Mid-exam Imperative Programming (5-10-2015)

- You can solve the problems in any order. The first problem is worth one grade point, the remaining four problems are worth two grade points each. You get one grade point for free.
- Your assessment is based solely on the judgment given by Justitia. If Justitia accepts your program, then the problem is considered 'solved'. There will be no assessment of programming style. However, (very) inefficient programs may be rejected. In such cases, the error will be 'time limit exceeded'. The limit for each problem is one second.
- Note the hints that Justitia gives when your program fails a test.
- Needless to say: you are not allowed to work together. If plagiarism is detected, both parties (supplier of the code and the person that sends in a copy) will be excluded from any further participation in the course. You are not allowed to use mail, phones, tablets, etc.
- For each problem, there are three examples of inputs and associated expected outputs. The input files of these examples can be found in Justitia, together with the corresponding output files. These files are called `1.in`, `2.in`, `3.in` and `1.out`, `2.out` and `3.out`. These files can be used to test whether the output of your program matches the requested layout, so that there can be no misunderstanding about the layout and spaces in the output.

Problem 1: multiples

The input of this problem consists of three integers `a`, `b`, and `n` such that $1 \leq \texttt{a} \leq \texttt{b} \leq \texttt{n} \leq 1000000$. The output should be the number of integers `x` such that $1 \leq \texttt{x} \leq \texttt{n}$ and `x` is a multiple of both `a` and `b`. Note that there are no spaces in the output, and that the output ends with a newline (`\n`):

Example 1:

input:
```
1 1 10
```
output:
```
10
```

Example 2:

input:
```
2 3 10
```
output:
```
1
```

Example 3:

input:
```
2 50 1000000
```
output:
```
20000
```

Problem 2: squaring digits

Given a non-negative integer n, we can create a number chain from it as follows. We add the squares of the digits of the number to form a new number, and repeat this process until we arrive at a number that has been encountered before. Two examples are:

$44 \to 32 \to 13 \to 10 \to 1 \to 1$ and $85 \to 89 \to 145 \to 42 \to 20 \to 4 \to 16 \to 37 \to 58 \to 89$

An amazing property (which you may use, you do not have to prove it) is that every starting number will eventually arrive at 1 or 89.
The input for this problem consists of an integer `n` such that $1 \leq \texttt{n} \leq 1000000 = 10^6$. The output of the program should be the number of starting numbers `x` (where $1 \leq \texttt{x} \leq \texttt{n}$) such that the corresponding number chain terminates with 89.

Example 1:

input:
```
10
```
output:
```
7
```

Example 2:

input:
```
100
```
output:
```
80
```

Example 3:

input:
```
1000000
```
output:
```
856929
```

Problem 3: peasant multiplication

An ancient method to multiply numbers is called *peasant multiplication*. To show how the method works, consider the example in which we want to compute the multiplication of `x` and `y` where `x=73` and `y=67`. A table consisting of two columns is constructed. The first row of the table consists of the numbers `x` and `y`. Any other row of the table is constructed from the row that precedes it: the first number is doubled, while the second is halved. In this halving process, the remainders of odd numbers are ignored (e.g. halving 67 yields 33). This process stops as soon as the second number has reached the value 0. From the table, the multiplication of `x` and `y` can be obtained by summing the numbers in the first column of all rows of which the number in the second column is odd. For the given example, the table would look like:

```
  73      67
 146      33
 292      16
 584       8
1168       4
2336       2
4672       1
9344       0
```

The conclusion is that `73*67=73+146+4672=4891`. Write a program that reads from the input two non-negative numbers `x` and `y` and outputs the product of `x` and `y` as a sum of values that are encountered by the peasant multiplication method. You may assume that the input numbers are both less than 1000. Make sure that the format of your output is in accordance with the following examples:

Example 1:

input:
```
73 67
```
output:
```
73*67=73+146+4672=4891
```

Example 2:

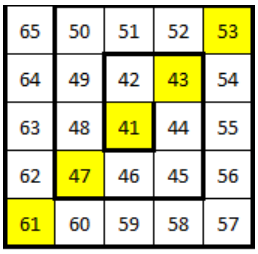
input:
```
0 100
```
output:
```
0*100=0
```

Example 3:

input:
```
12 34
```
output:
```
12*34=24+384=408
```

Problem 4: diagonal spiral sum

The input of this problem consists of two positive integers `n` and `m`. The number `n` is odd, and defines the size of an `n×n` grid of cells. The center cell is filled with the number `m`. The remaining cells are filled in a spiral fashion with the numbers `m+1`, `m+2`, ..., `m+n*n-1`.
For example, the input `5 41` represents the following grid:



65	50	51	52	53
64	49	42	43	54
63	48	41	44	55
62	47	46	45	56
61	60	59	58	57

The output of the program should be the sum of the numbers in the ascending diagonal of the grid (see figure). In this example, the right answer would be 245 (the sum 61+47+41+43+53).

Example 1:
input:
```
5 41
```
output:
```
245
```

Example 2:
input:
```
3 1
```
output:
```
11
```

Example 3:
input:
```
3 0
```
output:
```
8
```

Problem 5: your friend is my friend

In this problem we consider `n` persons (you may assume that $1 \leq \mathtt{n} \leq 30$). The persons are numbered $0, 1, .., \mathtt{n}-1$. Now you may assume the natural rule "*if* `a` *is a friend of* `b`*, then* `b` *is a friend of* `a`". Moreover, everyone is quite sociable in the sense that "*if* `x` *and* `y` *are friends, and* `y` *and* `z` *are friends, then* `x` *and* `z` *are also friends*".
As an example, consider the case in which `n=8`: 0 and 1 are friends, 2 and 4 are friends, 4 and 1 are friends, and 3 and 6 are friends. As a result of the above rule, there are four groups of friends: {0,1,2,4}, {3,6} and the two 'loner' groups {5} and {7}.

The first input line of this problem consist of a number `n` (the number of persons) and a positive integer `m` (the number of direct friendship pairs). The rest of the input consists of `m` pairs of numbers, designating direct friendships. The output of your program should be the number of groups.

Example 1:
input:
```
8 4
0 1
2 4
4 1
3 6
```
output:
```
4
```

Example 2:
input:
```
7 4
1 0
4 2
1 4
6 3
```
output:
```
3
```

Example 3:
input:
```
5 4
0 1
4 3
1 2
2 3
```
output:
```
1
```