// string str;

// cin>>str;

// sort(str.begin(),str.end());

// do{

// cout<<str<<" ";

// }while(next\_permutation(str.begin(),str.end()));

// cout<<endl;

// }

// return 0;

// }

//method-2- backtracking

#include<bits/stdc++.h>

using namespace std;

void permut(string str,int i,int n,vector<string> &v)

{

//base case

if(i==n-1)

{

v.push\_back(str);

return;

}

//rec case

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

{

swap(str[i],str[j]);

permut(str,i+1,n,v);

swap(str[i],str[j]);

}

return;

}

int main()

{

int t;

cin>>t;

while(t--)

{

string str;

cin>>str;

vector<string> v;

permut(str,0,str.length(),v);

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

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

cout<<v[i]<<" ";

cout<<endl;

}

return 0;

}

3. **Longest Palindrome in a String**

Given a string S, find the longest palindromic substring in S.**Substring of string S:** S[ i . . . . j ] where 0 ≤ i ≤ j < len(S)**. Palindrome string:** A string which reads the same backwards. More formally, S is palindrome if reverse(S) = S.**Incase of conflict**, return the substring which occurs first ( with the least starting index ).

**NOTE:** Required Time Complexity **O(n2).**

**Input:**  
The first line of input consists number of the testcases. The following **T** lines consist of a string each.

**Output:**  
In each separate line print the longest palindrome of the string given in the respective test case.

**Constraints:**  
1 ≤ T ≤ 100  
1 ≤ Str Length ≤ 104

**Example:  
Input:**  
1  
aaaabbaa

**Output:**  
aabbaa

**Code:**

using namespace std;

string palin(string str)

{

int ans=0;

string final="";

int n=str.length();

for(int l=1;l<=n;l++)

{

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

{

string s1=str.substr(i,l);

//cout<<s1<<"#"<<endl;

string s1r=s1;

reverse(s1r.begin(),s1r.end());

if(s1==s1r)

{

if(ans<s1.length())

{

ans=l;

final=s1;

// cout<<s1<<"#"<<endl;

// cout<<ans<<"-ans"<<endl;

}

}

}

}

return final;

}

int main()

{

int t;

cin>>t;

while(t--)

{

string str;

cin>>str;

cout<<palin(str)<<endl;

}

return 0;

}

4. **Recursively remove all adjacent duplicates**

Given a string **s**, recursively remove adjacent duplicate characters from the string **s**. The output string should not have any adjacent duplicates.

**Input:**  
The first line of input contains an integer **T,** denoting the no of test cases. Then T test cases follow. Each test case contains a string **str**.  
  
**Output:**  
For each test case, print a new line containing the resulting string.  
  
**Constraints:**  
1<=T<=100  
1<=Length of string<=50  
  
**Example:**  
**Input:**  
2  
geeksforgeek  
acaaabbbacdddd

**Output:**  
gksforgk  
acac

Code:

using namespace std;

string fn(string s)

{

if(s.empty())

return "";

int i,j,k;

int n=s.length();

stack<char>st;

int flag=0;

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

st.push(s[i]);

s="";

char c=st.top();

flag=0;

st.pop();

int key=0;

while(!st.empty())

{

//cout<<st.top()<<" ";

if(st.top()==c)

{

flag=1;

key=1;

}

else if(st.top()!=c )

{

if(flag==0)

s+=c;

c=st.top();

flag=0;

}

st.pop();

}

if(flag==0)

s+=c;

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

if(key)

s=fn(s);

return s;

}

int main()

{

int t,n,i,j,k;

cin>>t;

while(t--)

{

string s;

cin>>s;

s=fn(s);

cout<<s<<endl;

}

return 0;

}

5 . **Check if string is rotated by two places**.

Given two strings **a** and **b**. The task is to find if a string '**a**' can be **obtained** by **rotating** another string '**b**' by**2 places**.

**Input:**  
The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. In the next two line are two string a and b.

**Output:**  
For each test case in a new line print 1 if the string 'a' can be obtained by rotating string 'b' by two places else print 0.

**Constraints:**  
1 <= T <= 50  
1 <= length of a, b < 100

**Example:  
Input:**  
2  
amazon  
azonam  
geeksforgeeks  
geeksgeeksfor

**Output:**  
1  
0

Code:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

string str,str1;

cin>>str;

cin>>str1;

int n=str.length();

string a=str.substr(0,n-2);

string b=str.substr(0,2);

string a1=str.substr(n-2,2);

string b1=str.substr(2,n-2);

// cout<<a<<"-a"<<endl;

// cout<<a1<<"-a1"<<endl;

// cout<<b<<"-b"<<endl;

// cout<<b1<<"-b1"<<endl;

if(( str1==(a1+a) )|| (str1==(b1+b)) )

cout<<1<<endl;

else

cout<<0<<endl;

}

return 0;

}

6. **Roman Number to Integer**

Given an string in roman no format (s)  your task is to convert it to integer .  
  
**Input:**  
The first line of each test case contains the no of test cases T. Then T test cases follow. Each test case contains a string s denoting the roman no.  
  
**Output:**  
For each test case in a new line print the integer representation of roman number s.   
  
**Constraints:**  
1<=T<=100  
1<=roman no range<4000  
  
**Example:  
Input**  
2  
V  
III   
**Output**  
5  
3

Code:

using namespace std;

int main()

{

int t;

cin>>t;

//pre declaration

map<char,int> mp;

mp['I']=1;

mp['V']=5;

mp['X']=10;

mp['L']=50;

mp['C']=100;

mp['D']=500;

mp['M']=1000;

while(t--)

{

string str;

cin>>str;

int n=str.length();

stack<int> s;

s.push(mp[str[0]]);

int tp=s.top();

int i=1;

while(i<n)

{

int nxt=mp[str[i]];

//make top element -ve that is it will subs from total sum

if(tp<nxt)

{

s.pop();

s.push(-tp);

//now add next element in total

s.push(nxt);

tp=s.top();

}

//add nxt element in total

else

{

s.push(nxt);

tp=s.top();

}

i++;

}

//calcuating the sum

int sum=0;

while(!s.empty())

{

sum+=s.top();

s.pop();

}

cout<<sum<<endl;

}

return 0;

}

7. **Anagram**

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

**Input:**  
The first line of input contains an integer T denoting the number of test cases. Each test case consist of two strings in 'lowercase' only, in a single line.

**Output:**  
Print "**YES**" without quotes if the two strings are anagram else print "**NO**".

**Constraints:**  
1 ≤ T ≤ 300  
1 ≤ |s| ≤ 106

**Example:**  
**Input:**  
2  
geeksforgeeks forgeeksgeeks  
allergy allergic

**Output:**  
YES  
NO

Code:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

string str1,str2;

cin>>str1;

cin>>str2;

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

{

cout<<"NO"<<endl;

continue;

}

map<char,int> mp1;

map<char,int> mp2;

int n=str1.length();

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

{

mp1[str1[i]]++;

mp2[str2[i]]++;

}

int flag=0;

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

{

if(mp1[str1[i]]!=mp2[str1[i]])

{

cout<<"NO"<<endl;

flag=1;

break;

}

}

if(!flag)

cout<<"YES"<<endl;

}

return 0;

}

8. **Remove Duplicates**

Given a **string**, the task is to remove duplicates from it. Expected time complexity O(n) where n is length of input string and extra space O(1) under the assumption that there are total 256 possible characters in a string.

**Note:** that original order of characters must be kept same.

**Input:**  
First line of the input is the number of test cases **T**. And first line of test case contains a string.

**Output:**  
Modified string without duplicates and same order of characters.

**Constraints:**   
1 <= T <= 15  
1 <= |string|<= 1000  
  
**Example:**  
**Input:**  
2  
geeksforgeeks  
geeks for geeks

**Output:**  
geksfor  
geks for

Code:

using namespace std;

int main()

{

int t;

cin>>t;

cin.ignore();

while(t--)

{

string str;

getline(cin,str);

int n=str.length();

map<char,int> mp;

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

{

if(mp[str[i]]==0)

{

cout<<str[i];

mp[str[i]]++;

}

}

cout<<endl;

}

return 0;

}

9. **Form a palindrome**

Given a string, find the minimum number of characters to be inserted to convert it to palindrome.  
For Example:  
ab: Number of insertions required is 1. **b**ab or aba  
aa: Number of insertions required is 0. aa  
abcd: Number of insertions required is 3. **dcb**abcd

**Input:**

The first line of input contains an integer T denoting the number of test cases.  
The first line of each test case is S.  
  
**Output:**

Print the minimum number of characters.  
  
**Constraints:**

1 ≤ T ≤ 50  
1 ≤ S ≤ 40  
  
**Example:**

**Input:**  
3  
abcd  
aba  
geeks

**Output:**  
3  
0  
3

Code:

using namespace std;

int lps(string str)

{

string rstr=str;

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

int n=str.length();

int m=n;

int tb[n+1][m+1];

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

{

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

{

if(i==0 || j==0)

tb[i][j]=0;

else if(str[i-1]==rstr[j-1])

tb[i][j]=tb[i-1][j-1]+1;

else

tb[i][j]=max(tb[i-1][j],tb[i][j-1]);

}

}

return tb[n][m];

}

int main()

{

int t;

cin>>t;

while(t--)

{

string str;

cin>>str;

int n=str.length();

cout<<n-lps(str)<<endl;

}

return 0;

}

10. **Longest Distinct characters in string**

Given a string **S**, find length of the longest substring with all distinct characters.  For example, for input "abca", the output is 3 as "abc" is the longest substring with all distinct characters.

**Input:**  
The first line of input contains an integer T denoting the number of test cases.  
The first line of each test case is String str.

**Output:**  
Print length of smallest substring with maximum number of distinct characters.  
**Note:** The output substring should have all distinct characters.

**Constraints:**  
1 ≤ T ≤ 100  
1 ≤ size of str ≤ 10000

**Example:**  
**Input:**  
2  
abababcdefababcdab  
geeksforgeeks

**Output:**  
6  
7

Code:

int main()

{

int t;

cin>>t;

while(t--)

{

string s;

cin>>s;

int n=s.length();

int ans=1;

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

{

map<char,int> mp;

int cnt=0;

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

{

//new char in this substring so increase cnt and add it

if(mp[s[i]]==0)

{

mp[s[i]]++;

cnt++;

}

//char already then this cnt is max value for this substring

else if(mp[s[i]]>0)

{

break;

}

ans=max(cnt,ans);

}

}

cout<<ans<<endl;

}

return 0;

}

11. **Implement Atoi**

Your task  is to implement the function **atoi**. The function takes a string(str) as argument and converts it to an integer and returns it.

**Example 1:**

**Input:**

str = 123

**Output:** 123

**Example 2:**

**Input:**

str = 21a

**Output:** -1

**Explanation:** Output is -1 as all

characters are not digit only.

**Your Task:**  
Complete the function **atoi()** which takes a string as input parameter and returns integer value of it. if the input string is not a numerical string then returns 1..

**Expected Time Complexity:** O(|S|), |S| = length of string S.  
**Expected Auxiliary Space:** O(1)  
  
**Constraints:**  
1<=length of (s,x)<=10

Code:

#include <bits/stdc++.h>

using namespace std;

int atoi(string str);

int main()

{

int t;

cin>>t;

while(t--)

{

string s;

cin>>s;

cout<<atoi(s)<<endl;

}

}// } Driver Code Ends

/\*You are required to complete this method \*/

int atoi(string str)

{

int sum=0;

int p=0;

int flag=0;

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

for(int i=0;i<str.length();i++)

{

int v=int(str[i]);

if(str[i]=='-')

flag=1;

else if(v>=48 && v<=57)

{

sum+=int(str[i]-'0')\*pow(10,p);

p++;

}

else

return -1;

}

if(flag)

return -sum;

return sum;

//Your code here

}

12. **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).

**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| <= 1000

Code:

#include<bits/stdc++.h>

using namespace std;

int strstr(string ,string);

int main()

{

int t;

cin>>t;

while(t--)

{

string a;

string b;

cin>>a;

cin>>b;

cout<<strstr(a,b)<<endl;

}

}

// } Driver Code Ends

/\* The function should return position where the target string

matches the string str

Your are required to complete this method \*/

int strstr(string s, string x)

{

int n=s.length();

int m=x.length();

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

{

int cnt=0;

int t=i;

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

{

if(x[j]==s[t])

{

t++;

cnt++;

}

else

break;

}

if(cnt==m)

return i;

}

return -1;

//Your code here

}

13. **Longest Common Prefix in an Array**

Given a array of **N** strings, find the longest common prefix among all strings present in the array.

**Input:**  
The first line of the input contains an integer **T** which denotes the number of test cases to follow. Each test case contains an integer **N**. Next line has space separated **N** strings.

**Output:**  
Print the longest common prefix as a string in the given array. If no such prefix exists print "-1"(without quotes).  
  
**Constraints:**  
1 <= T <= 103  
1 <= N <= 103  
1 <= |S| <= 103

**Example:  
Input:**  
2  
4  
geeksforgeeks geeks geek geezer  
3  
apple ape april

**Output:**  
gee  
ap

Code:

using namespace std;

int main()

{

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

int t; cin>>t;

while(t--){

int n; cin>>n;

vector<string>v;

int x = INT\_MAX;

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

string s; cin>>s;

if(s.size()<x){

x = s.size();

}

v.push\_back(s);

}

int cnt = 0, flag = 0;

string s = "";

for(int i=0;i<x;i++){

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

if(v[j][i]!=v[0][i]){

flag = 1;

break;

}

}

if(flag){

break;

}

cnt++;

s += v[0][i];

}

if(cnt==0){

cout<<-1<<endl;

}

else{

cout<<s<<endl;

}

}

return 0;

}