Preliminary Electronic Contest

13th International 24-hour Programming Contest

http://ch24.org

Preliminary Electronic Contest

Welcome to the testing round of the 13th International 24-hour Programming Contest!

This document is the problem set for the Preliminary Electronic Contest to be held on February 16th, 2013.

The PreEC provides a way for teams to familiarize themselves with our submission system and the general atmosphere of the competition. Whether teams participate in this testing round or not, or whatever results they achieve, will not have any consequences later in the competition.

The three problems we selected for the PreEC may not be the kind of problems you would consider especially challenging, however they provide some clues about the Electronic Contest - the required tools, the usage of our submission system and so on.

Rules

The Preliminary Electronic Contest contains three problems. You have all the time in the world to solve them, but we take submissions from 8:00 to 20:00 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://sub.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). In the PreEC, P and R 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). In the PreEC, only Q 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 for the non-scaled problems 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 for those tasks.

For some non-scaled problems, after submitting an incorrect solution, there may be a certain short delay (a couple of minutes) until you can re-submit an updated solution. The delay is applied per team per task per input, and is reported on the submission web interface.

Additional information for scaled problems:

In this case there will be no score 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. However, to avoid overloading our server, after submitting a correct solution, you may not re-submit an updated solution for a certain short delay (a couple of minutes). The delay is applied per team per task per input, and is reported on the submission web interface.

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://sub.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`), on the following channels:

- `#challenge24` for general discussion about the contest,
- `#info` for a full summary of announcements (read-only),
- `#p`, `#q`, `#r` for problem specific questions.

Note: **all relevant questions/answers will be copied to #info**, which is also available on the web (http://igor2.repo.hu/ch24/info.txt).

Prologue

Your team traveled to a far away country for a holiday.

Just after you cleared customs at the airport, some local citizens mistook you for modern world-famous painters, and carried you on their shoulders to the local Institute of Modern Art. It seems there’s no other way out of this situation than to act as painters.

P. Whistles

Modern painting is often performed in dark rooms (this is why so many modern painters fall asleep while working on their latest masterpiece).

Modern painting is also often an activity that painters enjoy in groups, but only up to a certain number of people. Hence the problem: it can get difficult to tell how many painters are already in a dark room - does the painter want to enter, or not?

The trivial and traditional solution to this problem is to give a whistle to each painter. Each painter will blow their whistle every once in a while, when no one else is whistling.

http://commons.wikimedia.org/wiki/File:Pfeifenschnur.jpg

To check whether a room is overcrowded or not, you only have to listen for a while and count how many different whistles you can hear.

Given a long wav file with a large number of beeps. Each beep has the same length and the gap between two beeps is the same (but this magic length constant may change between files). Beep frequencies are always rounded to the next 100 Hz.

Your task is to count how many different frequency beeps are in the file.

Input

Input is a wav file with unsigned 8 bit samples at 44100 Hz (the wav header is 44 bytes long).

Output

Output is a single integer, the number of different frequency beeps modulo 1000000007.

Example input

Example input is provided as 0.wav among the real input files. Please do not submit the solution for the example input.

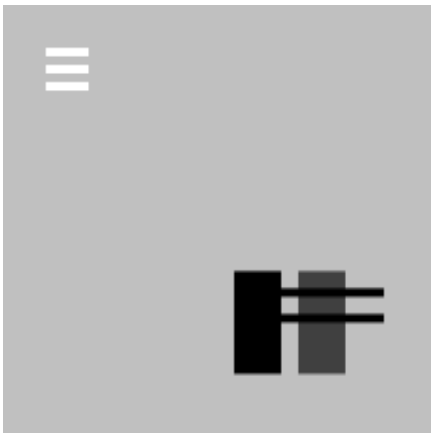
Example output

```
2
```

Q. Plot

The Institute of Modern Art graciously provided you with a set of brushes. Having unpacked and tried each of them, you realize you can't really paint. Fortunately you find a strange photoplotter in a dark corner (according to other artists also confined to the Institute, it's mainly used for typesetting the catalogues for exhibitions).

You quickly hook up the plotter to your computer and reverse-engineer the strange communication protocol. It seems the plotter runs some sort of interpreter and can draw squares (also known as Resolution Independent Pixels). You decide to dig up a few photos and try to program the device to copy them onto the canvas.



The albino. Drawn by the plotter

Your task is to draw a large grayscale picture similar to the input picture using a less-than-capable VM. Drawing primitives are squares given as x;y coordinates of their upper left corner and size. The image buffer is white at the beginning. There are two drawing modes: brush and erase. Brush increases the blackness of the image buffer at the given area by 1/4, erase decreases it by 1/4. Thus, it is possible to draw with 5 shades only (the possible pixel values on a 8-bit depth greyscale image: 255 (white), 192, 128, 64 and 0 (black)).

There is an upper limit on how many CPU instructions the plotter is willing to run - when this is reached, it simply stops interpreting. The plotter has very limited code memory as well, so only short code can be uploaded to it.

Constants:

Name	Value	Description
IMGW	16000	image width in pixels
IMGH	16000	image height in pixels
MAXINST	1000000	maximum number of evaluated instructions
MAXCODE	20000	maximum code size (memory slots)
MAXAREA	IMGW*IMGH*4	maximum number of pixels brushed or erased
NUMREGS	1024	number of registers

The interpreter

Registers

Registers are numbered from R0 to R(NUMREGS-1). The first few registers have special function and alias:

R0 is PC	program counter
R1 is the Z flag	1 means result of the last operation was zero (else it's 0)
R2 is the O flag	1 means last operation caused overflow (else it's 0)

Each register holds a signed integer between -2147483648 and 2147483647 inclusive. There is no dedicated stack or other accessible memory. Image buffer is write-only. Code memory is a separate read-only block of instructions, each instruction (all arguments included) taking up one slot.

A drawing instruction works from 3 adjacent registers, address of the first one specified by the user, and interprets them as X, Y, and W, where X and Y are the coordinates where the square shall be drawn and W is the width of the square. All coordinates are in pixels, from top-left of the image buffer, counting from 0. Width is in pixels. The coordinate specified will be the top left pixel of the square being drawn.

Value of all registers are 0 before the VM runs the first instruction.

When addressing in indirect mode, address is always taken as modulo NUMREGS.

Instruction set

```
Move:
 movr  Rdest, Rsrc   - move value from Rsrc to Rdest
 movc  Rdest, const  - move constant to Rdest
 movi  Rdest, Rsrc   - like movr, but both values of Rdest and Rsrc are interpreted
                       as register addresses (indirect memory addressing)

Arithmetics (all of these affect both Z and O flags):
 add  Rdest, Rsrc   - Rdest := Rdest + Rsrc
 sub  Rdest, Rsrc   - Rdest := Rdest - Rsrc
 mul  Rdest, Rsrc   - Rdest := Rdest * Rsrc
 div  Rdest, Rsrc   - Rdest := Rdest div Rsrc
 mod  Rdest, Rsrc   - Rdest := Rdest mod Rsrc

Draw:
 brush Rstart       - brush using the values in the 3 registers starting with the
                      one pointed to by Rstart
 erase Rstart       - erase using the values in the 3 registers starting with the
                      one pointed to by Rstart

Misc:
 exit               - stop execution (the image is ready)
```

PC and jumps

After the current instruction is loaded from the slot pointed by PC, but before the instruction takes effect, PC is immediately increased by one. Thus the shortest infinite loop is:

```
movc R3, 1
sub R0, R3
```

or using the PC alias:

```
movc R3, 1
sub PC, R3
```

The VM runs at most MAXINST instructions, if it doesn't encounter 'exit' before then, an error is generated. Code memory size is MAXCODE slots; above that, PC simply rolls over.

Arithmetics

Arithmetics is evaluated at arbitrary precision, then the result is fixed using the following method (using pseudo code, assuming the precise result is R):

```
Z := 0
O := 0
while R > 2147483647 do
        O := 1
        R -= 4294967296
end
while R < -2147483648 do
        O := 1
        R += 4294967296
end
if R == 0 then
        Z := 1
end
Rdest := R
```

The DIV and MOD instructions are evaluated the following way: if Rsrc == 0 then they fail; otherwise the arbitrary precision result of DIV and MOD are D and M such that

```
D := floor(Rdest/Rsrc)
M := Rdest - D*Rsrc
```

where floor(x) rounds the real value x downward to the nearest integer. Examples:

Rdest	Rsrc	D	M
22	5	4	2
-22	5	-5	3
22	-5	-5	-3
-22	-5	4	-2

(This is how division works in pascal and python, but not in c and java where the result is rounded toward zero and not downward)

Drawing

The image starts with all pixels white. Drawing beyond the canvas is tolerated by the vm - squares are clipped so that only those pixel changes take effect which are on the canvas. When drawing a square from 2;3 with width 4, the top-left pixel is 2;3, the bottom-right pixel is 5;6. Drawing with 0 or negative width is also ignored.

Lamp of the photoplotter is expensive, thus the device has a built-in limit on the total area it is willing to draw in a single program (MAXAREA). The total area is calculated as a sum of the area of all squares drawn (both brush and erase), even for those pixels that were beyond the canvas. When the limit is reached, an error is generated.

Input

The photo to be copied.

Output

A list of photoplotter instructions in plain text format, one instruction per line.

Scoring

At the end the Euclidean distance of the image buffer and the input image is calculated and a score is given based on the best submission so far.

Evaluated score:

```
D := round(sqrt(SUM((P-Q)*(P-Q))))
```

where the SUM is taken over each P, Q corresponding pixels of the images

Scaled real score:

```
SCORE := round(100 * (1 - sqrt(1 - Dmin/D)))
```

where Dmin is the best submission so far.

Example program

(Broken into two columns for clarity, same as `example.asm`)

```
movc R3,20
movc R4,5
movc R5,5
movc R6,60
movc R7,1
movc R8,4
movc R9,6
movc R10,5
movc R11,17
movi R3,R8
add  R3,R7
movi R3,R5
add  R3,R7
movi R3,R9
sub  R3,R7
sub  R3,R7
brush R3
brush R3
brush R3
add  R3,R5
add  R4,R5
add  R4,R6
sub  R10,R7
sub  R7,Z
mul  R11,R7
sub  PC,R11
movc R3,50
movr R50,R6
div  R50,R8
```

```
movc R51,-10
movc R52,40
erase R3
erase R3
movc R51,40
erase R3
erase R3
movc R50,90
movc R51,15
erase R3
erase R3
movc R52,25
brush R3
movc R51,30
brush R3
movc R50,145
movc R51,5
movc R52,50
erase R3
erase R3
movc R40,65
add  R50,R40
erase R3
erase R3
add  R50,R40
movc R51,15
movc R52,40
brush R3
exit
```

Output of example program

Top left corner of the output image for the example program:

R. History

While local citizens are viewing an exhibition, they're very much interested in the background and history of the paintings. They often ask what a specific work is based on. The artists name other works that inspired them to paint that particular painting. The work that borrows from other great works a lot is considered to be great itself.

Your goal is to create art that lasts, so you can't risk to base your paintings on questionable works. Given a set of credits (which pictures are borrowed from which), you need to find the greatness of each work.

The parameters required to calculate the greatness of a picture are carefully documented by art historians. According to the Academy of Numerical History of Art the greatness of a painting is strictly determined by the sum of two components: an intrinsic greatness (based on the size of the painting, the number of colors used and so on) and an inherited greatness: if P proportion of painting A is based on painting B then the greatness of A is increased by P times the greatness of B.

However if A and B are painted at the same time, they could be based on each other. This makes the calculations difficult, so the historians helpfully avoid mentioning such relations that may cause problems.

Input

First line of the input contains N, the number of paintings, and M, the number of relations between paintings. The second line contains the intrinsic greatness of each painting in order. Then M lines follow with A, B, P, saying that P proportion of painting A is borrowed from B. The paintings are numbered from 1.

Output

Output is the greatness of each painting on a single line in order. Each value must be precise to 3 digits after the decimal point.

Example input

```
3 2
0.5 1.0 2.0
1 2 .5
1 3 .2
```

Example output

```
1.4 1 2
```