During World War I, the Germans used a double columnar transposition cipher called *ubchi*. To encrypt a certain message with *ubchi*, one should write it out in rows of a fixed length, and then read out again column by column in specific order. The process should be repeated a second time to ensure safe encryption.

Both the width of the rows and the permutation of the columns are usually defined by a keyword. For example, let the keywordbe "pancake" (a word of length 7). The permutation is then defined by the alphabetical order of the letters in the keyword, i.e. 7, 1, 6, 3, 2, 5, 4 (note that that if the keyword has two equal letters, their order is defined by their order in the keyword).

Let the message to be encrypted be "target acquired. Successful extermination". When written out in columns, the following can be obtained:

7163254

target

acquire

d. Succ

essful

extermi

nation

The columns should be rearranged to obtain the following rectangle:

1234567

aeg trt

ciuerqa

.uScc d

suf lse

xreimte

aoi ntn

This rectangle produces the following string: "ac.sxaeiuuroguSfei ec i trclmnrq stttadeen".

The process should be repeated again with the following results:

7163254 1234567

ac.sxae cxsea.a

iuurogu uorugui

Sfei ec ---> f iceeS

i trcl irtlc

mnrq st n qtsrm

ttadeen tedneat

The resulting encrypted string is thus "cufintxo r esritqdeucltnagecse.ue raaiS mt".

Given encrypted message and the keyword, your task is to decrypt it and return the original message.

**Example**

For message = "cufintxo r esritqdeucltnagecse.ue raaiS mt"  
and keyword = "pancake", the output should be  
Ubachi(message, keyword) = "target acquired. Successful extermination ".

This example corresponds to the example given in the description above. Note, that it has a whitespace character at the end: that is because the encryption requires a message that can be written out in a rectangle.

**Input/Output**

* **[time limit] 3000ms (cs)**
* **[input] string message**

A message encrypted with *ubachi* encryption.

*Constraints:*  
keyword.length ≤ message.length ≤ 120.

* **[input] string keyword**

A keyword representing *ubachi* transposition key. It is guaranteed to contain only lowercase English letters.

*Constraints:*  
3 ≤ keyword.length ≤ message.length.

* **[output] string**

Decoded message.

<https://codefights.com/challenge/WbDRnJGc7TYLdtkcp?utm_source=featuredChallenge&utm_medium=email&utm_campaign=email_notification>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1

{

class Program

{

static int[] arrayPosiciones(string keyword)

{

List<int> posiciones = new List<int>();

string s = keyword;

//Console.WriteLine(s);

char[] sort = s.ToCharArray();

Array.Sort(sort);

//Console.WriteLine(new string(sort));

bool[] marcas = new bool[sort.Length];

//Dictionary<char, int> frec = new Dictionary<char, int>();

Dictionary<char, int> frec = sort.GroupBy(x => x)

.ToDictionary(x => x.Key, x => x.Count());

Dictionary<char, int> copia = (from x in frec

select x).ToDictionary(x => x.Key, x => x.Value);

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

{

int pos\_sort = Array.IndexOf(sort, s[i]);

int cant\_letra = frec[s[i]];

int cantidad\_original = copia[s[i]];

int res = cantidad\_original - cant\_letra;

frec[s[i]]--;

pos\_sort += res;

//Console.Write(pos\_sort + " ");

posiciones.Add(pos\_sort + 1);

}

return posiciones.ToArray();

}

static string pasoAtrasCortada(string cortada, int[] posiciones)

{

string concat = "";

for (int i = 0; i < posiciones.Length; i++)

{

concat += cortada[posiciones[i] - 1];

}

return concat;

}

static List<string> ArmarMatriz(string message, string keyword)

{

double d = (double)message.Length / (double)keyword.Length;

double filas = Math.Ceiling(d);

Console.WriteLine(filas);

char[,] matriz = new char[(int)filas, keyword.Length];

int index = 0;

for (int j = 0; j < matriz.GetLength(1); j++)

{

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

{

if (index < message.Length)

{

matriz[i, j] = message[index];

index++;

}

else

{

matriz[i, j] = ' ';

}

}

}

List<string> lista = new List<string>();

for (int i = 0; i < matriz.GetLength(0); i++)

{

string concat = "";

for (int j = 0; j < matriz.GetLength(1); j++)

{

concat += matriz[i, j];

// Console.Write(matriz[i, j] + "");

}

lista.Add(concat);

// Console.WriteLine();

}

return lista;

}

static string Ubachi(string message, string keyword)

{

int[] posiciones = arrayPosiciones(keyword);

List<string> lista\_dec = ArmarMatriz(message, keyword);

string mensaje\_anterior = "";

foreach (string elem in lista\_dec)

{

//nueva.Add(pasoAtrasCortada(elem, posiciones));

mensaje\_anterior += pasoAtrasCortada(elem, posiciones);

//Console.WriteLine(elem);

}

//Console.WriteLine(mensaje\_anterior);

List<string> nueva = ArmarMatriz(mensaje\_anterior, keyword);

string mens\_anterior2 = "";

foreach (string elem in nueva)

{

mens\_anterior2 += pasoAtrasCortada(elem, posiciones);

}

//Console.WriteLine(mens\_anterior2);

//foreach (string elem in nueva)

//{

// Console.WriteLine(elem);

//}

return mens\_anterior2;

}

static void Main(string[] args)

{

// 7, 1, 6, 3, 2, 5, 4

//string message = "cufintxo r esritqdeucltnagecse.ue raaiS mt";

//string keyword = "pancake";

//Ubachi(message, keyword);

//string cortada = "cxsea.a";

//int[] posiciones = { 7, 1, 6, 3, 2, 5, 4 };

//Console.WriteLine(pasoAtrasCortada(cortada, posiciones));

string keyword = "pancake";

//string msaje = "cufintxo r esritqdeucltnagecse.ue raaiS mt11";

//string decodificado = "target acquired. Successful extermination ";

string msaje = "cufintxo r esritqdeucltnagecse.ue raaiS mt";

//Console.WriteLine(msaje.Length);

//Console.WriteLine(decodificado.Length);

//List<string> lista = ArmarMatriz(msaje, keyword);

//foreach (string elem in lista)

//{

// Console.WriteLine(elem);

//}

Ubachi(msaje, keyword);

// PasoAtrasLista(msaje, keyword);

Console.ReadLine();

}

}

}