Alex often finds words that have letters both in upper and lower case, and they make him feel terrible. He's so tired of this that he decides to convert all these words to one letter case.

To choose the case, he calculates the number of lower case letters and the number of upper case letters. If these numbers have the same [parity](keyword://parity), Alex converts the word to lower case. Otherwise, he converts the word to upper case. Note that Alex doesn't convert words that only have letters that are one case.

Given the word that Alex wants to convert, return the modified word.

**Example**

* For word = "KeY", the output should be  
  convertToOneCase(word) = "KEY".

This word contains 2 upper case letters and 1 lower case letter. 2 and 1 have the opposite parity, so this word should be converted to upper case.

* For word = "FOObar", the output should be  
  convertToOneCase(word) = "foobar".

This word contains 3 letters in both upper and lower case. 3 and 3 have the same parity, so this word should be converted to lower case.

* For word = "chamomile", the output should be  
  convertToOneCase(word) = "chamomile".

The letters in this word are only in one case, so Alex doesn't convert it.

**Input/Output**

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

The word that Alex wants to convert. It contains only English letters.

*Guaranteed constraints:*  
1 ≤ word.length ≤ 1000.

* **[output] string**

The modified word.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication3

{

class Program

{

string convertToOneCase(string word)

{

int mayus = 0, minus = 0;

foreach (char ch in word)

{

if (char.IsUpper(ch))

{

mayus++;

}

else if (char.IsLower(ch))

{

minus++;

}

}

if (mayus != 0 && minus != 0)

{

if (mayus % 2 != 0 && minus % 2 != 0)

{

return word.ToLower();

}

else if (mayus % 2 == 0 && minus % 2 == 0)

{

return word.ToLower();

}

else if (mayus % 2 == 0 || minus % 2 == 0)

{

return word.ToUpper();

}

}

return word;

}

The *PreSuffix* of two numbers a and b is a number that is the prefix of one of the given numbers and the suffix of the other number. Both numbers are in decimal notation.

Given two integers, a and b, return their maximum *PreSuffix*, or -1 if they don't have any *PreSuffixes*.

**Example**

* For a = 123456 and b = 456123, the output should be  
  maxPreSuffix(a, b) = 456.

In this case, a and b have two *PreSuffixes*, 123 and 456. 456 is the maximum *PreSuffix*.

* For a = 123 and b = 456, the output should be  
  maxPreSuffix(a, b) = -1.

These two numbers don't have any *PreSuffixes*.

**Input/Output**

* **[time limit] 3000ms (cs)**
* **[input] integer a**

*Guaranteed constraints:*  
1 ≤ a ≤ 109.

* **[input] integer b**

*Guaranteed constraints:*  
1 ≤ b ≤ 109.

* **[output] integer**

The maximum *PreSuffix* of a and b, or -1if a *PreSuffix* doesn't exist.

static int maxPreSuffix(int a, int b)

{

string x = a.ToString();

string y = b.ToString();

if (x.EndsWith(y))

{

return int.Parse(y);

}

if (y.EndsWith(x))

{

return int.Parse(x);

}

if (a == b) return a;

int max = -1;

for (int i = x.Length-1; i> 0; i--)

{

string subs1 = x.Substring(0, i);

string subs2 = x.Substring(i, x.Length - i);

// Console.WriteLine(subs);

if (y.EndsWith(subs1))

{

max = Math.Max(max, int.Parse(subs1));

}

else if (y.EndsWith(subs2))

{

max = Math.Max(max, int.Parse(subs2));

}

//if (subs1.Length == 0) return -1;

}

for (int i = y.Length - 1; i > 0; i--)

{

string subs1 = y.Substring(0, i);

string subs2 = y.Substring(i, y.Length - i);

// Console.WriteLine(subs);

if (x.EndsWith(subs1))

{

max = Math.Max(max, int.Parse(subs1));

}

else if (x.EndsWith(subs2))

{

max = Math.Max(max, int.Parse(subs2));

}

//if (subs1.Length == 0) return -1;

}

return max;

}

The inhabitants of the country Murmuria want to choose a new capital city. To calculate which cities can become the capital, they draw a map of the country. The map is an n × m matrix in which some cells are marked as cities. A city can become the capital if, in this map, it has at least one city in the same row and at least one city in the same column except itself.

Given a (0, 1) matrix country where country[i][j] = 1 indicates a city, return the number of all the possible capitals in Murmuria.

**Example**

For

country = [[0, 0, 1, 0, 1],

[1, 0, 0, 0, 0],

[1, 1, 1, 1, 1],

[0, 0, 0, 1, 0],

[1, 0, 1, 1, 0]]

the output should be  
possibleCapitals(country) = 9.

There are 12 cities on this map, and only 3 of them don't meet the requirements needed to be the capital: the city in the second row, the city in the fourth row, and city in the second column.

**Input/Output**

* **[time limit] 3000ms (cs)**
* **[input] array.array.integer country**

The map of the country. country[i][j] = 1if there is a city at that location and 0otherwise.

*Guaranteed constraints:*  
1 ≤ country.length ≤ 100,  
1 ≤ country[i].length ≤ 100,  
country[i].length = country[j].length,  
0 ≤ country[i][j] ≤ 1.

* **[output] integer**

The number of possible capital cities in this country.

//---solucion por THUYTRANG12A2

static int possibleCapitals(int[][] country)

{

int row = country.Length;

int column = country[0].Length;

int[] rCount = new int[row];

int[] cCount = new int[column];

int ans = 0;

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

{

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

{

if (country[i][j] == 1)

{

rCount[i]++;

cCount[j]++;

}

}

}

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

{

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

{

if (country[i][j] == 1 && rCount[i] > 1 && cCount[j] > 1)

{

ans++;

}

}

}

return ans;

}

static void Main(string[] args)

{

int[][] country = { new int[]{ 0,0,1,0,1},

new int[] {1,0,0,0,0},

new int[] {1,1,1,1,1},

new int[] {0,0,0,1,0},

new int[] {1,0,1,1,0}};

Console.WriteLine(possibleCapitals(country));

Console.ReadLine();

}

}

}

You have three arrays of integers: a, b, and c. Return the number of all triplets i, j, and k such that a[i]b[j] = c[k], where a[i]b[j] is the concatenation of two numbers in decimal notation.

**Example**

* For a = [1, 2, 3, 4], b = [5, 6, 7, 8] and c = [17, 25, 83, 27], the output should be  
  numberOfConcatenations(a, b, c) = 3.

Here all the possible concatenations:

* + a[0]b[2] = 17 = c[0];
  + a[1]b[0] = 25 = c[1];
  + a[1]b[2] = 27 = c[3].
* For a = [1, 1, 1], b = [1] and c = [11], the output should be  
  numberOfConcatenations(a, b, c) = 3.

You can concatenate every number from a with b[0] to get c[0].

**Input/Output**

* **[time limit] 3000ms (cs)**
* **[input] array.integer a**

1 ≤ a.length ≤ 105,  
1 ≤ a[i] ≤ 1000.

* **[input] array.integer b**

*Guaranteed constraints:*  
1 ≤ b.length ≤ 105,  
1 ≤ b[i] ≤ 1000.

* **[input] array.integer c**

*Guaranteed constraints:*  
1 ≤ c.length ≤ 105,  
1 ≤ c[i] ≤ 106.

* **[output] integer64**

The number of all triplets i, j, and k such that a[i]b[j] = c[k].

--------TIME LIMIT EXCEEDED---------

static long numberOfConcatenations(int[] a, int[] b, int[] c)

{

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

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

{

for (int j = 0; j < b.Length; j++)

{

int conc = int.Parse(a[i].ToString() + b[j].ToString());

concat.Add(conc);

}

}

long cont = 0;

foreach (int elem in c)

{

cont += concat.Count(e => e == elem);

//cont++;

}

return cont;

}

----------MI SOLUCION ACEPTADO-------------

static long numberOfConcatenations(int[] a, int[] b, int[] c)

{

Dictionary<string, int> frec\_a = a.GroupBy(x => x.ToString())

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

Dictionary<string, int> frec\_b = b.GroupBy(x => x.ToString())

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

long cont = 0;

foreach (int elem in c)

{

string s = elem.ToString();

for (int i = 1; i < s.Length; i++)

{

string sub1 = s.Substring(0, i);

string sub2 = s.Substring(i, s.Length - i);

// Console.WriteLine(sub1 + " " + sub2);

//int s1 = int.Parse(sub1);

//int s2 = int.Parse(sub2);

if (frec\_a.ContainsKey(sub1) && frec\_b.ContainsKey(sub2))

{

cont += (long)frec\_a[sub1] \* (long)frec\_b[sub2];

}

}

}

return cont;

}

---------Solucion por maxim\_k9------

long long numberOfConcatenations(std::vector<int> a, std::vector<int> b, std::vector<int> c) {

unordered\_map<int, int> c\_map, a\_map, b\_map;

for (int v : c) {

c\_map[v]++;

}

for (int v : a) {

a\_map[v]++;

}

for (int v : b) {

b\_map[v]++;

}

long long result = 0;

for (auto& a\_it : a\_map) {

for (auto& b\_it : b\_map) {

int v = stoi(to\_string(a\_it.first) + to\_string(b\_it.first)); //va a dar cero en los que no estén en c\_map[v] , porque v es la composición entre a y b

result += (long long)c\_map[v] \* a\_it.second \* b\_it.second;

}

}

return result;

}