It's your lucky day: you just landed a role on the hit TV show "The Walking Dead"!

In your first episode, the scene is pretty stressful already: you've found a serum that will make you immune to zombies, but it's inside a locked safe (and the zombies are scratching outside the door!). First you try to shoot out the lock mechanism with your gun, to no avail. Suddenly you notice a message scrawled on the side of the safe: "the password is a perfect square or a perfect cube". There is also a number on this note.

Determine the number of ways you can rearrange the digits of the number to make a correct password. If the given number is already a perfect square or a perfect cube, include it in your answer as well. In your answer, also include correct passwords with leading zeroes.

**Example**

* For number = 414, the output should be  
  perfectSquareOrCube(number) = 2.

You can get numbers 144, which is a perfect square of 12, and 441, which is a perfect square of 21.

* For number = 64, the output should be  
  perfectSquareOrCube(number) = 1.

The number 64 is a perfect square of 8 and a perfect cube of 4.

* For number = 71, the output should be  
  perfectSquareOrCube(number) = 0.

71 and 17 aren't perfect squares or perfect cubes.

**Input/Output**

* **[execution time limit] 3 seconds (cs)**
* **[input] integer number**

An integer, the number on the note.

*Guaranteed constraints:*  
1 ≤ number ≤ 900.

* **[output] integer**

The number of ways to rearrange the digits of the number to get a perfect square or a perfect cube.

**[C#] Syntax Tips**

// Prints help message to the console

// Returns a string

**string** **helloWorld**(**string** name) {

Console.Write("This prints to the console when you Run Tests");

**return** "Hello, " + name;

}

<https://codefights.com/challenge/jtbjBgG7Fhhn7S4Ac/solutions>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

static bool nextPermutation(char[] array)

{

// Find non-increasing suffix

int i = array.Length - 1;

while (i > 0 && array[i - 1] >= array[i])

i--;

if (i <= 0)

return false;

// Find successor to pivot

int j = array.Length - 1;

while (array[j] <= array[i - 1])

j--;

char temp = array[i - 1];

array[i - 1] = array[j];

array[j] = temp;

// Reverse suffix

j = array.Length - 1;

while (i < j)

{

temp = array[i];

array[i] = array[j];

array[j] = temp;

i++;

j--;

}

return true;

}

static int perfectSquareOrCube(int number)

{

char[] ns = number.ToString().ToCharArray();

Array.Sort(ns);

int cont = 0;

do

{

int cuad = (int)Math.Sqrt(int.Parse(new string(ns)));

int cubo = (int) Math.Ceiling( Math.Pow(int.Parse(new string(ns)), (double)(1.0 / 3.0)));

if (cuad \* cuad == int.Parse(new string(ns)) ||

cubo \* cubo \* cubo == int.Parse(new string(ns)))

{

cont++;

}

Console.WriteLine(new string(ns));

} while (nextPermutation(ns));

return cont;

}

static void Main(string[] args)

{

//int number = 4444444;

int number = 521;

Console.WriteLine(perfectSquareOrCube(number));

//Console.WriteLine( (int) Math.Pow(125, 1.0 / 3.0));

Console.ReadLine();

}

}

}