PROBLEM 1: Two strings are said to be the same if they are of the same length and have the same character at each index. Backspacing in a string removes the previous character in the string. Given two strings containing lowercase English letters and the character '#' which represents a backspace key, determine if the two final strings are equal. Return 1 if they are equal or 0 if they are not. Note that backspacing an empty string results in an empty string.

Code:

#include <iostream>

#include <string>

using namespace std;

bool backspaceCompare(string s1, string s2) {

string final\_s1, final\_s2;

for (char ch : s1) {

if (ch == '#') {

if (!final\_s1.empty()) {

final\_s1.pop\_back();

}

} else {

final\_s1.push\_back(ch);

}

}

for (char ch : s2) {

if (ch == '#') {

if (!final\_s2.empty()) {

final\_s2.pop\_back();

}

} else {

final\_s2.push\_back(ch);

}

}

return final\_s1 == final\_s2;

}

int main() {

string s1 = "axx#bb#c";

string s2 = "axbd#c#c";

if (backspaceCompare(s1, s2)) {

cout << "The two strings are equal." << endl;

return 1;

} else {

cout << "The two strings are not equal." << endl;

return 0;

    }

}

PROBLEM 2: Data, in the form of a binary string has to be sent across two servers. However, according to a new network control protocol, data can only be sent in the form of binary strings that have no two adjacent characters same. Such binary strings with no two adjacent characters same are called special strings. Any data to be transmitted is first broken into one/numerous subsequences that are special strings and then each special string is sent as a data packet across the connected servers. Given a binary string that has to be sent across two servers, find the minimum number of data packets it will be broken into. Note: A subsequence of a string is obtained by deleting some characters from the string while maintaining the order. For example, "011" is a subsequence of "0101" while "100" is not.

Code:

#include <iostream>

#include <string>

using namespace std;

int minDataPackets(string input\_str) {

int count = 1;

int n = input\_str.length();

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

if (input\_str[i] == input\_str[i - 1]) {

count++;

input\_str[i] = '#';

}

}

return count;

}

int main() {

string input\_str = "00100";

int minPackets = minDataPackets(input\_str);

cout << "Minimum number of data packets: " << minPackets << endl;

    return 0;

}

PROBLEM 3: An English lecture at Elementary School is aimed at teaching students the letters of the alphabet. The students are provided with a string word that consists of lowercase English letters. In one move, they can choose any index i, and let the character at that index be c. Then, the first occurrence of c to the left of i, and the first occurrence of c to the right of i are deleted (Note: the operation can still be carried out even if either the left or right occurrence doesn't exist). For example, if word = "adabacaea", and if index 4 is chosen (0-based), the first occurrence of 'a' to the left and right of index 4 (bold, indices 2 and 6) are deleted leaving word = "adbacea". Find the minimum number of moves the students need to perform in order to get a word of minimal length.

Code:

#include <iostream>

#include <string>

using namespace std;

int minMovesToMinimalWord(string word) {

int moves = 0;

while (true) {

bool deletion = false;

string minimalWord;

int i = 0;

while (i < word.length()) {

char ch = word[i];

int left = i - 1;

int right = i + 1;

bool deleted = false;

while (left >= 0 && word[left] == ch) {

deletion = true;

deleted = true;

left--;

}

while (right < word.length() && word[right] == ch) {

deletion = true;

right++;

}

if (!deleted) {

minimalWord += ch;

i++;

}

else {

i = right;

}

}

if (!deletion) {

break;

}

word = minimalWord;

moves++;

}

while (right < word.length() && word[right] == ch) {

deletion = true;

right++;

}

if (!deleted) {

minimalWord += ch;

i++;

}

else {

i = right;

}

}

if (!deletion) {

break;

}

word = minimalWord;

moves++;

}

/\* PROBLEM 4:

A string is a pangram if it contains all letters of the English alphabet, ascii['a'-'z']. Given a list of strings, determine if each one is

a pangram or not. Return "1" if true and "0" if false.\*/

#include<stdio.h>

#include<conio.h>

void main()

{

char str[100]="The quick brown fox jumps over the lazy dog";

int i,value[26]={0},count=0;

for(i=0;str[i]!='\0';i++)

{

if('a'<=str[i] && str[i]<='z')

{

count+=!value[str[i]-'a'];

value[str[i]-'a']=1;

}

else if('A'<=str[i] && str[i]<='Z')

{

count+=!value[str[i]-'A'];

value[str[i]-'A']=1;

}

}

if(count==26)

{

printf("The String is a Pangram String.");

}

else

{

printf("The String is not a Pangram String.");

}

getch();

}

PROBLEM 5: Parentheses strings are strings containing only the characters '(' and ')'. A parentheses string is considered balanced when its opening parentheses align with its closing parentheses. For example, "()()" and "(()())" are balanced parentheses strings while ")(", "())(" are not. Given a string consisting of the same number of opening and closing parentheses, determine the minimum number of character swaps required to make the string a balanced parentheses string.

Code:

#include <iostream>

#include <string>

#include <stack>

using namespace std;

int minSwapsToBalance(const string& s) {

int swaps = 0;

int imbalance = 0;

for (char ch : s) {

if (ch == '(') {

imbalance++;

} else {

if (imbalance > 0) {

imbalance--;

} else {

swaps++;

}

}

}

return swaps;

}

int main() {

string s = "))((";

int minSwaps = minSwapsToBalance(s);

cout << "Minimum number of swaps: " << minSwaps << endl;

return 0;

}

PROBLEM 6: The 26 characters of the alphabet are each assigned a security value represented as an array of integers, where security\_values[i] is associated with the i th character of the alphabet. Given an encrypted message, msg, and the array security\_values, rearrange the characters in msg and find the minimum possible sum of the absolute differences of the security values of adjacent characters. Example: Given security\_values[i] = [1, 2, 1, 3, 1, 3, 5, 7, 1, 1, 5, 5, 8, 10, 11, 1, 23, 2, 3, 7, 8, 9, 1, 6, 5, 9] and s = "afeb". Here 'a' relates to 1, 'f' relates to 3, 'e' relates to 1, and 'b' relates to 2. Some of the rearrangements are: The optimal rearrangement is "eabf" or "fbae" with a sum of 2.

Code:

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

int minimumSumOfDifferences(const vector<int>& security\_values, const string& msg) {

vector<pair<int, char>> sorted\_chars;

// Create a vector of pairs (security\_value, character)

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

sorted\_chars.push\_back({ security\_values[msg[i] - 'a'], msg[i] });

}

// Sort the characters based on their security values

sort(sorted\_chars.begin(), sorted\_chars.end());

int sum = 0;

for (int i = 1; i < sorted\_chars.size(); i++) {

sum += abs(sorted\_chars[i].first - sorted\_chars[i - 1].first);

}

return sum;

}

int main() {

vector<int> security\_values = { 1, 2, 1, 3, 1, 3, 5, 7, 1, 1, 5, 5, 8, 10, 11, 1, 23, 2, 3, 7, 8, 9, 1, 6, 5, 9 };

string msg = "afeb";

int minSum = minimumSumOfDifferences(security\_values, msg);

cout << "Minimum sum of differences: " << minSum << endl;

return 0;

}

\* PROBLEM 7:

Given a string, how many different substrings exist in it that have no repeating characters? Two substrings are considered

different if they have a different start or end index.\*/

CODE:

def uniqueSubstringCount(word):

substrings = []

for i in range(len(word)):

temp\_word = ""

if word[i] not in substrings:

substrings.append(word[i])

for k in range(len(word)):

if (word[i] != word[k] and word[i] + word[k] not in substrings):

substrings.append(word[i] + word[k])

if (word[i] not in temp\_word):

temp\_word += word[i]

if (word[k] not in temp\_word):

temp\_word += word[k]

for j in range(len(word)):

if word[j] in temp\_word:

continue

elif(word[j] not in temp\_word and str(temp\_word + word[j]) not in substrings):

temp\_word += word[j]

substrings.append(temp\_word)

temp\_word = ""

return len(substrings)

word = "sky"

print("Word:", word)

print("Total substrings with unique characters:", uniqueSubstringCount(word.lower()))

/\*PROBLEM 8:

A palindrome is a string that reads the same from the left and from the right. For example, mom and tacocat are palindromes,

as are any single-character strings. Given a string, determine the number of its substrings that are palindromes.\*/

#include <iostream>

#include<string>

using namespace std;

bool isPalindromic(string s, int i, int j)

{

if (i > j)

return 1;

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

return 0;

return isPalindromic(s, i + 1, j - 1);

}

int main() {

string str;

cin>>str;

int count =0;

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

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

if(isPalindromic(str,i,j)){

count++;

}

}

}

cout<<count;

}

PROBLEM 9:

Different compression techniques are used in order to reduce the size of the messages sent over the web. An algorithm is

designed to compress a given string by describing the total number of consecutive occurrences of each character next to it.\*/

Code:

#include <iostream>

using namespace std;

void gen\_compressed\_str(string str)

{

int len = str.length();

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

// Count occurrences of current character

int count = 1;

while (i < len - 1 && str[i] == str[i + 1]) {

count++;

i++;

}

// Print character and its count

if (count == 1)

{

cout << str[i];

}

else

{

cout << str[i] << count;

}

}

}

int main() {

string str = "wwwwaaadexxxxxxywww";

gen\_compressed\_str(str);

}