Electronic Contest

11th BME International 24-hour Programming Contest

http://ch24.org

Electronic Contest

Welcome to the qualifying round of the 11th BME International 24-hour Programming Contest!

This document is the problem set for the Electronic Contest to be held on February 19th, 2011.

Rules

The Electronic Contest contains six problems. You have all the time in the world to solve them, but we take submissions from 10:00 to 15:00 CET. The inputs of the problems can be found in a zip file that you have probably already downloaded from the website. Each problem will have exactly 10 test cases.

You can use any platform or programming language to solve the problems. We are interested only in the output files, you don't need to upload the source code of the programs that solved them. Once you are done, you can upload your output files via the submission site: http://ch24.org/sub. Your solutions will be evaluated on-line.

There are two major problem types:

- Non-scaled problems: problems that have an exact solution. When submissions to these are evaluated, a final score is given immediately. From one team, only one correct submission will be accepted for each input (since the input is either solved or not). A, B, C, D and F are non-scaled problems.
- Scaled problems: problems that do not have a known "best" solution. Outputs for these problems compete against each other, and scores are scaled according to the best uploaded output. A team may submit multiple correct submissions to one input (only the latest submission will be taken into consideration). Only problem E is a scaled problem.

Note that points are awarded per output file and not per problem. If your solution only works for some of the input files, you will still be awarded points for the correct output files. A single output file however is either correct or wrong - partially correct output files are not worth any points.

Additional information for non-scaled problems:

Be quick about uploading the output files, because the scores awarded for every output file decrease with time. Uploading it just before the end of the contest is worth **70%** of the maximum points achievable for the test case. During the contest its value decreases linearly with time. However you should also be careful with uploading solutions. Uploading an incorrect solution is worth **-5** points. This penalty is additive, if you upload more incorrect solutions, you will receive it multiple times. For some problems, we distinguish format errors (unparsable outputs) from incorrect outputs, and the former will not be penalised.

Please note that there is no point in uploading another solution for an already solved testcase because you cannot achieve more points with it. Therefore the system will not register additional uploads for solved testcases.

Additional information for scaled problems:

In this case there will be no penalty for uploading a solution later, so you are able to achieve the maximum amount of points by submitting in the very last minute - if you beat the other teams' solutions, that is.

Scores to scaled problems are recalculated occasionally (every few minutes). Your points may decrease in time (when another team submits a better solution than yours).

Please be aware that only your last submission is considered - not your best one.

Good luck and have fun!

About the Submission site

The location of the submission site is:

http://ch24.org/sub

You will be able to log in to the submission site with your registered team name and password. After login you can access three main views:

Team Status

You can see your team's status here, with all your submissions and the points received for them.

Submit

This is where you can post your solution files. You can upload multiple output files for multiple problems with a single submit. The naming of the output files must strictly match the following format: X99.out - where X is the problem's character code followed by a number (1 or 2 digits) identifying the test case.

Scores

Here you can see the current standings of the contest. This will not be available in the last hour.

Contact

You should subscribe to the public mailing list at http://lists.ch24.org to receive announcements and to be able to send feedback. The address of the list is `ch24@ch24.org`.

During the contest we will be available on IRC on the `irc.ch24.org` server (using the default port, `6667`). For general discussion about the contest use the `#challenge24` channel, for problem specific discussion use `#a`, `#b`, `#c`, `#d`, `#e`, `#f` channels.

A. Equilateral

You are given N rods with sizes S1,S2,S3 ... SN. You can connect two rods with sizes Si and Sj to turn them into one rod with size Si+Sj.

You need to create the largest equilateral triangle possible out of the rods. So connect the rods in a way that you have 3 rods with exactly the same size.

You do not need to utilise all the rods.

Input

Each input will contain several test cases.

The first line of each test case will contain one number N indicating the amount of rods. The next line will contain N numbers (S1 ... SN) separated by spaces. The last line of the input will contain a 0.

Looking into the inputs you can verify that 1 <= Si <= 100 and 4 <= N <= 100.

Output

For each test case print one line with the size of the side of the biggest equilateral triangle that can be built out of the given rods. If no equilateral triangle can be built then print 0.

Example input

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

Example output

```
1
0
11
```

B. Spanning Tree

A university professor has distributed a graph with M nodes to her N students, and asked them to generate a spanning tree of the graph. To try to avoid cheating, she has reordered the nodes in each graph sent to students, and kept a mapping so she can decode the submissions later.

In the meanwhile, the secret service has become suspicious of certain students, and decided to capture the submissions. They found that some students cheated and copied the homeworks of other students (renumbering the nodes in their spanning trees as necessary).

Your task is to scan the submissions and find trees that are copies of each other.

Input

The first line of the input contains N and M. The following N lines each contains a tree described by A1 B1 A2 B2 ... A{M-1} B{M-1} numbers, where Ai - Bi is an edge (Ai and Bi are the indices of two nodes, nodes are indexed starting from zero).

Output

Each line of the output should contain the indices of trees which are copies of each other. The index of a tree without any copy should be alone on its line.

Trees are indexed from zero and the order of the indices in the output does not matter.

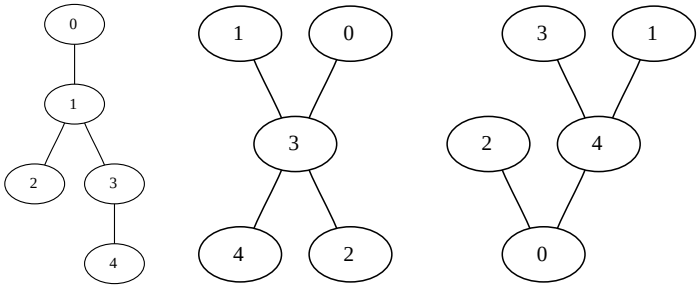
Example input

```
3 5
0 1 1 2 1 3 3 4
3 4 1 3 0 3 3 2
4 0 2 0 3 4 1 4
```

Example output

```
0 2
1
```

The three input trees are:



C. Gramophone

A secret agency got its hands on old classified records. They ask you to rescue the data from the disks.

These are 120rpm disks with few seconds of audio and are supposed to hold a sequence of decimal digits.

The track begins at the outer edge of the disks and the needle of the gramophone follows it towards inside.

Input

As the original medium is valuable you only get scanned images in png format.

Output

A sequence of digits. It is known that the last two digits are the same and give the sum of the previous digits modulo ten.

Example output

```
1 2 3 4 0 0
```

D. Uniq

Given N strings, count the number of different strings.

The strings will be generated by a pseudo random number generator, so you don't have to worry about downloading and parsing large text files.

A string is represented by a sequence of integers. The first integer is the number of characters in the string, each one of the following integers encode a character of the string. Such integer sequences can be concatenated to represent multiple strings.

The integer sequence representing the given N strings is generated by the following linear congruential generator:

```
X := SEED
A := 8433437992146984169
B := 7905438737954111703

function nextinteger():
        X := (A*X + B) mod (2^64)
        return X / (2^56)
```

where ^, / and mod are exponentiation, integer division and modulo operations with the usual semantics.

Input

A single line containing two integers: N and SEED.

Output

The number of different strings.

Example input

```
4 11413960417
```

Example output

```
3
```

Random numbers are 2, 245, 71, 1, 46, 4, 105, 177, 135, 114, 1, 46, 154, 195, ...
The lengths of the four strings are 2, 1, 4, 1.

E. Labeling

The task is to place city name labels on various zoom levels of a world map.

A list of cities with name, location and population is given in a tab separated file, `cities.txt`, the description of the columns is in the `cities.readme` file.

The label is written on the map using a 6x10 unit fixed width font, so it covers a rectangular area which height is 10 units and width is 6 units times the number of characters in the name of the city.

A label should be placed right next to its city and labels must not overlap.

The coordinates of a geographic location on the map is given by

```
x = longitude * SCALE
y = latitude * SCALE
```

where longitude and latitude are in degrees as in `cities.txt` and x, y are in the same units as the font of the labels.

The map wraps around horizontally so a label truncated on the right continues on the left side of the map. Vertically labels must not get truncated.

People are happy when they see the label of their city displayed on the map and you want to make as many people happy as possible.

Input

Single line containing the SCALE number.

Output

Each line of the output must contain three numbers separated by whitespace: `ID,X,Y`. Where `ID` is the id of a city (first column in `cities.txt`), and `X,Y` are the coordinates of the center of the label of the city (in map coordinate system).

Labels may appear in arbitrary order in the output and you don't have to label every city.

The location of a city must be on the boundary of its label and labels must not overlap, these constraints will be checked with 0.1 unit precision.

The map is cylindrical and the coordinate system wraps around horizontally: `X+360*SCALE,Y` is the same point as `X,Y`. When printing the coordinates of a label make sure that

```
-180 * SCALE <= X <= 180 * SCALE
-90 * SCALE + 5 <= Y <= 90 * SCALE - 5
```

Score

```
P = sum(population of labelled cities)
SCORE = 100 * (1 - sqrt(1 - P/MAXP))
```

where `MAXP` is the best `P` submitted so far.

Note that a rendered map with the best labeling will be published at http://ch24.org/ec/labeling during the contest.

cities.txt

Example lines from `cities.txt`

```
...
37051   HU      Budakeszi       47.51667        18.93333        13248
37052   HU      Budaors 47.46181        18.95845        25089
37053   HU      Budapest        47.49801        19.03991        1696128
37054   HU      Bugac   46.68889        19.67944        3066
37055   HU      Bugyi   47.21667        19.15   5412
...
```

Example input

```
2.0
```

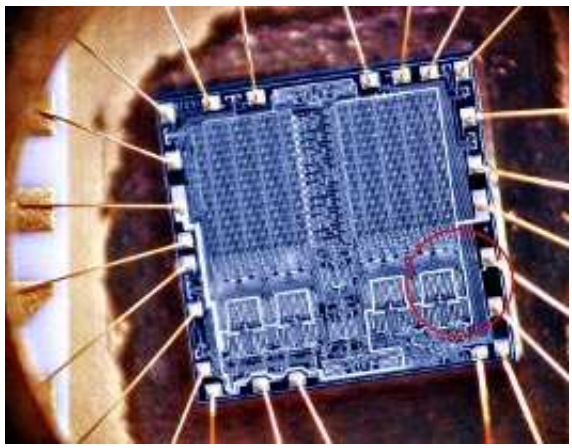
Example output

```
37053 62.07982 89.99602
```

In this case the city is at the top left corner of the label.

F. Allocator

Pear, manufacturer of the now popular jDesk portable music player, faces a problem: the cheap SRAM they purchased is all faulty, right before the great demo where the company is to present the new model to the potential customers. As a quick workaround, they want a special memory allocator to be designed, that allows the few common memory usage patterns to work on the device, so that at least those few programs they want to show on the demo would happen to work.



There are three types of RAM problems, each affecting random bits of the memory:

- bit stuck in 1 state
- bit stuck in 0 state
- bit is unreliable (returns random number for each read, unusable)

To isolate the problem, Pear mapped the memory use of those few common memory usage patterns they want to sell their devices with. These maps contain only the following instructions:

a n	allocate *n* bytes of memory
f IDX	free the memory allocated by the *IDXth* 'a' instruction
w IDX ofs sz str	write string from byte offset *ofs* of the allocation; number of bytes is *sz*
r IDX ofs sz	read *sz* long string from byte offset *ofs* of allocation *IDX*

String is given as a space-separated list of decimal numbers between 0 and 255; *ofs* and *sz* are always decimal numbers in bytes, >= 0. *sz* is always smaller than 1024.

Note: *IDX* is the reference of the allocation, the index of the "a" command in the dump; first instruction of the dump has *IDX*=0.

Programs written for Pear products are perfect - they never write/read beyond the allocation, and never read uninitizalized parts of the memory, never allocate <= 0 bytes, never allocate more than the amount of host memory at a given time, never do illegal free operations. They also always free all memory used by the end of the program execution.

The memory fault map is given as a sequence of the following structure, each in a new line:

```
address mask0 mask1 maskF
```

where address is a decimal address in the memory (byte offset), mask0 is the bitmask of bits stuck 0, mask1 is the bitmask of bits stuck 1, maskF is the bitmask of faulty (unusable bits).

Input

- size of the memory (in bytes), number of memory faults, number of dump entries
- memory fault map
- memory operation dump

Output

A linear list of allocation addresses in the order of allocation, 0 <= address < size of memory

Example input

```
26 2 7
21 5 8 18
25 1 2 12
a 6
a 10
w 0 2 3 112 177 221
w 1 0 4 112 6 6 6
r 0 2 2
r 1 0 3
f 0
f 1
```

Notes: first instruction is an allocation.

Example output

```
2
10
```