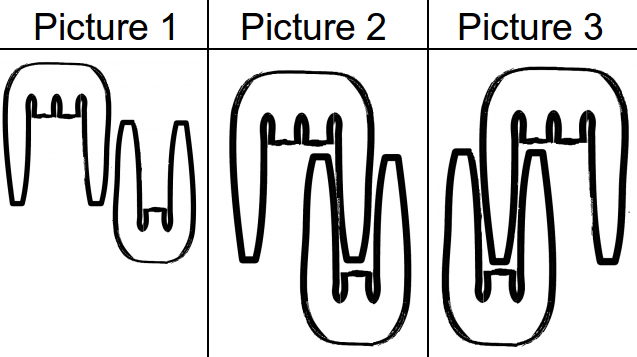
Miss X has only two combs in her possession, both of which are old and miss a tooth or two. She also has many purses of different length, in which she carries the combs. The only way they fit is horizontally and without overlapping. Given teeth' positions on both combs, find the minimum length of the purse she needs to take them with her.

It is guaranteed that there is at least one tooth at each end of the comb.  
It is also guaranteed that the total length of two strings is smaller than 32.  
Note, that the combs can **not** be rotated/reversed.

**Example**

For comb1 = "\*..\*" and comb2 = "\*.\*", the output should be  
combs(comb1, comb2) = 5.

Although it is possible to place the combs like on the first picture, the best way to do this is either picture 2 or picture 3.



**Input/Output**

* **[time limit] 3000ms (cs)**
* **[input] string comb1**

A comb is represented as a string. If there is an asterisk ('\*') in the ith position, there is a tooth there. Otherwise there is a dot ('.'), which means there is a missing tooth on the comb.

*Constraints:*  
1 ≤ comb1.length ≤ 10.

* **[input] string comb2**

The second comb is represented in the same way as the first one.

*Constraints:*  
1 ≤ comb2.length ≤ 10.

* **[output] integer**

The minimum length of a purse Miss X needs.

<https://codefights.com/arcade/code-arcade/spring-of-integration/6ceKutpnCs4LzBKgf>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1

{

class Program

{

static int combs(string comb1, string comb2)

{

string c1 = comb1;

string c2 = comb2;

if (c2.Length <= c1.Length)

{

c1 = comb2;

c2 = comb1;

}

int izq = 0, der = comb1.Length + comb2.Length;

int minLen = comb1.Length + comb2.Length;

while (der >= 0)//mientras haya ceros a la derecha

{

/\*

\*completo con ceros a la izquierda y ceros a la derecha

\*en ambas copias de comb1 y de comb2 para hacer las comparaciones

\*/

string cerosIzqC1 = new string('0', comb2.Length);

string cerosDerC1 = new string('0', comb2.Length);

c1 = cerosIzqC1 + comb1 + cerosDerC1;

string cerosIzq = new string('0', izq);

string cerosDer = new string('0', der);

c2 = cerosIzq + comb2 + cerosDer;

Console.WriteLine(c1);

Console.WriteLine(c2);

Console.WriteLine();

bool encajanTodos = true;

for (int i = 0; i < c1.Length; i++)

{

if (c1[i] != '0' && c2[i] != '0')

{

/\*si los dos no son cero, (que estan puestos para comparar)

entonces si son dienten en ambos es porque no encajan,

pueden ser en ambos '.' o sea puede haber

\* en c1 un '.' y en c2 un '.' (espacios sin dientes)\*/

if (c1[i] == '\*' && c2[i] == '\*')

{

encajanTodos = false;

break;

}

}

}

if (encajanTodos)

{

int primerC1 = c1.IndexOf('\*');

int primerC2 = c2.IndexOf('\*');

int ultimoC1 = c1.LastIndexOf('\*');

int ultimoC2 = c2.LastIndexOf('\*');

/\*

busco el primer asterisco en ambos y busco el ultimo asterisco

\* en ambos (que son los bordes de los peines)

\* entonces resto al ultimo el primero, porque

\* si hago c1.length o c2.length me estaria contando

\* los ceros que completé para comparar

\*/

minLen = Math.Min(minLen, Math.Max(ultimoC1, ultimoC2)- Math.Min(primerC1, primerC2)+1);

}

Console.WriteLine("Min len: " + minLen);

der--;

izq++;

}

return minLen;

}

static void Main(string[] args)

{

//string s = "\*..\*";

//string s = "\*....\*\*\*.\*";

//foreach (string elem in contarPartes(s))

//{

// Console.WriteLine(elem);

//}

//string comb1 = "\*..\*" ;

//string comb2 = "\*.\*";

//string comb1 = "1234567";

//string comb2 = "123";

//combs(comb1, comb2);

//string comb1 = "123";

//string comb2 = "1234567";

//combs(comb1, comb2);

//string comb1 = "123";

//string comb2 = "123";

//combs(comb1, comb2);

//string comb1 = "\*..\*";

//string comb2 = "\*.\*";

//Console.WriteLine(combs(comb1, comb2));

//string comb1 = "\*..\*.\*";

//string comb2 = "\*.\*\*\*";

//Console.WriteLine(combs(comb1, comb2));

string comb1= "\*...\*";

string comb2 = "\*.\*";

Console.WriteLine(combs(comb1, comb2));

Console.ReadLine();

}

}

}