GT Spring Practice, Feb 23, 2012

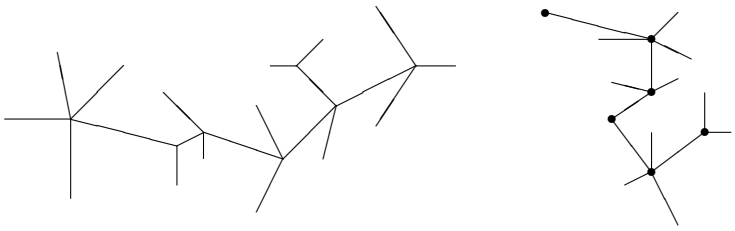
A. Caterpillar
B. Triangular Sums
C. Baskets of Gold Coins
D. Permutation Recovery
E. Q
F. Visible Lattice Points
G. Rounders

Notes:
1. Please name your Java class names A, B, C, D, E, F, G.
2. Input from standard in (System.in) and output to standard out (System.out)
3. Java heap size is 1GB and stack size is 8MB
The command: "java -Xmx1024m -Xss8m PROGRAM_NAME"

Problem A: Caterpillar

An undirected graph is called a *caterpillar* if it is connected, has no cycles, and there is a path in the graph where every node is either on this path or a neighbor of a node on the path. This path is called the *spine* of the caterpillar and the spine may not be unique. You are simply going to check graphs to see if they are caterpillars.

For example, the left graph below is not a caterpillar, but the right graph is. One possible spine is shown by dots.



Input

There will be multiple test cases. Each test case starts with a line containing n indicating the number of nodes, numbered 1 through n (a value of $n = 0$ indicates end-of-input). The next line will contain an integer e indicating the number of edges. Starting on the following line will be e pairs n_1 n_2 indicating an undirected edge between nodes n_1 and n_2. This information may span multiple lines. You may assume that $n \leq 100$ and $e \leq 300$. Do not assume that the graphs in the test cases are connected or acyclic.

Output

For each test case generate one line of output. This line should either be

Graph g is a caterpillar.

or

Graph g is not a caterpillar.

as appropriate, where g is the number of the graph, starting at 1.

Sample Input

```
22
21
1 2 2 3 2 4 2 5 2 6 6 7 6 10 10 8 9 10 10 12 11 12 12 13 12 17
18 17 15 17 15 14 16 15 17 20 20 21 20 22 20 19
16
15
1 2 2 3 5 2 4 2 2 6 6 7 6 8 6 9 9 10 10 12 10 11 10 14 10 13 13 16 13 15
0
```

Sample Output

```
Graph 1 is not a caterpillar.
Graph 2 is a caterpillar.
```

B • Triangular Sums

The **n**th *Triangular* number, `T(n) = 1 + ... + n`, is the sum of the first **n** integers. It is the number of points in a triangular array with **n** points on side. For example `T(4)`:

```
X
X X
X X X
X X X X
```

Write a program to compute the weighted sum of triangular numbers:

W(n) = `SUM[k = 1..n; k*T(k+1)]`

Input

The first line of input contains a single integer **N**, (1 ≤ **N** ≤ 1000) which is the number of datasets that follow.

Each dataset consists of a single line of input containing a single integer **n**, (1 ≤ **n** ≤300), which is the number of points on a side of the triangle.

Output

For each dataset, output on a single line the dataset number, (1 through **N**), a blank, the value of **n** for the dataset, a blank, and the weighted sum , **W(n)**, of triangular numbers for **n**.

Sample Input	Sample Output
4 3 4 5 10	1 3 45 2 4 105 3 5 210 4 10 2145

Baskets of Gold Coins

Input: `coins.in`
Output: `coins.out`

You are given N baskets of gold coins. The baskets are numbered from 1 to N. In all except one of the baskets, each gold coin weighs w grams. In the one exceptional basket, each gold coin weighs $w–d$ grams. A wizard appears on the scene and takes 1 coin from Basket 1, 2 coins from Basket 2, and so on, up to and including $N–1$ coins from Basket $N–1$. He does not take any coins from Basket N. He weighs the selected coins and concludes which of the N baskets contains the lighter coins. Your mission is to emulate the wizard's computation.

Input

The input file will consist of one or more lines; each line will contain data for one instance of the problem. More specifically, each line will contain four positive integers, separated by one blank space. The first three integers are, respectively, the numbers N, w, and d, as described above. The fourth integer is the result of weighing the selected coins.

N will be at least 2 and not more than 8000. The value of w will be at most 30. The value of d will be less than w.

Output

For each instance of the problem, your program will produce one line of output, consisting of one positive integer: the number of the basket that contains lighter coins than the other baskets.

Sample input

```
10 25 8 1109
10 25 8 1045
8000 30 12 959879400
```

Output for sample input

```
2
10
50
```

Permutation Recovery

Input: `permutation.in`
Output: `permutation.out`

Professor Permula gave a number of permutations of the n integers 1, 2, ..., n to her students. For each integer i ($1 <= i <= n$), she asks the students to write down the number of integers greater than i that appears before i in the given permutation. This number is denoted a_i. For example, if $n = 8$ and the permutation is 2,7,3,5,4,1,8,6, then $a_1 = 5$ because there are 5 numbers (2, 7, 3, 5, 4) greater than 1 appearing before it. Similarly, $a_4 = 2$ because there are 2 numbers (7, 5) greater than 4 appearing before it.

John, one of the students in the class, is studying for the final exams now. He found out that he has lost the assignment questions. He only has the answers (the a_i's) but not the original permutation. Can you help him determine the original permutation, so he can review how to obtain the answers?

Input

The input consists of a number of test cases. Each test case starts with a line containing the integer n ($n <= 500$). The next n lines give the values of a_1, ..., a_n. The input ends with $n = 0$.

Output

For each test case, print a line specifying the original permutation. Adjacent elements of a permutation should be separated by a comma. Note that some cases may require you to print lines containing more than 80 characters.

Sample input

```
8
5
0
1
2
1
2
0
0
10
9
8
7
6
5
4
3
2
1
0
0
```

Output for sample input

```
2,7,3,5,4,1,8,6
10,9,8,7,6,5,4,3,2,1
```

2006 South Central USA Regional Programming Contest

Q

Introduction

You've got a queue. And you just got to mess with it.
Given a queue of items and a series of queue operations, return the resulting queue.
Queue operations are defined as follows:

starting-position to *requested-position*

meaning one wants the item at the starting position to be moved to the requested position. So if the queue of items were:

```
Item1 Item2 Item3 Item4 Item5
```

(Item1 being in position 1, Item2 in position 2, etc.)

after applying the queue operation:

5 to 2

the resulting queue would be:

```
Item1 Item5 Item2 Item3 Item4
```

as Item5 (the item in position 5) was moved to position 2. Multiple queue operations are applied at the same time, however; e.g., given the queue of items:

```
Item1 Item2 Item3 Item4 Item5 Item6 Item7 Item8
```

If the following queue operations were applied:

```
2 to 6; 6 to 3; 4 to 5; 5 to 2; 7 to 4; 8 to 1
```

then the resulting queue would be:

```
Item8 Item5 Item6 Item7 Item4 Item2 Item1 Item3
```

As you can see, the queue operations are strictly enforced, with other items (not involved in queue operations) maintaining their order and moving to vacant positions in the queue. Note that no two queue operations will have the same *starting-position* or same *requested-position* defined.

Input

Input to this problem will begin with a line containing a single integer *x* indicating the number of datasets. Each data set consists of three components:

1. Start line – A single line, "*m n*" (1 <= *m, n* <= 20) where *m* indicates the number of items in the queue and *n* indicates the number of queue operations.
2. Queue items – A line of short (between 1 and 8 characters) alphanumeric names for the items in the queue. Names are unique for a given data set and contain no whitespace.
3. Queue operations – *n* lines of queue operations in the format "*`starting-position requested-position`*".

Output

For each dataset, output the queue after the queue operations have been applied. Print the elements of the queue on a single line, starting from the first and ending with the last, with a single space separating each item.

Sample Input

```
3
5 1
alpha beta gamma delta epsilon
5 2
8 6
a b c d e f g h
2 6
6 3
4 5
5 2
7 4
8 1
3 2
foo bar baz
3 1
1 3
```

Sample Output

```
alpha epsilon beta gamma delta
h e f g d b a c
baz bar foo
```

F • Visible Lattice Points

A lattice point (***x***, ***y***) in the first quadrant (***x*** and ***y*** are integers greater than or equal to 0), other than the origin, is *visible* from the origin if the line from (0, 0) to (***x***, ***y***) does not pass through any other lattice point. For example, the point (4, 2) is not visible since the line from the origin passes through (2, 1). The figure below shows the points (***x***, ***y***) with $0 \le x, y \le 5$ with lines from the origin to the visible points.

Write a program which, given a value for the size, **N**, computes the number of visible points (***x***,***y***) with $0 \le x, y \le N$.

Input

The first line of input contains a single integer **C**, ($1 \le C \le 1000$) which is the number of datasets that follow.

Each dataset consists of a single line of input containing a single integer **N**, ($1 \le N \le 1000$), which is the size.

Output

For each dataset, there is to be one line of output consisting of: the dataset number starting at 1, a single space, the size, a single space and the number of visible points for that size.

Sample Input	Sample Output
4 2 4 5 231	1 2 5 2 4 13 3 5 21 4 231 32549

2006 South Central USA Regional Programming Contest

Rounders

Introduction:

For a given number, if greater than ten, round it to the nearest ten, then (if that result is greater than 100) take the result and round it to the nearest hundred, then (if that result is greater than 1000) take that number and round it to the nearest thousand, and so on ...

Input:

Input to this problem will begin with a line containing a single integer *n* indicating the number of integers to round. The next *n* lines each contain a single integer *x* (0 <= *x* <= 99999999).

Output:

For each integer in the input, display the rounded integer on its own line.

Note: Round up on fives.

Sample Input:

```
9
15
14
4
5
99
12345678
44444445
1445
446
```

Sample Output:

```
20
10
4
5
100
10000000
50000000
2000
500
```