1. **A PROGRAM TO CHECK IF THE STRING ARE ROTATIONS OF EACH OTHER OR NOT**

**bool rotateString(string A, string B) {**

**if(A.length() != B.length()) return false;**

**if(A.length() == 0 && B.length() == 0) return true;**

**if(A.length() == 0 || B.length() == 0) return false;**

**B = B+B;**

**return B.find(A) != string::npos;**

**}**

1. **Write a program to check if a string is valid shuffle of 2 strings or not**

**//passes 86/101 test cases**

**bool isInterleave(string s1, string s2, string s3) {**

**int i=0,j=0,k=0;**

**int l1=s1.length(),l2=s2.length(),l3=s3.length();**

**if(l1==0 && l2==0 && l3==0) return true;**

**while(i<l1 || j<l2 || k<l3)**

**{**

**if(s1[i]==s3[k] && s2[j]!=s3[k])**

**{**

**i++;**

**k++;**

**}**

**else if(s1[i]!=s3[k] && s2[j]==s3[k])**

**{**

**j++;**

**k++;**

**}**

**else if(s1[i]==s3[k] && s2[j]==s3[k])**

**{**

**int k1=k,k2=k;**

**int i1=i,j1=j;**

**int cnt1=0,cnt2=0;**

**while(i1<l1 && s1[i1]==s3[k1])**

**{**

**i1++;**

**k1++;**

**cnt1++;**

**}**

**while(j1<l2 && s2[j1]==s3[k2])**

**{**

**j1++;**

**k2++;**

**cnt2++;**

**}**

**if(cnt1>cnt2)**

**{**

**i = i1;**

**k = k1;**

**}**

**else**

**{**

**j = j1;**

**k = k2;**

**}**

**}**

**else**

**return false;**

**}**

**return true;**

**}**

1. **Longest Palindrome Substring**

**string longestPalindrome(string s) {**

**int len = s.length();**

**if(len < 2) return s;**

**int newlen=0,maxlen=1,low=0;**

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

**{**

**int j=i,k=i; //j->left , k->right**

**//skip duplicates**

**while(k<len-1 && s[k]==s[k+1]) k++;**

**//Increment 'i' to k+1, so it will start next iteration**

**//from NON Duplicate element**

**i=k+1;**

**//expand the palimdrome**

**while(j>0 && k<len-1 && s[j-1]==s[k+1]) j--,k++;**

**//calculate new len**

**newlen = k-j+1;**

**//if newlen > maxlen, update low and maxlen**

**if(newlen>maxlen)**

**{**

**low = j;**

**maxlen = newlen;**

**}**

**}**

**return s.substr(low,maxlen);**

**}**

1. **Longest Common Subsequnce (LC)**

**int longestCommonSubsequence(string text1, string text2) {**

**int len1 = text1.length();**

**int len2 = text2.length();**

**vector<vector<int>> dp(len1+1,vector<int>(len2+1));**

**for(int i=1;i<=len1;i++)**

**for(int j=1;j<=len2;j++)**

**{**

**if(text1[i-1] == text2[j-1])**

**{**

**dp[i][j] = dp[i-1][j-1] + 1;**

**}**

**else**

**{**

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

**}**

**}**

**return dp[len1][len2];**

**}**

1. **Longest common Subsequence (GFG)**

**5-> Longest Repeating Subsequnce**

**int LongestRepeatingSubsequence(string str){**

**// Code here**

**int len = str.length();**

**vector<vector<int>> dp(len+1,vector<int>(len+1));**

**for(int i=1;i<=len;i++)**

**{**

**for(int j=1;j<=len;j++)**

**{**

**if(str[i-1] == str[j-1] && i!=j)**

**{**

**dp[i][j] = dp[i-1][j-1] + 1;**

**}**

**else**

**{**

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

**}**

**}**

**}**

**return dp[len][len];**

**}**

1. **Print all Subsequences of a string**

**void printSubSeqRec(string str, int n, int index = -1, string curr = "")**

**{**

**// base case**

**if (index == n)**

**return;**

**if (!curr.empty()) {**

**cout << curr << "\n";**

**}**

**for (int i = index + 1; i < n; i++) {**

**curr += str[i];**

**printSubSeqRec(str, n, i, curr);**

**// backtracking**

**curr = curr.erase(curr.size() - 1);**

**}**

**return;**

**}**

1. **Print all permutation of string**

**7-> Print all permutation of string**

**void backtrack(vector<string>&result,string curr,string s)**

**{**

**// base case**

**if (curr.length() == s.length())**

**{**

**result.push\_back(curr);**

**}**

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

**if(find(curr.begin(),curr.end(),s[i])!=curr.end()) continue;**

**curr += s[i];**

**backtrack(result,curr,s);**

**curr.pop\_back();**

**}**

**}**

**void permut(string s)**

**{**

**vector<string> result;**

**string temp;**

**backtrack(result,temp,s);**

**sort(result.begin() , result.end());**

**for(string str:result)**

**cout<<str<<" ";**

**}**

1. **Short Encoding of words**

**int minimumLengthEncoding(vector<string>& words) {**

**int res=0;**

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

**reverse(words[i].begin() , words[i].end());**

**sort(words.begin() , words.end());**

**for(auto str:words)**

**cout<<str<<" ";**

**for(int i=0;i<words.size()-1;i++)**

**{**

**// 32/33 cases passed by this method**

**// res+=words[i+1].find(words[i]) != string::npos?0:words[i].size()+1;**

**res += words[i]==words[i+1].substr(0,words[i].size())?0:words[i].size()+1;**

**}**

**return res + words[words.size()-1].size()+1;**

**}**