Problems Index Tue Oct 11 07:57:42 EDT 2016

BOSPRE 2016 PROBLEMS
------ ---- --------

The problems are in approximate order of difficulty, easiest first.

For the first 3 problems ONLY, the autojudge will return the input and output of the judge's first failed test case, on an incorrect submission.

PYTHON is fast enough to do the first 6 problems if you program with moderate care.

problems/oneboggle
Where oh where is the word when I need it!
Boston Preliminary 2016

problems/whatwave
What you see is what you get.
Boston Preliminary 2016

problems/notlost
Help for the wandering sole.
Boston Preliminary 2016

problems/colorpuzzle
Odd that you should think this way.
Boston Preliminary 2016

problems/exsort
Sorting under disk duress.
Boston Preliminary 2016

problems/monte
When what is hard is made easy.
Boston Preliminary 2016

problems/hulldiameter
Almost round is better than notched.
Boston Preliminary 2016

problems/gaudy
Overlapping colors are bad for business.
Boston Preliminary 2016

problems/substrings
Quick! Quick! Where are they all?
Boston Preliminary 2016

problems/economy
And you thought life was tough at the bottom.
Boston Preliminary 2016

One-Boggle

One-Boggle is a game in which you are given a line full of random letters and you are asked to find as many words in the line as possible. In our version, you are also given a dictionary containing all the words you are allowed to find.

Found words may go from left to right or right to left. Words may overlap. A dictionary word may appear more than once.

Input

For each of several test cases:

* A single line giving the test case name
* Up to 1000 lines, each containing a single dictionary word.
* A single line containing just '*'.
* Many boggle lines, each containing random letters.
* A single line containing just '*'.

All letters are lower case. Words only contain letters. No line (or word) is longer than 80 characters.

Input lines are read from the standard input. Input ends when an end of file is read.

Output

For each test case, first an exact copy of the test case name input line, and then a copy of each boggle line with letters that are not part of a word replaced by periods '.'.

Output lines are printed to the standard output. Output ends when the program terminates.

```
Sample Input
------ -----

-- SAMPLE 1 --
my
hi
*
xmybhiip
xymbihipqihihic
mxymwihibhhiynmyx
*
-- SAMPLE 2 --
bone
nobody
need
ted
*
biboneydoconedenobano
iwaydoboneedxy
aneedetapomporosaydoboneedetfew
*


Sample Output
------ ------

-- SAMPLE 1 --
.my.hi..
.ym.ihi..ihihi.
..ym.ihi..hi..my.
-- SAMPLE 2 --
..bone........enob...
...ydoboneed..
.needet..........ydoboneedet...
```

Actual boggle differs from One-Boggle in that (1) the game board is a 4x4 or 5x5 square, (2) words may run from any square to any of the 8 adjacent squares, (3) the human player can use any word they know, (4) the the player has a time limit.

```
File:      oneboggle.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Oct 11 07:49:24 EDT 2016
```

What Wave?
---- -----

You are programming a sophisticated electronic device repair robot, and it has to be able to figure out whether the screen it is looking at is showing a picture of a square wave or a sine wave, and what the period of the wave is.

For example,

```
xx      xxxxxx      xxxxxx      xxxxxx      xxxxxx



  xxxxxx      xxxxxx      xxxxxx      xxxxxx      xxxxxx
```

is a square wave with period 12 columns and

```
 xxx             xxx             xxx             xxx
x   x           x   x           x   x           x   x
     x         x     x         x     x         x     x
                     x               x               x
      x       x               x               x
       x     x         x     x         x     x         x
        x   x           x   x           x   x
         xxx             xxx             xxx
```

is a sine wave with period 16 columns.

Input

For each of several test cases, first a line containing just the test case name, then several lines containing the picture of the waveform, and then a line containing just ‘*’. At least two full cycles of the waveform are pictured.

All lines are at most 80 columns long, Picture lines contain only single spaces and ‘x’ characters. All picture lines are of the same length. There are no more than 20 lines in any picture.

Input lines are read from the standard input. Input ends when an end of file is read.

Output

For each test case, first an exact copy of the test case name input line, and then one line containing first the name of the wave type and second the period in columns.

The wave type name is either ‘sine’ or ‘square’. The period must be accurate to + or - 1 column.

Output lines are printed to the standard output. Output ends when the program terminates.

```
Sample Input
------ -----

-- SAMPLE 1 --
xx      xxxxxx      xxxxxx      xxxxxx      xxxxxx









  xxxxxx      xxxxxx      xxxxxx      xxxxxx      xxxxxx
*
-- SAMPLE 2 --
 xxx             xxx             xxx             xxx
x   x           x   x           x   x           x   x
     x         x     x         x     x         x     x
                     x               x               x
      x       x               x               x
       x     x         x     x         x     x         x
        x   x           x   x           x   x
         xxx             xxx             xxx
*
[see sample.in file for more sample test case inputs]
```

```
Sample Output
------ ------

-- SAMPLE 1 --
square 12
-- SAMPLE 2 --
sine 16
[see sample.test file for more sample test case outputs]


File:      whatwave.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Sep 27 11:51:37 EDT 2016

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.
```

Not Lost
--- ----

Redfur is exploring the concept of orienteering. He has obtained a compass and a pedometer and is trying them out. He goes to a place with a lot of open fields, chooses a direction, and walks some distance in that direction. Then he chooses another direction, and walks another distance in that direction. And so forth.

At the end of several such ‘legs’ he wants to know how to get back directly to his starting point. What direction should he walk in, and how far away is it?

Input

For each of several test cases, first a test case name line, and then lines each describing one leg of his walk in the form:

direction length

where direction is the direction of the leg and length is its length. Directions are angles in degrees with

North	0 degrees
North East	45 degrees
East	90 degrees
South East	135 degrees
South	180 degrees
South West	225 degrees
West	270 degrees
NorthWest	315 degrees

Lengths are in feet.

0 <= direction < 360
0 < length <= 10,000

The direction description lines are followed by a line containing just ‘*’.

Lines are at most 80 columns long.

Input lines are read from the standard input. Input ends when an end of file is read.

Output

For each test case, first an exact copy of the test case name input line, and a single line of the form:

direction length

that says that to get back to his starting point from the end of the last leg, Redfur must walk ‘length’ feet in the given ‘direction’.

Directions should be accurate to the nearest degree and lengths to the nearest foot. You can round output to the nearest integer if you like, but you do not have to.

Output lines are printed to the standard output. Output ends when the program terminates.

```
Sample Input
------ -----

-- SAMPLE 1 --
90 100
0  100
270 100
*
-- SAMPLE 2 --
45 1414
180 2000
*
-- SAMPLE 3 --
20 100
200 100
200 100
*
-- SAMPLE 4 --
45 100
90 100
135 100
180 100
225 100
270 100
315 100
*
-- SAMPLE 5 --
125 170
85 1410
*
```

```
Sample Output
------ ------

-- SAMPLE 1 --
180 100
-- SAMPLE 2 --
315 1414
-- SAMPLE 3 --
20 100
-- SAMPLE 4 --
0 100
-- SAMPLE 5 --
269 1544


File:      notlost.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Sep 27 11:55:12 EDT 2016

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.
```

Color Puzzle

You have been asked to solve puzzles involving different plastic creatures each with some color.

A typical puzzle is:

```
  if the mouse is blue or red, then the bear is red
  if the mouse is blue or green, then the hawk is red
  if the bear is green, then the squirrel is blue or red
  if the hawk is red, then the mouse is red
  if the squirrel is green, then the bear is green
  if the squirrel is blue, then the hawk is green
  if the squirrel is red, then the hawk is blue
```

The answer to this puzzle is:

```
  the mouse is red
  the bear is red
  the hawk is blue or green
  the squirrel is blue or red
```

Input

For each of several test cases, first a test case name line. This is followed by statement lines, and at the end of these, a line containing only '*'.

Statement lines have the form:

```
    if the C1 is COLORS1, then the C2 is COLORS2
```

where C1 and C2 name creatures and COLORS1 and COLORS2 are lists containing color names separated by 'or'.

You may assume that only creatures mentioned in some statement are given to you, there is only one of each kind of creature, and each creature has a color mentioned in some statement. There are at most 8 creatures, 5 colors, and 100 statements in a test case.

Words are separated by whitespace or a comma (,) followed by whitespace. All words contain only lower case letters. All lines are at most 80 columns long.

Input lines are read from the standard input. Input ends when an end of file is read.

Output

For each test case, first an exact copy of the test case name input line, and then one creature description line for each creature. Each creature description line has the form

```
    the C is COLORS
```

where C names the creature and COLORS is a list containing color names separated by 'or'. The creature description lines may be in any order, and within one of these lines, the colors may be in any order.

Notice that in the example there are two colorings that are compatible with the statements:

	Coloring 1	Coloring 2
mouse	red	red
hawk	green	blue
bear	red	red
squirrel	blue	red

In 'the C is COLORS' lines, a color must be listed if and only if C can have the color in SOME compatible coloring. So both green and blue must be listed for the hawk, and both blue and red must be listed for the squirrel.

Output lines are printed to the standard output. Output ends when the program terminates.

At least one coloring is compatible with the statements of each test case, so every creature description line will have at least one color.

Sample Input
------ -----

```
-- SAMPLE 1 --
if the cow is brown or white, then the sheep is white
if the sheep is brown or white, then the cow is brown
*
-- SAMPLE 2 --
if the dog is black, then the cat is orange
if the horse is brown, then the cat is blue
if the dog is blue, then the horse is brown
if the cat is orange, then the dog is blue
if the cat is blue, then the horse is black
if the horse is orange, then the dog is black
if the horse is orange, then the dog is brown
if the horse is blue, then the dog is black
if the horse is blue, then the dog is brown
*
-- EXAMPLE --
[Same as example at beginning of problem description:
 see sample.in]
*
```

Sample Output
------ ------

```
-- SAMPLE 1 --
the cow is brown
the sheep is white
-- SAMPLE 2 --
the dog is orange or brown
the cat is black or brown or blue
the horse is black
-- EXAMPLE --
the mouse is red
the bear is red
the hawk is blue or green
the squirrel is blue or red
```

File: colorpuzzle.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Tue Oct 11 07:51:43 EDT 2016

External Memory Sort
-------- ------ ----

You have been asked to sort some data on disk while minimizing the number of disk block reads to a RAM cache and disk block writes from the cache. In the ultimate application of your program, the disk is much much bigger than computer RAM memory. But in the test setup, everything is being simulated with much smaller sizes.

You have a RAM memory cache that can hold C disk blocks, where a disk block is B 32-bit signed integers. You can read a block from disk to a cache block, or write a cache block to disk. In the test setup, you are being asked to sort the integers in the first I blocks of disk.

In this contest environment, the cache is implemented in a separate program with which you communicate by writing commands to your standard output and reading command responses from your standard input. You can read a disk block to a cache block, write a cache block to a disk block, read a cache block into your own program memory, and copy an integer from one cache location to another. To avoid cheating, there is no command to let you write a cache location with an integer of your own choosing.

The size of the disk is D = 2*I, twice the size of the data to be sorted. You must sort the integers in the first I disk blocks, leaving the answer in these first I disk blocks. You are permitted at most K*I disk block read operations, at most K*I disk block write operations, and at most K*I cache block to program memory read operations, where K is the smallest integer such that (C-1)**K >= I (note that C >= 3 always).

Your Input/Output
---- ------------

You write the program whose binary name is ‘exsort’. The disk and cache are implemented by a separate ‘disk controller’ program whose name is ‘disk’. The ‘disk’ program runs your ‘exsort’ program as a subprocess. The command that does this is

```
disk exsort <TEST-CASE-INPUT >TEST-CASE-OUTPUT
```

You write disk commands to your standard output which the disk controller program reads and executes. For some of these commands the disk controller then writes results which you read from your standard input.

Note that your program DOES NOT READ the test case input or write the test case output. The ‘disk’ program does this.

The commands you may write to your standard output are:

Command	Results
case	One line containing B D C I K [debug-options] or a line containing ‘0 0 0 0 0’ if there are no more test cases.

The ‘case’ command must be the first command you issue for each test case, and the next ‘case’ command indicates your are done with sorting for the last test case and are starting the next test case.

Here ‘debug-options’ denotes optional arguments you may provide to control debugging output from your program. See Test Case Descriptions and Output below.

Note that D and K are computable from I and C.

read d c	One line containing the B integers of the block read from disk. These are printed one integer per 8 columns, so the line can be up to 8*16 = 128 characters long. However, if this command or a previous command in the same test case is in error, then instead of a line containing integers, the result is a line containing just ‘ERROR’.

The d+1’st block on disk is read into the c+1’st block of cache, overwriting any previous contents of cache block. 0 <= d < D, 0 <= c < C.

write c d	No results.

The c+1’st block in cache is written to the d+1’s block of disk, overwriting any previous contents of the disk block. 0 <= c < C, 0 <= d < D.

move c1 i1 c2 i2	No results.

The i1+1’st element of the c1+1’st cache block is copied to the i2+1’st element of the c2+1’st cache block. 0 <= c1,c2 < C, 0 <= i1,i2 < B.

debug ...	No results.

This line is traced even if tracing is off. It has no other effect. It is used to convey debugging output from your program to the ‘disk’ program standard output.

Do NOT use this command if there are no debug-options in the ‘case’ command result, as doing so will cause an Incorrect Output judgment.

WARNING: If you use printf in C or C++ you must use fflush after printing a line, as in

```
printf ( "...\n", ... );
fflush (stdio);
```

Similarly if you use ‘print’ in python you must flush after printing a line, as in

```
print ’....’
sys.stdout.flush()
```

Otherwise your output will be trapped in a buffer and never get to the disk program. The C++ 'endl' IO manipulator and JAVA 'println' functions flush this buffer, so if you are using these functions nothing special needs to be done.

Test Case Descriptions And Output
---- ---- ------------ --- ------

In the test environment, the disk controller is a separate program named 'disk' that runs your 'exsort' program as a subproccess. The controller (and not your program) reads test case descriptions from its standard input and the controller (and not your program) writes test case output to its standard output.

Each test case description has two lines. First a line containing just the test case name. Then a line containing

B C I SEED TRACE [debug-options]

where B, C, I are as given above and SEED is an unsigned integer used to seed a pseudo-random number generator that generates the input on disk. 10**8 <= SEED. The debug-options are passed to your program as part of 'case' command result, and may be used to control debugging output in your program.

TRACE is either 'no' indicating you do not want communication between your program and the controller traced, or 'yes' indicating you do want it traced. Tracing is only used to debug, and consists of the controller outputting to its standard output (not your program) the lines it reads from your program, prefaced by '<< ', and the lines it writes to your program, prefaced by '>> '. Tracing also adds some other information lines beginning with '**'.

Since this is the initial test of your program, the initial sizes of parameters are quite small:

2 <= B <= 16, 3 <= C <= 17, 4 <= I <= 1024

Test case input lines (read by the 'disk' program from its standard input) must not be longer than 80 characters. Test case input ends with an end of file.

The 'disk' program, at the beginning of the test case (after reading the 'case' command from the 'exsort' program and the test case input lines from the standard input), writes integers from 1 though B*I into the first I blocks of the disk, and then uses the SEED to shuffle them randomly, before returning a reply to the 'exsort' 'case' command. When the test case is finished (i.e., the 'exsort' program outputs its NEXT 'case' command), the 'disk' program checks that the first I blocks of disk now contain these integers in sorted order.

Test case output (written by the 'disk' program to its standard output) consists of two lines, the first being an exact copy of the test case name line, and the second containing either just 'OK' or an error message describing the first error found when running the test case or checking to see if its output is in fact sorted.

It is an error if your program executes too many disk reads or writes during a test case (the limit for both is K*I). The only other limit on your program is the CPU time limit which should be more than sufficient to allow any algorithm that does not exceed the read/write limits.

```
Sample Input
------ -----

-- SAMPLE 1 --
2 3 8 675874567 no
-- SAMPLE 2 --
4 3 32 789467324 no
-- SAMPLE 3 --
4 5 32 647583765 no
-- SAMPLE 4 --
4 5 31 548741098 no
-- SAMPLE 5 --
4 6 64 508980765 no
-- SAMPLE 6 --
13 8 674 389701520 no
-- SAMPLE 7 --
16 17 1024 648935623 no

Sample Output
------ ------

-- SAMPLE 1 --
OK
-- SAMPLE 2 --
OK
-- SAMPLE 3 --
OK
-- SAMPLE 4 --
OK
-- SAMPLE 5 --
OK
-- SAMPLE 6 --
OK
-- SAMPLE 7 --
OK
```

```
SAMPLE TRACE INPUT
------ ----- -----

-- SAMPLE TRACE 1 --
2 3 8 675874567 yes


SAMPLE TRACE OUTPUT
------ ----- ------

-- SAMPLE TRACE 1 --
<< case
>> 2 16 3 8 3
<< read 0 0
>>       10       7
<< read 1 1
>>       11       4
<< move 1 1 2 0
<< move 0 1 2 1
<< write 2 0
** DISK 0:       4       7
<< move 0 0 2 0
<< move 1 0 2 1
<< write 2 1
** DISK 1:      10      11
..... see sample-trace.trace for omitted output .....
<< case
** 24 blocks read out of 24 allowed
** 24 blocks written out of 24 allowed
OK
>> 0 0 0 0 0

File:      exsort.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Mon Oct  3 14:45:26 EDT 2016

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.
```

Monte

You work for Applied Probability Enterprises Inc., and have been asked by one of their customers to write a program that answers the question:

  If N points are placed uniform randomly in the unit interval, what is the probability that the smallest distance between two of these points will be at most D.

Note that the end points of the unit interval itself DO NOT count as points to which a distance can be measured.

Input

For each of several test cases, a line containing just

        N D

where

    2 <= N <= 20   0 < D < 1

D has exactly 4 decimal places. Input ends with an end of file.

Output

For each test case, a line containing just

        N D P

where N and D are copied from the input and P is the desired probability. D has exactly 4 decimal places and P has exactly 2 decimal places.

Sample Input
------ -----

```
 8 0.0200
10 0.0200
10 0.0100
10 0.0050
15 0.0050
15 0.0020
20 0.0020
```

Sample Output
------ ------

```
8 0.0200 0.70
10 0.0200 0.86
10 0.0100 0.61
10 0.0050 0.37
15 0.0050 0.66
15 0.0020 0.35
20 0.0020 0.54
```

File: monte.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Fri Aug 12 21:42:59 EDT 2016

Hull Diameter
---- --------

Your boss wants you to find the diameter of a set of points, defined as the longest distance between any pair of the points. The difficulty is that there are very many points.

So your boss has built special hardware that finds the convex hull with exceptional speed. Given this, he wants you to find the diameter of the set, which is the same as the diameter of the convex hull.

Input

Input consists of one or more test cases. Each test case begins with a test case name line. This is followed by a line containing just N, the number of points. Then come N point description lines of the format

```
x y
```

Here

```
3 <= N <= 100,000
-1000 <= x, y <= 1000
number of test cases per input file <= 1,000
```

The points (x,y) are the vertices of the convex hull given in clockwise order, where x and y are floating point numbers. The interior angle at any vertex is less than 180 degrees.

Input ends with an end of file. No input line is longer than 80 characters.

Output

For each case, output one line containing exactly the test case name, followed by one line containing:

```
D i j
```

where D is the hull diameter printed with exactly 9 decimal places and i and j are the indices of the vertices separated by D. Vertex indices are 1, 2, ..., N in the order that their description lines appear in the input.

You are REQUIRED to output the two vertices so i < j. For simplicity, the input will be such that the output will be unambiguous; no two pairs of vertices will be at distance D.

```
Sample Input
------ -----

-- TRIANGLE --
3
0 1
1 0
0 0
-- ALMOST EQUILATERAL TRIANGLE --
3
-0.5 0.866025403784438
1 0
-0.515038074910054 -0.857167300702112
-- DECAGONAL --
10
-20 -30
-27.6265470493195 -6.7520357810226
-30 50
-25.7378332533286 57.4019499735396
40 90
46.6457282195071 82.2651502425173
70 -4
65.7983884328896 -22.3095807142023
20 -80
0.726418814447584 -62.7190939693087


Sample Output
------ ------

-- TRIANGLE --
1.414213562 1 2
-- ALMOST EQUILATERAL TRIANGLE --
1.740711392 2 3
-- DECAGONAL --
171.172427686 5 9
```

```
File:      hulldiameter.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Aug 30 04:49:22 EDT 2016

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.
```

Gaudy Artworks

You have been hired by Gaudy Artworks Inc. and are being asked to help them manufacture one of their specialty products. This consists of a white board in which a number of circles are randomly placed. Then a subset of the circles which do NOT overlap each other too much have their interiors colored various colors, while the outlines of all the circles are overprinted in black.

Given a board with the circles placed randomly on it, you have been asked to find a largest set of non-overlapping circles.

Your boss has defined overlapping as meaning that the intersection of two circles has area greater than 0.5.

Input

Input consists of one or more test cases. Each test case begins with a test case name line. This is followed by a line containing just N, the number of circles. 2 <= N <= 40.

Then come N circle description lines, each of the format

```
X Y R
```

where

```
-1000 <= X, Y <= 1000
5 <= R <= 100
```

Each line describes a circle of center (X,Y) and radius R.

The circles are assigned indices 1, 2, 3, ..., N in the order that their descriptions appear in the input.

Input ends with an end of file. No input line is longer than 80 characters.

Output

For each case, output one line containing exactly the test case name, followed by one line containing the maximum number M of circles that have no overlap, followed by the indices of M circles that have no overlap. M must be maximized. There may be several solutions with the same M; output only one.

```
Sample Input
------ -----

-- SAMPLE 1 --
5
 0  0 5
 0 10 5
10  0 5
10 10 5
 5  5 5
-- SAMPLE 2 --
6
 0  0 10
10  0 10
20  0 10
30  0 10
40  0 10
50  0 10
-- SAMPLE 3 --
6
 0  0 12.5
10  0 10.3
20  0 12.5
30  0 10.3
40  0 11.1012295764
50  0 10.3
```

```
Sample Output
------ ------

-- SAMPLE 1 --
4 1 2 3 4
-- SAMPLE 2 --
3 1 4 6
-- SAMPLE 3 --
2 1 6


File:      gaudy.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Thu Oct  6 23:23:42 EDT 2016

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.
```

Substrings

You work for a new group doing DNA analysis, and have been asked to write a program that will accept a DNA string D and a large number of queries of the form

    How many times does string S appear as a substring
    of D and what are the starting locations of these
    appearances?

In the long run both D and the number of queries will be very large, but in the first version of your program D will not be excessively large.

Input

For each of several test cases, first a line containing just the test case name, then a possibly very long line containing the string D, then many lines each containing a query, and then a line containing just '*'.

Each query line contains just the string S as it is in the query above.

D and S are strings of the letters A, C, G, and T.

    1 <= length D <= 20,000
    1 <= length S <= 20,000
    1 <= sum of length S over all queries <= 200,000,000

The test case name line contains at most 80 columns. Other lines contain at most 20,000 columns. Input ends with an end of file.

Output

For each test case, first an exact copy of the test case name input line, and then for each query one answer line of the form

    L M p1 p2 ... pM

where L is the length of the query input string S, M is the number of occurrences of S in D, and p1, p2, ..., pM are the positions of the M occurrences. M may equal 0. The first letter in D has position 0, the last has position 'length D - 1'.

The positions in a query answer line may be in any order but must not contain duplicates.

```
Sample Input
------ -----

-- SAMPLE 1 --
AAAAAAAACCCCAAAAA
AAA
AAAAA
AACC
AACCCAA
AACCCCAA
AACCCCCAA
CAAAAA
CAAAAAA
*
-- SAMPLE 2 --
ACGGCACGTCGTGTACGTC
AC
ACG
ACGT
CT
GT
*
```

```
Sample Output
------ ------

-- SAMPLE 1 --
3 9 14 13 12 0 1 2 3 4 5
5 5 12 0 1 2 3
4 1 6
7 0
8 1 6
9 0
6 1 11
7 0
-- SAMPLE 2 --
2 3 0 14 5
3 3 0 14 5
4 2 14 5
2 0
2 4 12 16 7 10


File:      substrings.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Wed Oct  5 05:05:03 EDT 2016

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.
```

WhatWhat? Economy

Congratulations! You have been appointed the Economics Minister of the fabulous country What?What?. The economy of What?What? is doing well, but the Parliament and Prime Minister are demanding that you tell them how much better it can be if everyone just produces more and consumes more.

Luckily for you, your predecessor compiled precise economic statistics before she left for the Asylum. Also lucky is the fact that What?What? has a one item economy. The good citizens, in their never ending quest for answers, decided some time ago that only Calories mattered, and

> "A Calorie is a Calorie is a Calorie, be it from carrots or spinach or steak!"

So all the producers of What?What? produce Calories, the transporters transport Calories (all equal by law, regardless of weight or volume), and the consumers consume Calories, heedless of where they come from. Also the unit of money is the Calorie!

What?What? has a bunch of towns connected by roads. Some towns have producers and some consumers and some both and some neither. Shipping is done by specifying a route that is a sequence of links.

A link is a road between two towns plus the trucking company that transports Calories over the road. Each link has a capacity that depends mostly upon the number of What?What?iens who are willing to drive trucks over the road. There is a cost of transporting a Calorie over a link, expressed as a small positive number in units of Calories per Calorie. If there is a link from town X to town Y, there may or may not be a link from town Y back to town X, and if there is, the capacities and costs may differ. However, there is at most one link from any town X to any other town Y (the trucking companies are possessive).

What you are to maximize is the sum of the Calories produced minus the Calories spent on transport. This is called the GCP: Gross Calorie Product.

There is one more thing, however. Some links are run by Mafiosi who REQUIRE that a minimum number of Calories be transported over their links. Your predecessor found that to keep the Mafiosi happy, it was occasionally necessary to ship Calories in an aimless loop; though to keep costs down this had to be minimized.

Input

Input consists of one or more test cases. Each test case begins with a test case name line. This is followed by a line with the format:

```
N M
```

where N is the number of towns and M the number of links. There follow N town description lines, each with the format:

```
producer-capacity consumer-capacity
```

which gives the maximum producer and consumer capacities for a town. The towns are indexed 1, 2, 3, ... N in the order of their town description lines.

Then follow M link description lines, each with the format:

```
s d capacity cost minimum
```

which describes the link from town s to town d. The link has the given capacity in Calories, cost in Calories per Calorie, and minimum number of Calories that must be transported in order to satisfy the local Mafiosi.

```
2 <= N <= 20
1 <= M <= 100
0 <= producer-capacity <= 2,000
0 <= consumer-capacity <= 1,000
1 <= s,d <= N
1 <= capacity <= 1,000
0.01 <= cost <= 0.05
0 <= minimum <= capacity
S <= 1,000.00
```

where

```
S = sum over all links of:
       (capacity - minimum) * cost
```

No two links have the same source and destination. All numbers are integers except cost which has 2 decimal places. Input ends with an end of file. No input line is longer than 80 characters.

Output

For each case, output one line containing exactly the test case name, followed by one line containing either the maximum GCP with exactly 2 decimal places, or containing exactly the text:

```
Mafiosi prevent solution!
```

Sample Input
------ -----

```
-- SAMPLE 1 --
4 4
100 0
0   0
0 100
0   0
1 2 1000 0.01 0
2 3 1000 0.05 0
2 4 1000 0.01 0
4 3 1000 0.01 0
-- SAMPLE 2 --
4 3
100 0
0 100
0 0
0 0
1 2 1000 0.01 0
3 4 1000 0.01 200
4 3 1000 0.01 0
```

```
-- SAMPLE 3 --
4 3
100 0
0 100
0 0
0 0
1 2 1000 0.01 0
3 4 1000 0.01 200
4 2 1000 0.01 0
-- SAMPLE 4 --
4 4
100 0
0 100
100 0
0 100
1 2 1000 0.01 200
2 3 1000 0.01 0
3 4 1000 0.01 0
4 1 1000 0.01 0


Sample Output
------ ------

-- SAMPLE 1 --
97.00
-- SAMPLE 2 --
95.00
-- SAMPLE 3 --
Mafiosi prevent solution!
-- SAMPLE 4 --
194.00
```

```
File:      economy.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Mon Oct 10 12:04:32 EDT 2016

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.
```