**Sum of two large numbers**

// C++ program to find sum of two large numbers.

#include<bits/stdc++.h>

using namespace std;

// Function for finding sum of larger numbers

string findSum(string str1, string str2)

{

// Before proceeding further, make sure length of str2 is larger.

if (str1.length() > str2.length())

swap(str1, str2);

string str = "";

int n1 = str1.length();

int n2 = str2.length();

// Reverse both of strings

reverse(str1.begin(), str1.end());

reverse(str2.begin(), str2.end());

int carry = 0;

for (int i=0; i<n1; i++)

{

// Do school mathematics, compute sum of current digits and carry

int sum = ((str1[i]-'0')+(str2[i]-'0')+carry);

str.push\_back(sum%10 + '0');

// Calculate carry for next step

carry = sum/10;

}

// Add remaining digits of larger number

for (int i=n1; i<n2; i++)

{

int sum = ((str2[i]-'0')+carry);

str.push\_back(sum%10 + '0');

carry = sum/10;

}

// Add remaining carry

if (carry)

str.push\_back(carry+'0');

// reverse resultant string

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

return str;

}

// Driver code

int main()

{

string str1 = "12";

string str2 = "198111";

cout << findSum(str1, str2);

return 0;

}

**Difference of two large numbers**

#include<bits/stdc++.h>

using namespace std;

// Returns true if str1 is smaller than str2, else false.

bool isSmaller(string str1, string str2)

{

    int n1 = str1.length(), n2 = str2.length();

    if (n1 < n2)

        return true;

    if (n2 > n1)

        return false;

    for (int i=0; i<n1; i++)

    {

        if (str1[i] < str2[i])

            return true;

        else if (str1[i] > str2[i])

            return false;

    }

    return false;

}

// Function for finding difference of larger numbers

string findDiff(string str1, string str2)

{

    // Before proceeding further, make sure str1 is not smaller

    if (isSmaller(str1, str2))

        swap(str1, str2);

    string str = "";

    int n1 = str1.length(), n2 = str2.length();

    int diff = n1 - n2;

      int carry = 0;

   // Traverse from end of both strings

    for (int i=n2-1; i>=0; i--)

    {

        // Do school mathematics, compute difference of current digits and carry

        int sub = ((str1[i+diff]-'0') -

                   (str2[i]-'0') -

                   carry);

        if (sub < 0)

        {

            sub = sub+10;

            carry = 1;

        }

        else

            carry = 0;

        str.push\_back(sub + '0');

    }

    // subtract remaining digits of str1[]

    for (int i=n1-n2-1; i>=0; i--)

    {

        if (str1[i]=='0' && carry)

        {

            str.push\_back('9');

            continue;

        }

        int sub = ((str1[i]-'0') - carry);

        if (i>0 || sub>0) // remove preceding 0's

            str.push\_back(sub+'0');

        carry = 0;

    }

    // reverse resultant string

    reverse(str.begin(), str.end());

    return str;

}

// Driver code

int main()

{

    string str1 = "88";

    string str2 = "1079";

    cout << findDiff(str1, str2);

    return 0;

}

**Multiply Large Numbers represented as Strings**

import java.util.Scanner;

public class StringMultiplication

{

    // Driver code

    public static void main(String[] args)

    {

        String num1 = "1235421415454545454545454544";

        String tempnum1 = num1;

        String num2 = "1714546546546545454544548544544545";

        String tempnum2 = num2;

        // Check condition if one string is negative

        if(num1.charAt(0) == '-' && num2.charAt(0)!='-')

        {

            num1 = num1.substring(1);

        }

        else if(num1.charAt(0) != '-' && num2.charAt(0) == '-')

        {

            num2 = num2.substring(1);

        }

        else if(num1.charAt(0) == '-' && num2.charAt(0) == '-')

        {

            num1 = num1.substring(1);

            num2 = num2.substring(1);

        }

        String s1 = new StringBuffer(num1).reverse().toString();

        String s2 = new StringBuffer(num2).reverse().toString();

        int[] m = new int[s1.length()+s2.length()];

          // Go from right to left in num1

        for (int i = 0; i < s1.length(); i++)

        {

           // Go from right to left in num2

            for (int j = 0; j < s2.length(); j++)

            {

                m[i+j] = m[i+j]+(s1.charAt(i)-'0')\*(s2.charAt(j)-'0');

            }

        }

 String product = new String();

 //Multiply with current digit of first number and add result to previously stored product at current position.

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

        {

            int digit = m[i]%10;

            int carry = m[i]/10;

            if(i+1<m.length)

            {

                m[i+1] = m[i+1] + carry;

            }

            product = digit+product;

        }

        // ignore '0's from the right

        while(product.length()>1 && product.charAt(0) == '0')

        {

            product = product.substring(1);

        }

        // Check condition if one string is negative

        if(tempnum1.charAt(0) == '-' && tempnum2.charAt(0)!='-')

        {

            product = new StringBuffer(product).insert(0,'-').toString();

        }

        else if(tempnum1.charAt(0) != '-' && tempnum2.charAt(0) == '-')

        {

            product = new StringBuffer(product).insert(0,'-').toString();

        }

        else if(tempnum1.charAt(0) == '-' && tempnum2.charAt(0) == '-')

        {

            product = product;

        }

        System.out.println("Product of the two numbers is :"+"\n"+product);

    }

}

**Divide large number represented as string**

#include <bits/stdc++.h>

using namespace std;

// A function to perform division of large numbers

string longDivision(string number, int divisor)

{

    string ans;

    // Find prefix of number that is larger than divisor.

    int idx = 0;

    int temp = number[idx] - '0';

    while (temp < divisor)

       temp = temp \* 10 + (number[++idx] - '0');

    // Repeatedly divide divisor with temp. After every division, update temp to include one more digit.

    while (number.size() > idx)

    {

        // Store result in answer i.e. temp / divisor

        ans += (temp / divisor) + '0';

        // Take next digit of number

        temp = (temp % divisor) \* 10 + number[++idx] - '0';

    }

    // If divisor is greater than number

    if (ans.length() == 0)

        return "0";

    // else return ans

    return ans;

}

// Driver program to test longDivison()

int main()

{

    string number = "1248163264128256512";

    int divisor = 125;

    cout << longDivision(number, divisor);

    return 0;

}

**Remainder with 7 for large numbers**

class GFG

{

    // Function which return Remainder  after dividingthe number by 7

    static int remainderWith7(String num)

    {

        // This series is used to  find remainder with 7

        int series[] = {1, 3, 2, -1, -3, -2};

        // Index of next element in series

        int series\_index = 0;

        int result = 0;

        // Traverse num from end

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

        {

            /\* Find current digit of nun \*/

            int digit = num.charAt(i) - '0';

            // Add next term to result

            result += digit \* series[series\_index];

            // Move to next term in series

            series\_index = (series\_index + 1) % 6;

            // Make sure that result never goes beyond 7.

            result %= 7;

        }

        // Make sure that remainder is positive

        if (result < 0)

        result = (result + 7) % 7;

        return result;

    }

    // Driver code

    public static void main (String[] args)

    {

        String str = "12345";

        System.out.print("Remainder with 7 is "

                          +remainderWith7(str));

    }

}

**Sorting array of strings (or words) using Trie**

Given an array of strings, print them in alphabetical (dictionary) order. If there are duplicates in input array, we need to print them only once.

Examples:

Input : "abc", "xy", "bcd"

Output : abc bcd xy

Input : "geeks", "for", "geeks", "a", "portal",

"to", "learn", "can", "be", "computer",

"science", "zoom", "yup", "fire", "in", "data"

Output : a be can computer data fire for geeks

in learn portal science to yup zoom

[Trie](https://www.geeksforgeeks.org/trie-insert-and-search/) is an efficient data structure used for storing data like strings. To print the string in alphabetical order we have to first insert in the trie and then perform preorder traversal to print in alphabetical order.

#include <bits/stdc++.h>

using namespace std;

const int MAX\_CHAR = 26;

struct Trie {

    // index is set when node is a leaf node, otherwise -1;

    int index;

    Trie\* child[MAX\_CHAR];

    /\*to make new trie\*/

    Trie()

    {

        for (int i = 0; i < MAX\_CHAR; i++)

            child[i] = NULL;

        index = -1;

    }

};

/\* function to insert in trie \*/

void insert(Trie\* root, string str, int index)

{

    Trie\* node = root;

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

        /\* taking ascii value to find index of

          child node \*/

        char ind = str[i] - 'a';

        /\* making new path if not already \*/

        if (!node->child[ind])

            node->child[ind] = new Trie();

        // go to next node

        node = node->child[ind];

    }

    // Mark leaf (end of word) and store

    // index of word in arr[]

    node->index = index;

}

/\* function for preorder traversal \*/

bool preorder(Trie\* node, string arr[])

{

    if (node == NULL)

        return false;

    for (int i = 0; i < MAX\_CHAR; i++) {

        if (node->child[i] != NULL) {

            /\* if leaf node then print key\*/

            if (node->child[i]->index != -1)

                cout << arr[node->child[i]->index] << endl;

            preorder(node->child[i], arr);

        }

    }

}

void printSorted(string arr[], int n)

{

    Trie\* root = new Trie();

    // insert all keys of dictionary into trie

    for (int i = 0; i < n; i++)

        insert(root, arr[i], i);

    // print keys in lexicographic order

    preorder(root, arr);

}

// Driver code

int main()

{

    string arr[] = { "abc", "xy", "bcd" };

    int n = sizeof(arr) / sizeof(arr[0]);

    printSorted(arr, n);

    return 0;

}

**Print distinct sorted permutations with duplicates allowed in input**

*Input : BAC  
Output : ABC ACB BAC BCA CAB CBA*

*Input : AAB  
Output : AAB ABA BAA*

*Input : DBCA  
Output: ABCD ABDC ACBD ACDB ADBC ADCB BACD BADC BCAD BCDA BDAC BDCA CABD CADB CBAD CBDA CDAB CDBA DABC DACB DBAC DBCA DCAB DCBA*

import java.io.\*;

import java.util.\*;

class Solution {

  // Calculating factorial of a number

  static int factorial(int n) {

    int f = 1;

    for (int i = 1; i <= n; i++)

      f = f \* i;

    return f;

  }

  // Method to print the array

  static void print(char[] temp) {

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

      System.out.print(temp[i]);

    System.out.println();

  }

  // Method to find total number of permutations

  static int calculateTotal(char[] temp, int n) {

    int f = factorial(n);

    // Building HashMap to store frequencies of

    // all characters.

    HashMap<Character, Integer> hm =

                     new HashMap<Character, Integer>();

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

      if (hm.containsKey(temp[i]))

        hm.put(temp[i], hm.get(temp[i]) + 1);

      else

        hm.put(temp[i], 1);

    }

    // Traversing hashmap and finding duplicate elements.

    for (Map.Entry e : hm.entrySet()) {

      Integer x = (Integer)e.getValue();

      if (x > 1) {

        int temp5 = factorial(x);

        f = f / temp5;

      }

    }

    return f;

  }

  static void nextPermutation(char[] temp) {

    // Start traversing from the end and

    // find position 'i-1' of the first character

    // which is greater than its  successor.

    int i;

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

      if (temp[i] > temp[i - 1])

        break;

    // Finding smallest character after 'i-1' and

    // greater than temp[i-1]

    int min = i;

    int j, x = temp[i - 1];

    for (j = i + 1; j < temp.length; j++)

      if ((temp[j] < temp[min]) && (temp[j] > x))

        min = j;

    // Swapping the above found characters.

    char temp\_to\_swap;

    temp\_to\_swap = temp[i - 1];

    temp[i - 1] = temp[min];

    temp[min] = temp\_to\_swap;

    // Sort all digits from position next to 'i-1'

    // to end of the string.

    Arrays.sort(temp, i, temp.length);

    // Print the String

    print(temp);

  }

  static void printAllPermutations(String s) {

    // Sorting String

    char temp[] = s.toCharArray();

    Arrays.sort(temp);

    // Print first permutation

    print(temp);

    // Finding the total permutations

    int total = calculateTotal(temp, temp.length);

    for (int i = 1; i < total; i++)

      nextPermutation(temp);

  }

  // Driver Code

  public static void main(String[] args) {

    String s = "AAB";

    printAllPermutations(s);

  }

}

**Permute a string by changing case**

Input : ab

Output : AB Ab ab aB

Input : ABC

Output : abc Abc aBc ABc abC AbC aBC ABC

public class PermuteString

{

    // Function to generate permutations

    static void permute(String input)

    {

        int n = input.length();

        // Number of permutations is 2^n

        int max = 1 << n;

        // Converting string to lower case

        input = input.toLowerCase();

        // Using all subsequences and permuting them

        for(int i = 0;i < max; i++)

        {

            char combination[] = input.toCharArray();

            // If j-th bit is set, we convert it to upper case

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

            {

                if(((i >> j) &  1) == 1)

                    combination[j] = (char) (combination[j]-32);

            }

            // Printing current combination

            System.out.print(combination);

            System.out.print("   ");

        }

    }

    // Driver Program to test above function

    public static void main(String[] args)

    {

        permute("ABC");

    }

}

**How to design a tiny URL or URL shortener?**

A URL character can be one of the following  
1) A lower case alphabet [‘a’ to ‘z’], total 26 characters  
2) An upper case alphabet [‘A’ to ‘Z’], total 26 characters  
3) A digit [‘0’ to ‘9’], total 10 characters

There are total 26 + 26 + 10 = 62 possible characters.

So the task is to convert a decimal number to base 62 number.

To get the original long url, we need to get url id in database. The id can be obtained using base 62 to decimal conversion.

#include<iostream>

#include<algorithm>

#include<string>

using namespace std;

// Function to generate a short url from intger ID

string idToShortURL(long int n)

{

    // Map to store 62 possible characters

    char map[] = "abcdefghijklmnopqrstuvwxyzABCDEF"

                 "GHIJKLMNOPQRSTUVWXYZ0123456789";

    string shorturl;

    // Convert given integer id to a base 62 number

    while (n)

    {

        // use above map to store actual character

        // in short url

        shorturl.push\_back(map[n%62]);

        n = n/62;

    }

    // Reverse shortURL to complete base conversion

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

    return shorturl;

}

// Function to get integer ID back from a short url

long int shortURLtoID(string shortURL)

{

    long int id = 0; // initialize result

    // A simple base conversion logic

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

    {

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

          id = id\*62 + shortURL[i] - 'a';

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

          id = id\*62 + shortURL[i] - 'A' + 26;

        if ('0' <= shortURL[i] && shortURL[i] <= '9')

          id = id\*62 + shortURL[i] - '0' + 52;

    }

    return id;

}

// Driver program to test above function

int main()

{

    int n = 12345;

    string shorturl = idToShortURL(n);

    cout << "Generated short url is " << shorturl << endl;

    cout << "Id from url is " << shortURLtoID(shorturl);

    return 0;

}

**Smallest length string with repeated replacement of two distinct adjacent**

Input : cab

Output : 2

We can select any two adjacent letters,

say 'ca' and transform it into 'b', this

leaves us with string 'bb' of length two.

Input : bcab

Output : 1

Selecting 'bc' and transforming it to 'a'

leaves us with 'aab'. We can then select

'ab' and transform it to 'c', giving 'ac'.

This can further be transformed into 'b',

which is of length one.

#include<bits/stdc++.h>

using namespace std;

// Program to find length of reduced string

// in a string made of three characters.

#define MAX\_LEN 110

// To store results of subproblems

int DP[MAX\_LEN][MAX\_LEN][MAX\_LEN];

// A memoized function find result recursively.

// a, b and c are counts of 'a's, 'b's and

// 'c's in str

int length(int a, int b, int c)

{

    // If this subproblem is already evaluated

    if (DP[a][b] != -1)

        return DP[a][b];

    // If there is only one type of character

    if (a == 0 && b == 0)

        return (DP[a][b] = c);

    if (a == 0 && c == 0)

        return (DP[a][b] = b);

    if (b == 0 && c == 0)

        return (DP[a][b] = a);

    // If only two types of characters are present

    if (a == 0)

        return (DP[a][b] =

                    length(a + 1, b - 1, c - 1));

    if (b == 0)

        return (DP[a][b] =

                    length(a - 1, b + 1, c - 1));

    if (c == 0)

        return (DP[a][b] =

                    length(a - 1, b - 1, c + 1));

    // If all types of characters are present.

    // Try combining all pairs.

    return (DP[a][b] =

                min(length(a - 1, b - 1, c + 1),

                    min(length(a - 1, b + 1, c - 1),

                        length(a + 1, b - 1, c - 1))));

}

// Returns smallest possible length with given

// operation allowed.

int stringReduction(string str)

{

    int n = str.length();

    // Counting occurrences of three different

    // characters 'a', 'b' and 'c' in str

    int count[3] = {0};

    for (int i=0; i<n; ++i)

        count[str[i]-'a']++;

    // Initialize DP[][] entries as -1

    for (int i = 0; i <= count[0]; ++i)

        for (int j = 0; j < count[1]; ++j)

            for (int k = 0; k < count[2]; ++k)

                DP[i][j][k] = -1;

    return length(count[0], count[1], count[2]);

}

// Driver code

int main()

{

    string str = "abcbbaacb";

    cout << stringReduction(str);

    return 0;

}

**Distributing all balls without repetition**

Input : 4 2 // value of N and K

aabb // colors of given balls

Output : YES

We can give 1st and 3rd ball to the first person,

and 2nd and 4th to the second.

Input : 6 3 // value of N and K

aacaab // colors of given balls

Output : NO

We need to give all balls of color a, but one

ball will stay, that's why answer is NO

// Java program to find if its possible

// to distribute balls without repitiio

import java.io.\*;

public class GFG {

    static int MAX\_CHAR = 26;

    // function to find if its possible

    // to distribute balls or not

    static boolean distributingBalls(long k,

                         long n, String str)

    {

        // count array to count how many

        // times each color has occurred

        int []a = new int[MAX\_CHAR];

        for (int i = 0; i < n; i++)

        {

            // increasing count of each

            // color every time it appears

            a[str.charAt(i) - 'a']++;

        }

        for (int i = 0; i < MAX\_CHAR; i++)

            // to check if any color appears

            // more than K times if it does

            // we will print NO

            if (a[i] > k)

                return false;

        return true;

    }

    // Driver code

    static public void main (String[] args)

    {

        long n = 6, k = 3;

        String str = "aacaab";

        if (distributingBalls(k, n, str))

            System.out.println("YES");

        else

            System.out.println("NO");

    }

}

**Check whether two strings are anagram of each other**

// JAVA program to check whether two strings

// are anagrams of each other

import java.io.\*;

class GFG{

    /\* function to check whether two strings are

    anagram of each other \*/

    static boolean areAnagram(char[] str1, char[] str2)

    {

        // Get lenghts of both strings

        int n1 = str1.length;

        int n2 = str2.length;

        // If length of both strings is not same,

        // then they cannot be anagram

        if (n1 != n2)

            return false;

        // Sort both strings

        quickSort(str1, 0, n1 - 1);

        quickSort(str2, 0, n2 - 1);

        // Compare sorted strings

        for (int i = 0; i < n1;  i++)

            if (str1[i] != str2[i])

                return false;

        return true;

    }

    // Following functions (exchange and partition

    // are needed for quickSort)

    static void exchange(char A[],int a, int b)

    {

        char temp;

        temp = A[a];

        A[a]   = A[b];

        A[b]   = temp;

    }

    static int partition(char A[], int si, int ei)

    {

        char x = A[ei];

        int i = (si - 1);

        int j;

        for (j = si; j <= ei - 1; j++)

        {

            if(A[j] <= x)

            {

                i++;

                exchange(A, i, j);

            }

        }

        exchange (A, i+1 , ei);

        return (i + 1);

    }

    /\* Implementation of Quick Sort

    A[] --> Array to be sorted

    si  --> Starting index

    ei  --> Ending index

    \*/

    static void quickSort(char A[], int si, int ei)

    {

        int pi;    /\* Partitioning index \*/

        if(si < ei)

        {

            pi = partition(A, si, ei);

            quickSort(A, si, pi - 1);

            quickSort(A, pi + 1, ei);

        }

    }

    /\* Driver program to test to print printDups\*/

    public static void main(String args[])

    {

        char str1[] = {'t','e','s','t'};

        char str2[] = {'t','t','e','w'};

        if (areAnagram(str1, str2))

            System.out.println("The two strings are"+

                             " anagram of each other");

        else

            System.out.println("The two strings are not"+

                               " anagram of each other");

    }

}

**Check if two strings are k-anagrams or not**

// Java program to check if two strings are k anagram

// or not.

public class GFG {

    static final int MAX\_CHAR = 26;

    // Function to check that string is k-anagram or not

    static boolean arekAnagrams(String str1, String str2,

                                                 int k)

    {

        // If both strings are not of equal

        // length then return false

        int n = str1.length();

        if (str2.length() != n)

            return false;

        int[] count1 = new int[MAX\_CHAR];

        int[] count2 = new int[MAX\_CHAR];

        int count = 0;

        // Store the occurrence of all characters

        // in a hash\_array

        for (int i = 0; i < n; i++)

            count1[str1.charAt(i) - 'a']++;

        for (int i = 0; i < n; i++)

            count2[str2.charAt(i) - 'a']++;

        // Count number of characters that are

        // different in both strings

        for (int i = 0; i < MAX\_CHAR; i++)

            if (count1[i] > count2[i])

                count = count + Math.abs(count1[i] -

                                          count2[i]);

        // Return true if count is less than or

        // equal to k

        return (count <= k);

    }

    // Driver code

    public static void main(String args[])

    {

        String str1 = "anagram";

        String str2 = "grammar";

        int k = 2;

        if (arekAnagrams(str1, str2, k))

            System.out.println("Yes");

        else

            System.out.println("No");

    }

}

**Given a sequence of words, print all anagrams together using STL**

Given an array of words, print all anagrams together. For example, if the given array is {“cat”, “dog”, “tac”, “god”, “act”}, then output may be “cat tac act dog god”.

// CPP program for finding all anagram

// pairs in the given array

#include <iostream>

#include <algorithm>

#include <unordered\_map>

#include <vector>

using namespace std;

// utility function for printing anagram list

void printAnagram(unordered\_map<string,

                              vector<string> >& store)

{

    unordered\_map<string, vector<string> >::iterator it;

    for (it = store.begin(); it != store.end(); it++) {

        vector<string> temp\_vec(it->second);

        int size = temp\_vec.size();

        if (size > 1) {

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

                cout << temp\_vec[i] << " ";

            }

            cout << "\n";

        }

    }

}

// utility function for storing the vector of strings

// into HashMap

void storeInMap(vector<string>& vec)

{

    unordered\_map<string, vector<string> > store;

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

        string tempString(vec[i]);

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

        // check for sorted string if it already exists

        if (store.find(tempString) == store.end()) {

            vector<string> temp\_vec;

            temp\_vec.push\_back(vec[i]);

            store.insert(make\_pair(tempString, temp\_vec));

        }

        else {

            // push new string to already existing key

            vector<string> temp\_vec(store[tempString]);

            temp\_vec.push\_back(vec[i]);

            store[tempString] = temp\_vec;

        }

    }

    // print utility function for printing

    // all the anagrams

    printAnagram(store);

}

// Driver code

int main()

{

    // initialize vector of strings

    vector<string> arr;

    arr.push\_back("geeksquiz");

    arr.push\_back("geeksforgeeks");

    arr.push\_back("abcd");

    arr.push\_back("forgeeksgeeks");

    arr.push\_back("zuiqkeegs");

    arr.push\_back("cat");

    arr.push\_back("act");

    arr.push\_back("tca");

    // utility function for storing strings

    // into hashmap

    storeInMap(arr);

    return 0;

}

**Minimum Number of Manipulations required to make two Strings Anagram Without Deletion of Character**

/

/ Java Program to find minimum number of manipulations

// required to make two strings identical

public class Similar\_strings {

    // Counts the no of manipulations required

    static int countManipulations(String s1, String s2)

    {

        int count = 0;

        // store the count of character

        int char\_count[] = new int[26];

        // iterate though the first String and update

        // count

        for (int i = 0; i < s1.length(); i++)

            char\_count[s1.charAt(i) - 'a']++;

        // iterate through the second string

        // update char\_count.

        // if character is not found in char\_count

        // then increase count

        for (int i = 0; i < s2.length(); i++)

            if (char\_count[s2.charAt(i) - 'a']-- <= 0)

                count++;

        return count;

    }

    // Driver code

    public static void main(String[] args)

    {

        String s1 = "ddcf";

        String s2 = "cedk";

        System.out.println(countManipulations(s1, s2));

    }

}

**Rearrange characters in a string such that no two adjacent are same**

// C++ program to rearrange characters in a string

// so that no two adjacent characters are same.

#include<bits/stdc++.h>

using namespace std;

const int MAX\_CHAR = 26;

struct Key

{

    int freq; // store frequency of character

    char ch;

    // function for priority\_queue to store Key

    // according to freq

    bool operator<(const Key &k) const

    {

        return freq < k.freq;

    }

};

// Function to rearrange character of a string

// so that no char repeat twice

void rearrangeString(string str)

{

    int n = str.length();

    // Store frequencies of all characters in string

    int count[MAX\_CHAR] = {0};

    for (int i = 0 ; i < n ; i++)

        count[str[i]-'a']++;

    // Insert all characters with their frequencies

    // into a priority\_queue

    priority\_queue< Key > pq;

    for (char c = 'a' ; c <= 'z' ; c++)

        if (count[c-'a'])

            pq.push( Key { count[c-'a'], c} );

    // 'str' that will store resultant value

    str = "" ;

    // work as the previous visited element

    // initial previous element be. ( '#' and

    // it's frequency '-1' )

    Key prev {-1, '#'} ;

    // traverse queue

    while (!pq.empty())

    {

        // pop top element from queue and add it

        // to string.

        Key k = pq.top();

        pq.pop();

        str = str + k.ch;

        // IF frequency of previous character is less

        // than zero that means it is useless, we

        // need not to push it

        if (prev.freq > 0)

            pq.push(prev);

        // make current character as the previous 'char'

        // decrease frequency by 'one'

        (k.freq)--;

        prev = k;

    }

    // If length of the resultant string and original

    // string is not same then string is not valid

    if (n != str.length())

        cout << " Not valid String " << endl;

    else // valid string

        cout << str << endl;

}

// Driver program to test above function

int main()

{

    string str = "bbbaa" ;

    rearrangeString(str);

    return 0;

}

**Lexicographical Maximum substring of string**

Given a string s we have to find the lexicographical maximum substring of a string

Examples:

**Input :** s = "ababaa"

**Output :** babaa

**Explanation :** "babaa" is the maximum lexicographic susbtring formed from this string

**Input :** s = "asdfaa"

**Output :** sdfaa

// CPP program to find the lexicographically

// maximum substring.

#include <bits/stdc++.h>

using namespace std;

string LexicographicalMaxString(string str)

{

    // loop to find the max leicographic

    // substring in the substring array

    string mx = "";

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

        mx = max(mx, str.substr(i));

    return mx;

}

int main()

{

    string str = "ababaa";

    cout << LexicographicalMaxString(str);

    return 0;

}

**Identify and mark unmatched parenthesis in an expression**

Given an expression, find and mark matched and unmatched parenthesis in it. We need to replace all balanced opening parenthesis with 0, balanced closing parenthesis with 1 and all unbalanced with -1.

// CPP program to mark balanced and unbalanced

// parenthesis.

#include <bits/stdc++.h>

using namespace std;

void identifyParenthesis(string a)

{

    stack<int> st;

    // run the loop upto end of the string

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

        // if a[i] is opening bracket then push

        // into stack

        if (a[i] == '(')

            st.push(i);

        // if a[i] is closing bracket ')'

        else if (a[i] == ')') {

            // If this closing bracket is unmatched

            if (st.empty())

                a.replace(i, 1, "-1");

            else {

                // replace all opening brackets with 0

                // and closing brackets with 1

                a.replace(i, 1, "1");

                a.replace(st.top(), 1, "0");

                st.pop();

            }

        }

    }

    // if stack is not empty then pop out all

    // elements from it and replace -1 at that

    // index of the string

    while (!st.empty()) {

        a.replace(st.top(), 1, "-1");

        st.pop();

    }

    // print final string

    cout << a << endl;

}

// Driver code

int main()

{

    string str = "(a))";

    identifyParenthesis(str);

    return 0;

}

**Check if two expressions with brackets are same**

Input : exp1 = "-(a+b+c)"

exp2 = "-a-b-c"

Output : Yes

Input : exp1 = "-(c+b+a)"

exp2 = "-c-b-a"

Output : Yes

Input : exp1 = "a-b-(c-d)"

exp2 = "a-b-c-d"

Output : No

// CPP program to check if two expressions

// evaluate to same.

#include <bits/stdc++.h>

using namespace std;

const int MAX\_CHAR = 26;

// Return local sign of the operand. For example,

// in the expr a-b-(c), local signs of the operands

// are +a, -b, +c

bool adjSign(string s, int i)

{

    if (i == 0)

        return true;

    if (s[i - 1] == '-')

        return false;

    return true;

};

// Evaluate expressions into the count vector of

// the 26 alphabets.If add is true, then add count

// to the count vector of the alphabets, else remove

// count from the count vector.

void eval(string s, vector<int>& v, bool add)

{

    // stack stores the global sign

    // for operands.

    stack<bool> stk;

    stk.push(true);

    // + means true

    // global sign is positive initially

    int i = 0;

    while (s[i] != '\0') {

        if (s[i] == '+' || s[i] == '-') {

            i++;

            continue;

        }

        if (s[i] == '(') {

            // global sign for the bracket is

            // pushed to the stack

            if (adjSign(s, i))

                stk.push(stk.top());

            else

                stk.push(!stk.top());

        }

        // global sign is popped out which

        // was pushed in for the last bracket

        else if (s[i] == ')')

            stk.pop();

        else {

            // global sign is positive (we use different

            // values in two calls of functions so that

            // we finally check if all vector elements

            // are 0.

            if (stk.top())

                v[s[i] - 'a'] += (adjSign(s, i) ? add ? 1 : -1 :

                                                  add ? -1 : 1);

            // global sign is negative here

            else

                v[s[i] - 'a'] += (adjSign(s, i) ? add ? -1 : 1 :

                                                  add ? 1 : -1);

        }

        i++;

    }

};

// Returns true if expr1 and expr2 represent

// same expressions

bool areSame(string expr1, string expr2)

{

    // Create a vector for all operands and

    // initialize the vector as 0.

    vector<int> v(MAX\_CHAR, 0);

    // Put signs of all operands in expr1

    eval(expr1, v, true);

    // Subtract signs of operands in expr2

    eval(expr2, v, false);

    // If expressions are same, vector must

    // be 0.

    for (int i = 0; i < MAX\_CHAR; i++)

        if (v[i] != 0)

            return false;

    return true;

}

// Driver code

int main()

{

    string expr1 = "-(a+b+c)", expr2 = "-a-b-c";

    if (areSame(expr1, expr2))

        cout << "Yes\n";

    else

        cout << "No\n";

    return 0;

}