Problems Index Tue Oct 15 03:56:22 EDT 2019

BOSPRE 2019 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 7 problems if you program with moderate care.

problems/cuneiform
Old ways are the best ways?

problems/collatz2
Into the realm of unproven conjecture.

problems/bubblegram
To approve or not to approve?

problems/annual
How much do you want in return?

problems/convexpainter
Computer assisted art is fine art.

problems/jmaze
Jumping is better than walking!

problems/treesearch
Where in this thing are you?

problems/whoosh
An elliptical problem.

problems/flowman
Get the most from the budget.

problems/railgun
Best be a straight shooter.

problems/trisurvey
Triangles are the thing.

(This Page Intentionally Left Blank)

(This Page Intentionally Left Blank)

Cuneiform Numbers
--------- -------

The Babylonians used a base 60 number system. To represent digits, they used symbols for 1 and 10 and put them together as in a word, with the symbol for 10 first. We will use 'v' for 1 and '<' for 10. The Babylonians just used a space for 0; we will use '_'.

Given this notation, here are some Babylonian numbers and their decimal equivalents:

```
    vvv                 3
    <<                  20
    <<vvv               23
    v  v                1*60 + 1 = 61
    v  <                1*60 + 10 = 70
    <  _  v             10*60*60 + 0*60 + 1 = 36001
    <vvv  _  _          13*60*60 + 0*60 + 0 = 46800
```

Some friends of yours have taken to using cuneiform numbers in their text messages, which is annoying. Some other friends are pestering you to write an app that translates these numbers into decimal. So as a warmup, you are to write a program that takes cuneiform numbers and translates them into decimal.

Input

A sequence of lines each containing one cuneiform number. No line will be longer than 80 characters or will represent a number greater than 10^9. Input ends with an end of file.

Output

Output is a copy of the input with the decimal number represented by an input line appended to that input line (separated from the rest of the line by a space character). For example, the input lines

```
        vvv
        <<
        <<vvv
        v v
```

produce the output lines

```
        vvv 3
        << 20
        <<vvv 23
        v v 61
```

Note

Be sure to test your program against the sample.in and sample.test files (see next page) BEFORE you submit. You can do this by simply executing the command:

```
        make test
```

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

```
vvv
<<
<<vvv
<< _
<< vvv
v v
v _ _
v _ _ _
v v v v
<<<<<vvvvvvvvv
v _ _ _ _
<<<<<vvvvvvvvv _ _ _
v <<<<<vvvvvvvvv _ _ _
vv << vvv <<< vvvv
<<v <vvv vvvvvvvvv <<<<<
<<<<vvv <<vvvvv <<<<<v <vvvvvvvvv
```

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

```
vvv 3
<< 20
<<vvv 23
<< _ 1200
<< vvv 1203
v v 61
v _ _ 3600
v _ _ _ 216000
v v v v 219661
<<<<<vvvvvvvvv 59
v _ _ _ _ 12960000
<<<<<vvvvvvvvv _ _ _ 12744000
v <<<<<vvvvvvvvv _ _ _ 25704000
vv << vvv <<< vvvv 30252604
<<v <vvv vvvvvvvvv <<<<< 4583390
<<<<vvv <<vvvvv <<<<<v <vvvvvvvvv 9381079
```

```
File:      cuneiform.txt
Author:    Shai Simonson <shai@stonehill.edu>
Editor:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Oct 15 10:07:23 EDT 2019

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

Collatz Revisted
------- --------

Consider the operation F(N) defined as:

```
        F(N) = N/2      if N can be divided by 2

        F(N) = N/3      if N can be divided by 3

        F(N) = 5N + 1   otherwise
```

We conjecture that for any natural number N, applying F repeatedly to any non-zero natural number will eventually arrive at the number 1.

Thus

```
        [1]     F(5) = 26
        [2]     F(26) = 13
        [3]     F(13) = 66
        [4]     F(66) = 33
        [5]     F(33) = 11
        [6]     F(11) = 56
        [7]     F(56) = 28
        [8]     F(28) = 14
        [9]     F(14) = 7
        [10]    F(7) = 36
        [11]    F(36) = 18
        [12]    F(18) = 9
        [13]    F(9) = 3
        [14]    F(3) = 1
```

so F applied 14 times starting at 5 will get to 1.

The original Collatz Conjecture, named after Lothar Collatz, who first proposed it in 1937, is for a simpler F, and has never been proved or disproved.

Given N, you are to find the number of times F must be applied starting at N to get to 1.

Input

Input contains several test cases. For each test case there is one input line containing N. Input ends with an end of file.

All numbers N input will be such that applying F many times to N will not generate a number above 10^9.

Output

For each test case, one line containing first N and then the number of times F must be applied starting at N to get to 1.

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

```
1
2
3
4
5
6
7
8
9
10
100
1000
10000
100000
1000000
10000000
100000000
1000000000
```

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

```
1 0
2 1
3 1
4 2
5 14
6 2
7 5
8 3
9 2
10 15
100 11
1000 28
10000 28
100000 55
1000000 44
10000000 90
100000000 55
1000000000 87
```

```
File:      collatz2.txt
Authors:   Tom Widland
           Bob Walton <walton@seas.harvard.edu>
Date:      Tue Oct  8 07:49:59 EDT 2019
```

Bubblegram

Iffy the 'artist' produces a kind of abstract painting that he calls a 'bubblegram'. This consists of a bunch of non-intersecting circles that Iffy paints different colors. To help him, Iffy has built a machine that will project a random circle on the canvas. He can then accept or reject the circle.

However, many of his bubblegrams have so many circles that Iffy has trouble seeing if the projected circle intersects any existing circle. So he wants you to build a second machine that tells him if a projected circle intersects an existing circle.

More specifically, he wants your machine to output the FIRST one of the following 'judgments' that applies:

inside	The projected circle is inside some existing circle, OR, some existing circle is inside the projected circle.
overlap	The projected circle partly overlaps an existing circle (the intersection has non-zero area less than the area of either circle).
touch	The projected circle just touches an existing circle.
outside	The projected circle is completely outside all existing circles.

As a first cut, you are to write a program that works if there is only one existing circle.

Input

A sequence of test cases. Each test case consists of a single line of the form:

```
XP YP RP XE YE RE
```

defining a projected circle with center (XP,YP) and radius RP and an existing circle of center (XE,YE) and radius RE.

```
-1,000 <= XP,YP,XE,YE <= +1,000
1 <= RP,RE <= 500
```

All numbers are integers. Input ends with an end of file.

Output

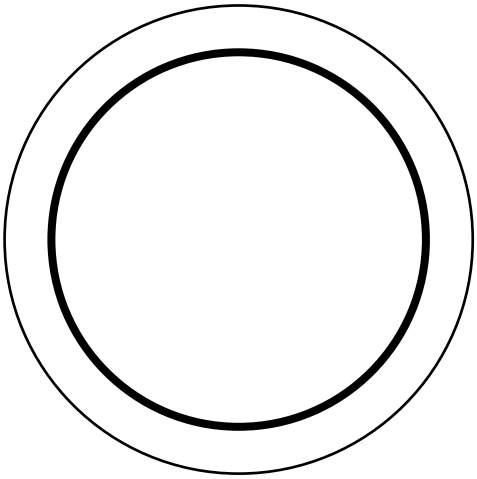
For each test case, one line, consisting of a copy of the test case input line followed by space character followed by the judgment.

NOTE: Because input numbers are integers, you can solve this problem without using floating point arithmetic in a way that makes the judgments precise. The judge's solution does this. It is still possible to solve the problem using floating point if you treat tiny values as equal to zero.

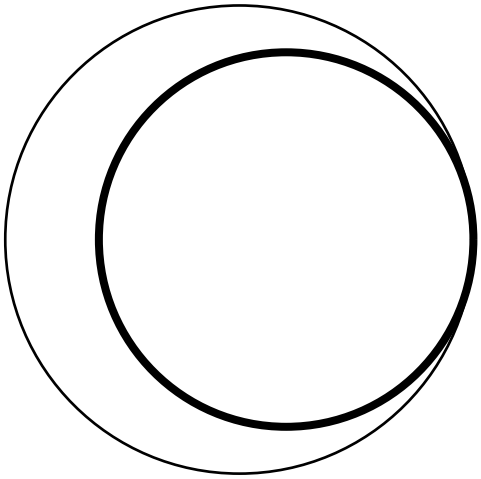
```
Display
-------

The input can be displayed in X-Windows or printed by
the commands

        display_bubbles sample.in
        display_bubbles sample.test
        print_bubbles sample.in
        print_bubbles sample.test

where the .in or .test files can be replaced by any test
case input or output file.


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

0 0 5 0 0 4
0 0 5 1 0 4
0 0 5 1 0 6
0 0 5 8 0 4
0 0 5 8 0 3
0 0 5 9 0 3
-3 10 10 1 13 15
-3 10 10 1 13 5
-3 10 10 1 13 6
-3 10 10 9 19 6
-3 10 10 9 19 5
-3 10 10 9 19 4
```

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

0 0 5 0 0 4 inside
0 0 5 1 0 4 inside
0 0 5 1 0 6 inside
0 0 5 8 0 4 overlap
0 0 5 8 0 3 touch
0 0 5 9 0 3 outside
-3 10 10 1 13 15 inside
-3 10 10 1 13 5 inside
-3 10 10 1 13 6 overlap
-3 10 10 9 19 6 overlap
-3 10 10 9 19 5 touch
-3 10 10 9 19 4 outside


File:           bubblegram.txt
Author:         Bob Walton <walton@seas.harvard.edu>
Contributor:    Shai Simonson <shai@stonehill.edu>
Date:           Tue Oct 15 14:00:26 EDT 2019

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

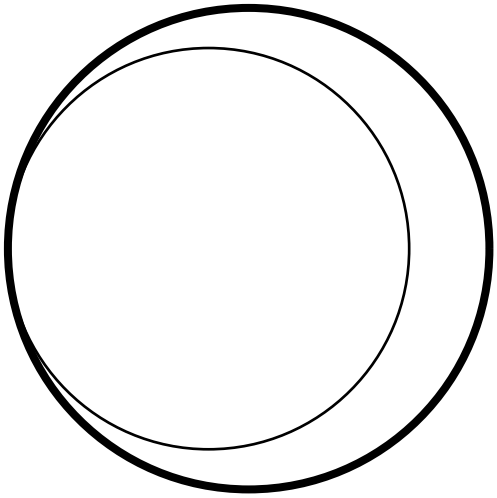
0 0 5 0 0 4 inside



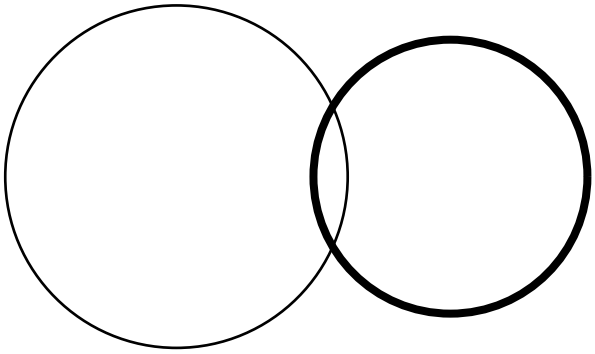
0 0 5 1 0 4 inside



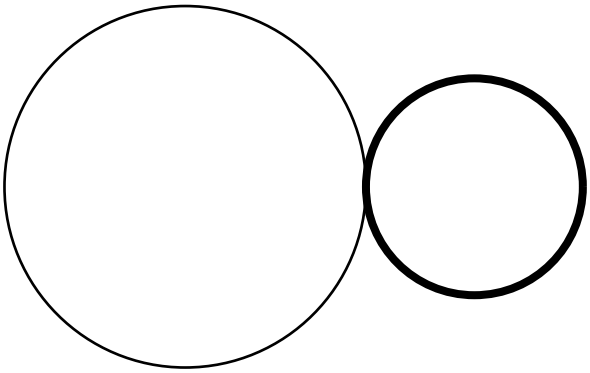
0 0 5 1 0 6 inside



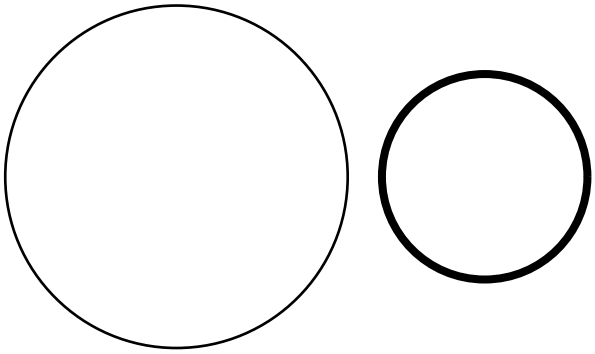
0 0 5 8 0 4 overlap



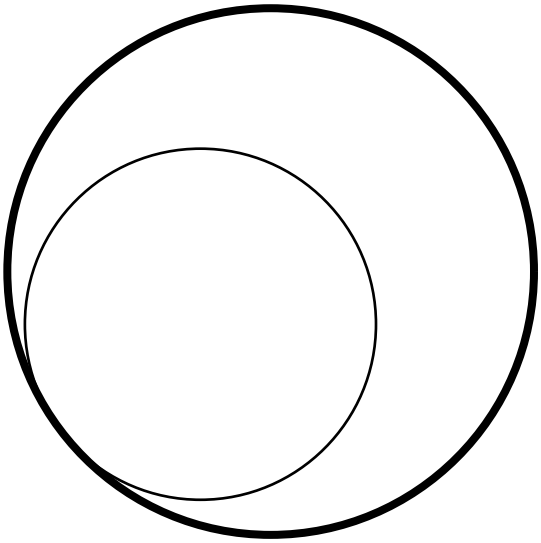
0 0 5 8 0 3 touch



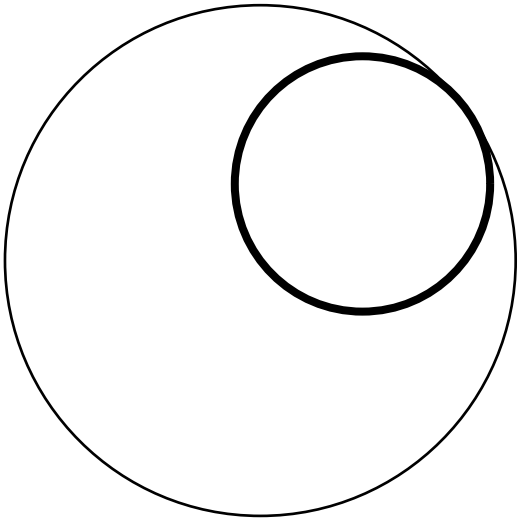
0 0 5 9 0 3 outside



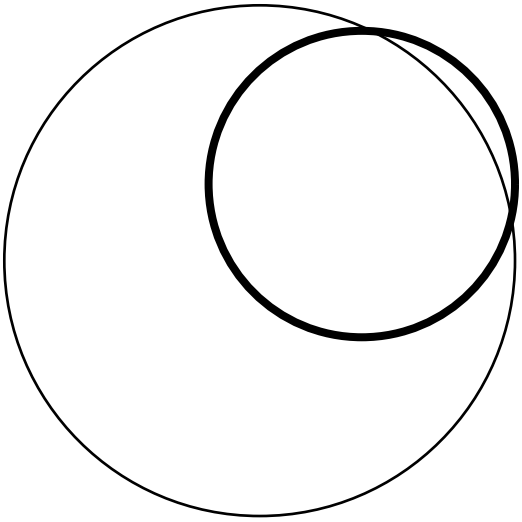
-3 10 10 1 13 15 inside



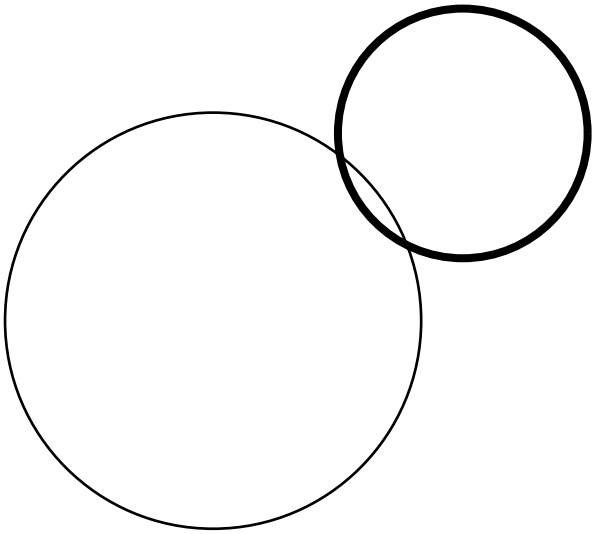
-3 10 10 1 13 5 inside



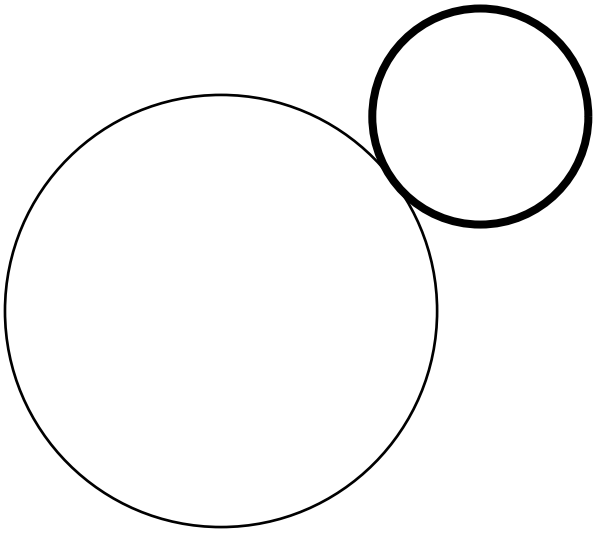
-3 10 10 1 13 6 overlap



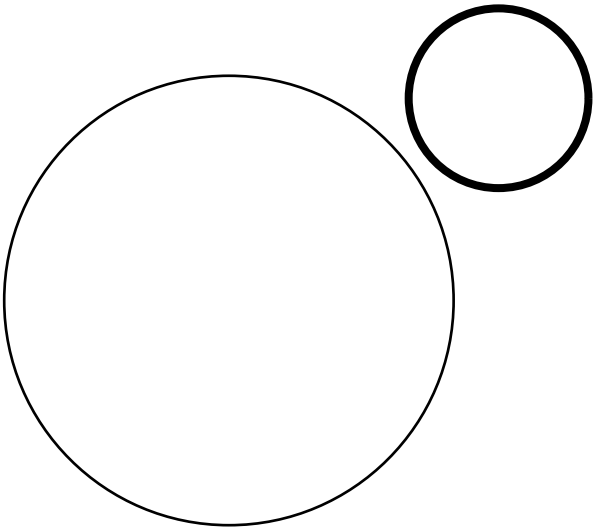
-3 10 10 9 19 6 overlap



-3 10 10 9 19 5 touch



-3 10 10 9 19 4 outside



Average Annual Return
------- ------ ------

You have a brokerage account and you let your broker buy and sell stocks and bonds in the account for your benefit. You want to know the average annual return of the account between any two times. So what do we mean by 'average annual return'?

Suppose we have an account that at the beginning of year y contains V(y) dollars, and suppose we do NOT make any deposits or withdrawals in the account. Then the average annual return A(y1,y2) from the beginning of year y1 to the beginning of year y2 is such that:

```
V(y2) = V(y1) * ( 1 + A(y1,y2)/100 )**(y2 - y1)
```

Note that A(.,.) is a percentage. we can solve for it by taking the logarithms of both sides of this equation.

However, what if we make deposits and withdrawals? Suppose at the beginning of year y when the account has value V(y) we deposit D(y) dollars and withdraw W(y) dollars. Immediately afterwards the account has

```
V(y) + D(y) - W(y)
```

dollars. Then how do we define A(y1,y2).

We define it by defining a virtual account that has v(y) dollars at the beginning of each year (before deposits and withdrawals), and is such that

```
v(y1) = V(y1)

for y1 <= y < y2:
    v(y+1) = (1 + A(y1,y2)/100)
           * (v(y) + D(y) - W(y))

v(y2) = V(y2)
```

Thus the virtual account earns at a constant rate A(y1,y2) that is independent of the year in between y1 and y2, and in this sense is the average annual rate from y1 to y2. Note that for y1 < y < y2, v(y) is NOT required to equal V(y).

When there are deposits and withdrawals, it is hard to calculate A(y1,y2) without the aid of a computer. So of course we want you to compute A(y1,y2).

However, you actually make deposits and withdrawals on the first of each month, so we want you to deal in months m1, m2 and compute the average monthly growth rate M(m1,m2) instead of the annual rate. But as monthly rates are hard to understand, we want you to report the annual rate A(m1,m2) defined by solving the equation:

```
1 + A(m1,m2)/100 = (1 + M(m1,m2)/100)**12
```

Input

A sequence of test cases. Each test case begins with a line containing the test case name. The next line has the form

```
N Q
```

where N is the number of months and Q the number of queries. This is followed by N lines of the form:

```
V D W
```

where for the I'th line V is the actual value of the account at the beginning of month I just before you deposit D and withdraw W. For simplicity, V, D, and W are non-negative integers.

This is followed by Q lines each of the form:

```
m1 m2
```

which represents a query asking for the average annual rate of return from the beginning of month m1 to the beginning of the month m2.

```
2 <= N <= 241
1 <= Q <= 20
0 <= V,D,W <= 1,000,000
V + D - W >= 0
1 <= m1 < m1 <= N
```

Input ends with an end of file. The test case name is at most 80 characters.

Input will be such that the virtual value will never be negative and -20% <= A(m1,m2) <= 20% (which implies that -2% <= M(m1,m2) <= +2%).

Output

For each test case, first an exact copy of the test case name line. Then for each query one line containing:

```
m1 m2 A
```

where 'm1 m2' copies the query input line and A is the average annual rate of return from the beginning of month m1 to the beginning of month m2. 'A' should be accurate to two decimal places.

Note that A can be negative.

Sample Input

```
-- SAMPLE 1 --
13 2
 10000      0      0
 10100      0      0
 10000    100      0
  9800      0    100
 10000      0      0
 10100     50      0
 10000      0      0
  9700      0     50
 10400      0    200
 10000    200      0
  9900      0      0
 10100      0      0
 10500      0      0
  1  13
  2  11
[ see sample.in for more sample input ]
```

Sample Output

```
-- SAMPLE 1 --
1 13 5.00
2 11 -2.63
[ see sample.test for more sample output ]
```

```
File:      annual.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sun Oct 13 20:50:47 EDT 2019
```

Convex Painter
------ -------

Iffy the 'artist' has developed a new drawing machine to help him paint abstract art. The art is to consist of a set of nested convex polygons, the outlines of which the machine is to draw. Then Iffy will paint the areas bounded by consecutively nested polygons various different colorful colors.

More specifically, the input to the machine is a convex polygon on the canvas and a sequence of points randomly chosen to be somewhere inside the canvas. The machine then chooses a subsequence of the randomly chosen points that together are the vertices of a new convex polygon that is inside the input polygon. It does this by accepting or rejecting each random point as it is presented. The first two random points that are inside the input polygon are always accepted. Thereafter the machine accepts a point P if and only if:

(1) P is inside the input polygon, and NOT on the boundary of that polygon.

(2) If P1 and P2 are the first two accepted points in order of acceptance, and Q1 and Q2 are the last two accepted points in order of acceptance, then:

    (2a) P is to the right of the directed line from Q1 to Q2.

    (2b) P1 is to the right of the directed line from Q2 to P.

    (2c) P2 is to the right of the directed line from P to P1.

Given this, if the machine accepts at least 3 points, the accepted points in order of acceptance will be the vertices of a convex polygon in clockwise order.

If Iffy does not like the result, he will reject it and ask the machine to try again. But this is outside the purview of this problem.

Input

A sequence of test cases. Each test case begins with a line containing the test case name. Following are exactly two 'point sequence lines', each of the form

```
K X1 Y1 X2 Y2 ... XK YK
```

where K >= 3 is an integer and (X1,Y1), (X2,Y2), ... (XK,YK) are K points in the plane. The first of the two lines defines the input convex polygon by giving its vertices in clockwise order. The second of the two lines defines the sequence of random points on the canvas.

The X's and Y's are integers. If the values of K for the two lines in a test case are designated K1 and K2, then

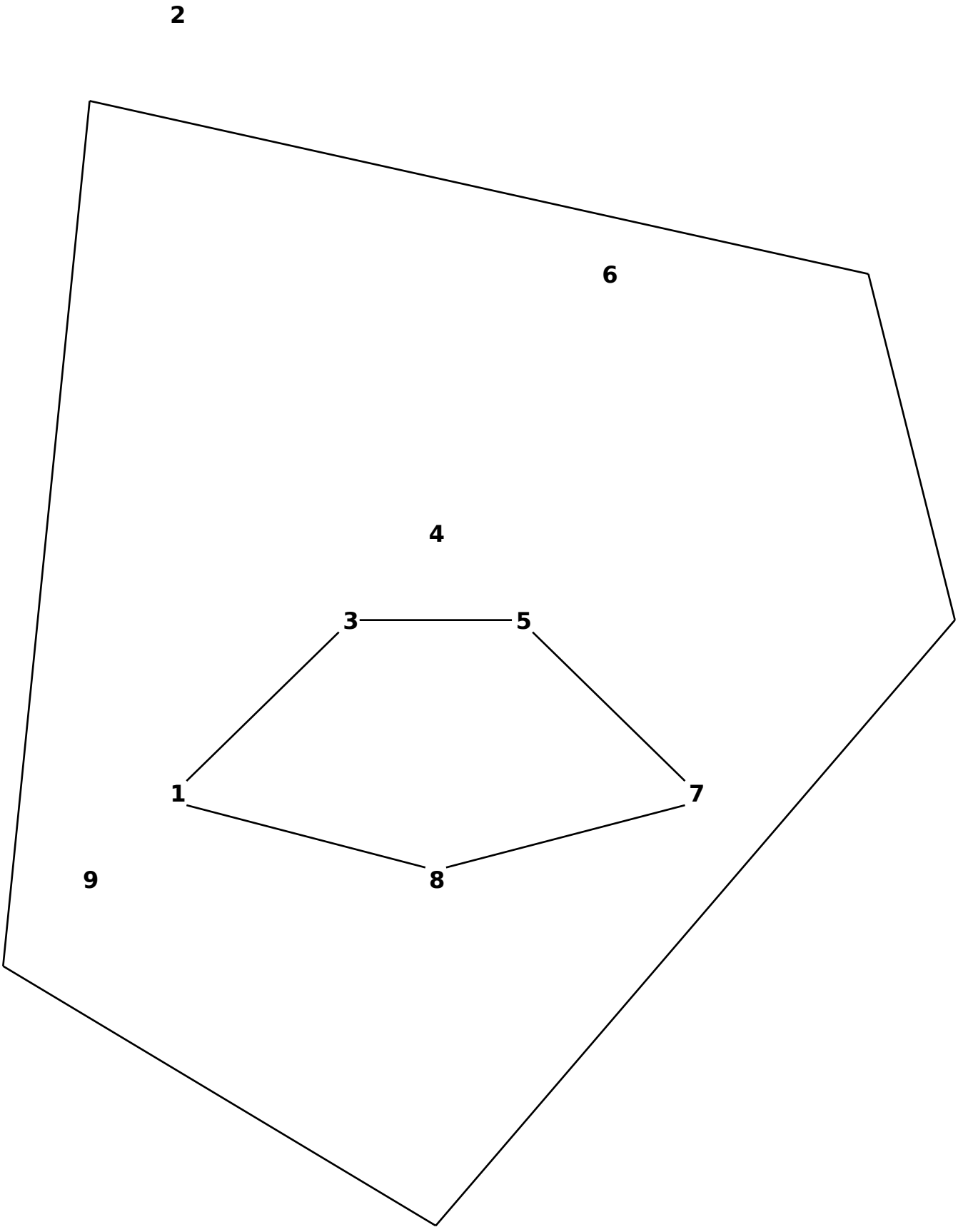
```
3 <= K1, K2 <= 1,000
sum K1*K2 over all test cases <= 1,000,000
-10,000 <= X,Y <= +10,000
```

The test case name line has at most 80 characters. Input ends with an end of file.

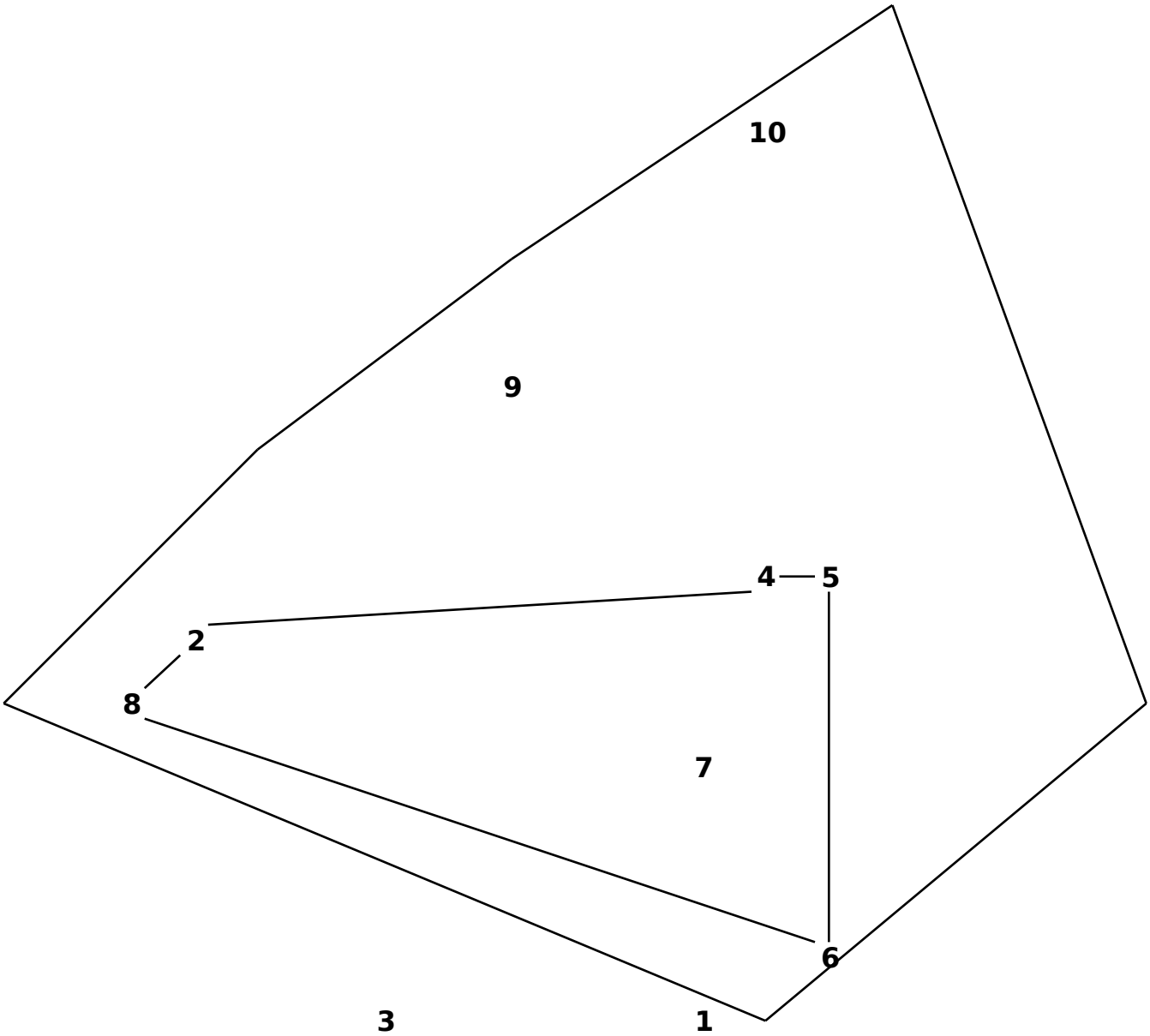
```
Output
------

For each test case, first an exact copy of the test case
name line.  Then just one point sequence line, formatted
as above, that gives the sequence of accepted points, in
order of acceptance.

The input will be such that at least 3 points will be
accepted, so the sequence of accepted points will be the
vertices of a convex polygon.


Display
-------

The input and output can be displayed in X-Windows or
printed by the commands

        display_canvas sample.in [sample.test]
        print_canvas sample.in [sample.test]

where sample.test is the output file corresponding to
sample.in, and these files can be replaced by any
test case input file and its corresponding output.
The output file can be omitted in which case only the
input will be displayed.
```

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

-- SAMPLE 1 --
5 0 0 1 10 10 8 11 4 5 -3
9 2 2 2 11 4 4 5 5 6 4 7 8 8 2 5 1 1 1
-- SAMPLE 2 --
6 3 -9 -9 -4 -5 0 -1 3 5 7 9 -4
10 2 -9 -6 -3 -3 -9 3 -2 4 -2 4 -8 2 -5 -7 -4 -1 1 3 5


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

-- SAMPLE 1 --
5 2 2 4 4 6 4 8 2 5 1
-- SAMPLE 2 --
5 -6 -3 3 -2 4 -2 4 -8 -7 -4


File:      convexpainter.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sun Oct 13 21:02:34 EDT 2019

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

-- SAMPLE 1 --



-- SAMPLE 2 --



Jumping Maze

You've been given yet another maze to traverse. This one's a bit different.

Its a grid and you can enter from the outside at any boundary square. Once you are at a square in the grid, you will find there an instruction as to what to do next. The possible instructions are:

r:c	Go to the grid square in column c of row r.
m	Go to ANY grid square whose Manhattan distance from the current square is EXACTLY m.
X	DIE!!! You Lose!
G	goal! Success! You Win!

Here r, c, and m denote integers, whereas X and G denote the letters 'X' and 'G'. The Manhattan distance between square r1:c1 and square r2:c2 is defined to be $|r1-r2| + |c1-c2|$.

As we said above, you start by moving to any boundary square. You want to win, of course; i.e., you want to move to a point with a G. You are not allowed to move outside the grid. If you go to a square from which the goal is unreachable, you lose.

Input

A sequence of test cases. Each test case begins with a line containing the test case name. The next line has the form

M N

where the maze has M rows each with N columns. The next M lines each contain N instructions chosen from the ones above. The c'th instruction in the r'th line is the instruction for the square r:c.

```
2 <= M, N <= 1,000
sum M*N over all test cases <= 1,000,000
1 <= r <= M
1 <= c <= N
1 <= m <= M + N
```

In any test case there will be exactly one G, and it will be reachable from some boundary square.

Input ends with an end of file. The test case name line is at most 80 characters.

Output

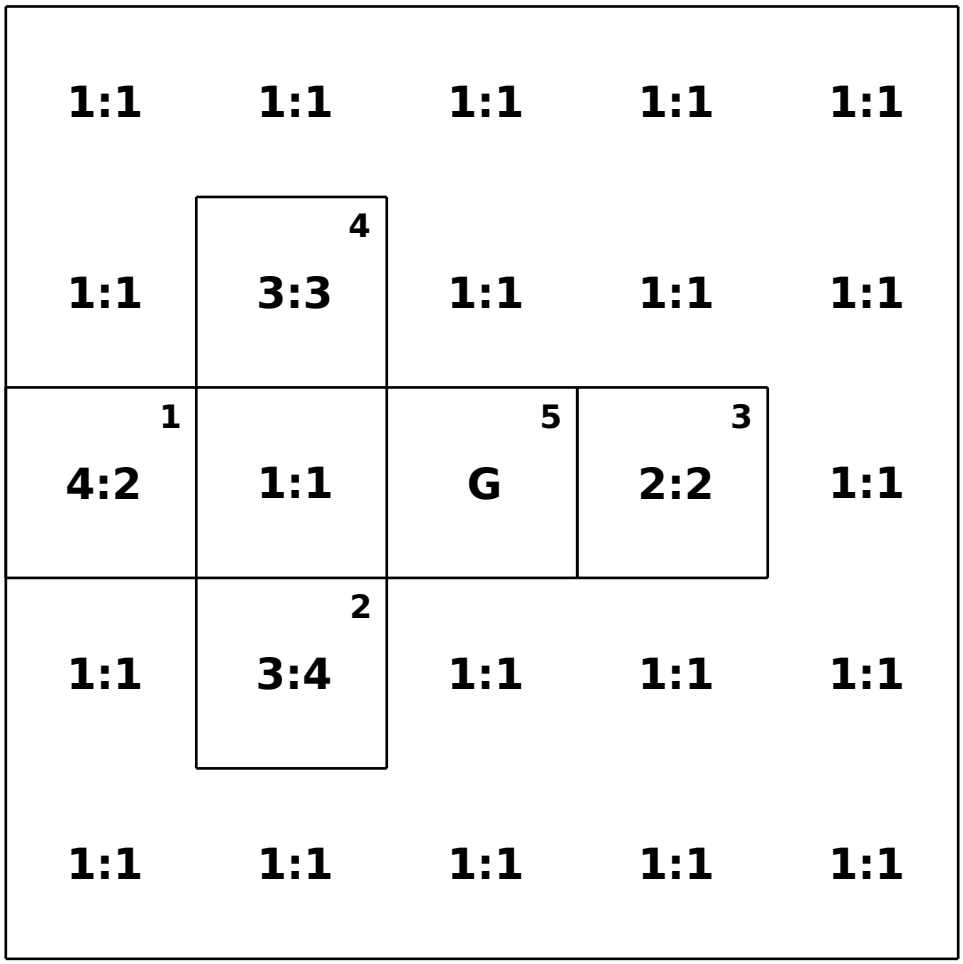
For each test case, first an exact copy of the test case name line. Then just one line containing the coordinates of your path to the goal, in the form 'r:c'. The first entry should designate the boundary square at which you enter the maze, and the last should designate the square containing G.

Input will be such that there is always a solution. YOUR SOLUTION SHOULD BE THE SHORTEST. If there are several shortest solutions, any one will do.

```
Display
-------

The input and output can be displayed in X-Windows or
printed by the commands

        display_maze sample.in [sample.test]
        print_maze sample.in [sample.test]

where sample.test is the output file corresponding to
sample.in, and these files can be replaced by any
test case input file and its corresponding output.
The output file can be omitted in which case only the
input will be displayed.


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

-- SAMPLE 1 --
5 5
1:1 1:1 1:1 1:1 1:1
1:1 3:3 1:1 1:1 1:1
4:2 1:1   G 2:2 1:1
1:1 3:4 1:1 1:1 1:1
1:1 1:1 1:1 1:1 1:1
-- SAMPLE 2 --
9 9
   3 4:8 5:5 4:5 6:6   3   X 2:9 5:2
   X 6:5 2:2 9:8 2:3 1:1   X   4   3
   3   X 8:1   4 2:6 7:6 1:2 9:7 5:9
 9:2 9:1   X   X 8:6 3:1 5:3 8:4   4
   4   X   4 8:8 4:6 8:3 9:8 1:1 4:2
   X 6:8 7:3   4   3   3   X 4:2   X
 3:7   4 9:7   4 4:2 3:2 4:5   3 9:8
 8:1   3 2:5 7:8 5:1 1:5 3:5   G 7:8
 6:5 3:5 9:2   3 5:3 8:4   3 7:5   4
```

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

-- SAMPLE 1 --
3:1 4:2 3:4 2:2 3:3
-- SAMPLE 2 --
1:5 6:6 5:4 8:8

File:      jmaze.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Mon Oct 14 01:00:37 EDT 2019

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

-- SAMPLE 1 --


1:1 1:1 1:1 1:1 1:1
4
1:1 3:3 1:1 1:1 1:1
1
5
3
4:2 1:1 G 2:2 1:1
2
1:1 3:4 1:1 1:1 1:1
1:1 1:1 1:1 1:1 1:1

-- SAMPLE 2 --

3	4:8	5:5	4:5	6:6 [1]	3	X	2:9	5:2
X	6:5	2:2	9:8	2:3	1:1	X	4	3
3	X	8:1	4	2:6	7:6	1:2	9:7	5:9
9:2	9:1	X	X	8:6	3:1	5:3	8:4	4
4	X	4	8:8 [3]	4:6	8:3	9:8	1:1	4:2
X	6:8	7:3	4	3	3 [2]	X	4:2	X
3:7	4	9:7	4	4:2	3:2	4:5	3	9:8
8:1	3	2:5	7:8	5:1	1:5	3:5	G [4]	7:8
6:5	3:5	9:2	3	5:3	8:4	3	7:5	4

Tree Search

Trax and Jax live in the tunnels inside the planet Pax. There are a lot of tunnels. But there is only one path through the tunnels from any point to any other point (no running around in circles possible).

Trax needs to locate Jax, and fortunately she has a locator device that can help. The device, however, does not simply tell Trax where Jax is. Instead Trax can suggest to the device a possible location for Jax, and the device will tell her whether Jax is there and if not, which direction from the suggested location Jax is.

Unfortunately the locator device uses a lot of power for each response, and can only answer a few queries.

In this problem you write a program that talks to the locator device until you suggest the location where Jax is or run out of power.

Your Input/Output

You write the program whose binary name is 'treesearch'. The locator is implemented by a separate program whose name is 'locator'. The 'locator' program runs your 'treesearch' program as a subprocess. The command that does this is

```
locator [-trace] treesearch ... <TEST-CASE-INPUT
```

where ... are arguments passed to treesearch (only used for debugging). You write queries to your standard output. The locator program reads these and writes a response so that you can read it from your standard input. The locator also gives you a tunnel map which you read from your standard input at the very beginning before any queries.

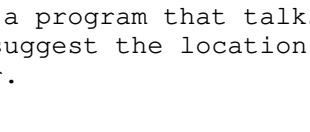
Note that YOUR program DOES NOT READ the test case input or write the test case output. The 'locator' program does this.

The tunnel map is a description of the tunnels viewed as a computer science tree with you at the root. The syntax is

```
tree ::= () | (tree-list)
tree-list ::= tree | tree tree-list
```

Here () represents a tree leaf with no children and (tree-list) is a tree whose children are the trees listed in tree-list. For example,

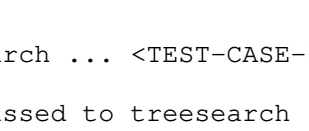
(()(()())) represents:



((()())()) represents:

In order to identify tree nodes we number them in the order that their '(' s occur in the tree map. Thus

(()(()())) represents:



((()())()) represents:

The locator begins by writing the tree map to your standard input. You may then make queries.

To make a query you output the number S of the tree node you are suggesting on a single line. The response is a single line that you can read which contains one of the following:

* just 'FOUND' if Jax is at the suggested node
* just 'POWER' if the locator has run out of power and you have failed to locate Jax (and your submission will be rejected)
* the number R of another tree node such that S and R are directly connected by a single tunnel and R is closer to Jax than S is; here distance between nodes is the number of tunnels needed to get from one node to the other node

If the tunnel map has N nodes, you are allowed Q queries before power runs out, where

```
2 <= N <= 1,000,000

Q = 1 + ceiling of log base 2 of N
```

All lines input to your program will have at most 80 characters, except the line containing the tunnel_map, which has 2*N characters. All query lines output from your program must have less than 80 characters.

WARNING

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

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

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 locator 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.

Debugging

If you specify -trace in the command that runs your treesearch program, your program will be run in trace mode.

In trace mode the locator outputs 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 '>> '. In trace mode, any line output by your program that begins with a '*' will be traced but otherwise ignored. You can use such lines for debugging output, and you can use arguments passed to your program to tell your program to output these lines.

The input tree, if not very large, may be printed or displayed in an X-window by the commands:

```
print_tree sample.in
display_tree sample.in
```

You can replace 'sample.in' by another locator input file. Jax's location is marked by a '*'.

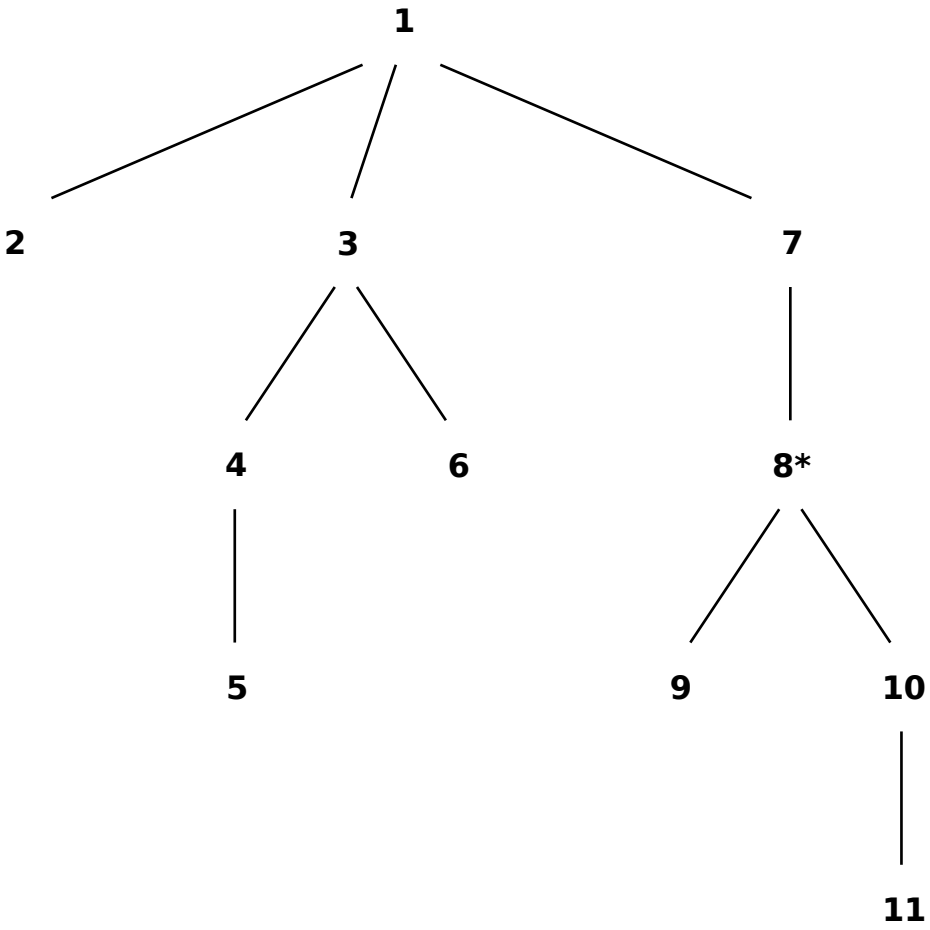
```
Test Case Descriptions And Output
---- ---- ------------ --- ------

In the test environment, the locator (and not your
program) reads test case descriptions from its standard
input and the locator (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

        N SEED

where N is the number of tree nodes and SEED is an
unsigned integer used to seed a pseudo-random number
generator that generates the tunnel map.  SEED
is 9 digits long: 10**8 <= SEED < 10**9.

The test case output written by the locator to its
standard output (not your program) consists of
first a line copying the test case name input line,
then any trace output, and finally a single line
containing either just 'FOUND' or 'POWER'.


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

-- SAMPLE 1 --
11 785983564
-- SAMPLE 2 --
35 352401876
```

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

-- SAMPLE 1 --
FOUND
-- SAMPLE 2 --
FOUND


Sample Output in Trace Mode (with -trace)
------ ------ -- ----- ---- ----- -------

-- SAMPLE 1 --
>> (()(((()))(((()(((((
<< 1
>> 7
<< 8
>> FOUND
FOUND
-- SAMPLE 2 --
>> (((((((())...rest omitted, see sample.trace
<< 8
>> 5
<< 26
>> 27
<< 28
>> FOUND
FOUND


File:      treesearch.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Oct 15 14:04:42 EDT 2019

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

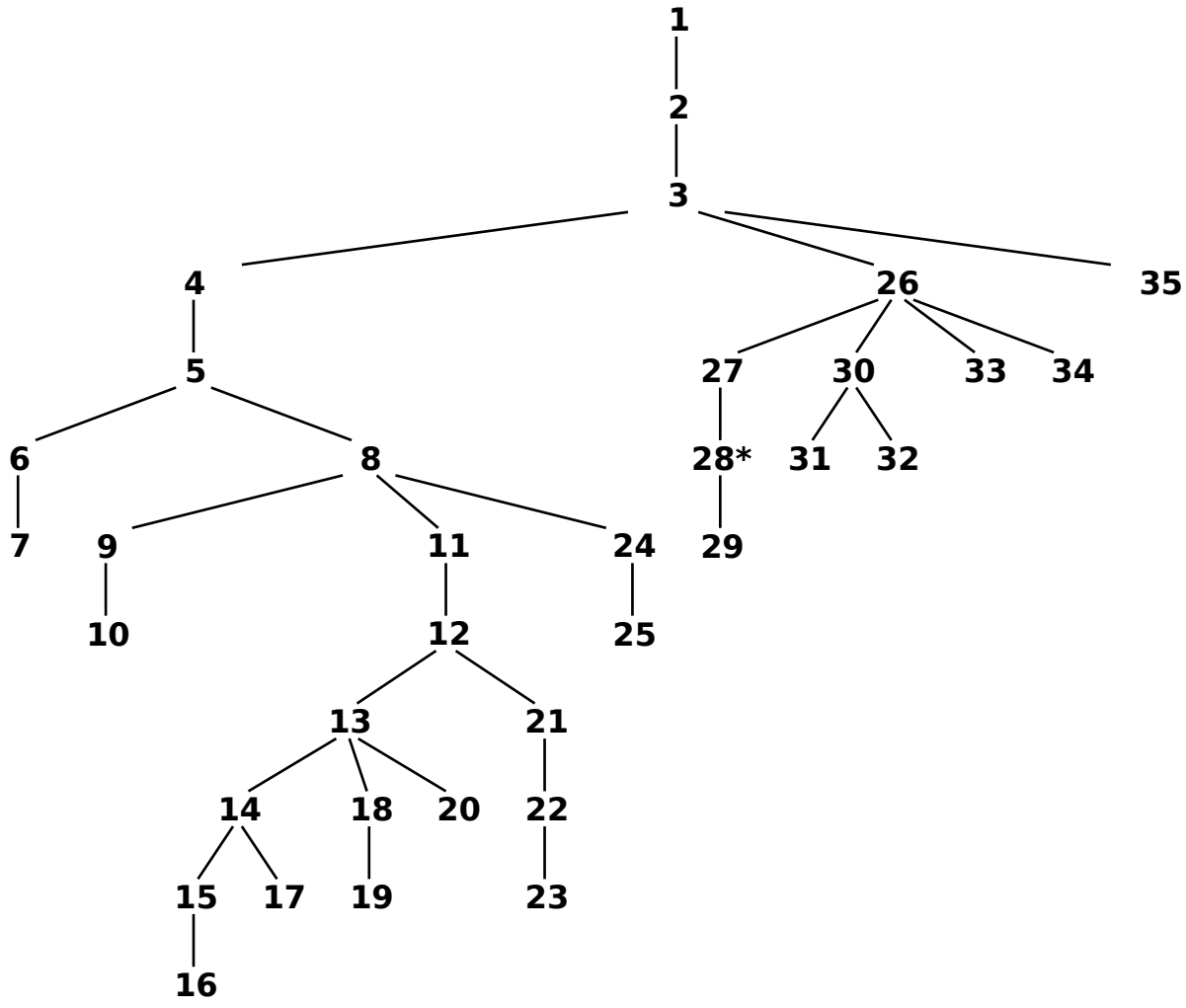
(This Page Intentionally Left Blank)

(This Page Intentionally Left Blank)

-- SAMPLE 1 --



-- SAMPLE 2 --


1
2
3
4
26
35
5
27
30
33
34
6
8
28*
31
32
7
9
11
24
29
10
12
25
13
21
14
18
20
22
15
17
19
23
16

Whoosh

Every ten years the city of Whoosh has a solo-pedal race from one end of the city to the other. Whoosh is known as a city that is just one big paved flat area in which solo-pedallers go whooshing around. But the race is made interesting because the buildings are in the way; solo-pedallers may not go into a building.

You have been asked by a friend who is participating to find a shortest path around the buildings.

Whoosh is laid out as a North-South East-West grid of squares, each square containing exactly one building. The corners of the squares are race waypoints, and by rule your route must go from waypoint to waypoint. In addition, any two consecutive waypoints on your route must be corners of the SAME square. If you go straight North, South, East, or West from a waypoint you will not go through a building. But if you go diagonally across a square, you may or may not have to go around the side of the building in the square. If you need to go around, your route may hug the side of the building.

The buildings in Whoosh are all ellipses, so solo-pedallers cannot injure themselves by running into sharp corners.

You have a map of Whoosh showing the buildings, so it should be simple.

WARNING: The sine an cosine functions take too long to be executed millions of times per test case, so they must be used sparingly.

Input

A sequence of test cases. Each test case begins with a line containing the test case name. The next line contains

```
M N L
```

specifying that the Whoosh grid has M rows of N squares each, and the square sides are each L feet long. The waypoint xy-coordinates range from (0,0) in the lower left to (L*M,L*N) in the upper right. The races starts at (0,0) and ends at (L*M,L*N).

Then there are M*N lines each specifying one building. Each of these lines has the form:

```
cx cy angle minor major
```

and describes a building whose footprint is an ellipse with center (cx,cy), given angle of major axis, and given lengths in feet of minor and major semi-axes. The angle is measured in degrees counter-clockwise from the positive x-axis in the usual way. A semi-axis is one half an axis, so if angle == 0 the equation of the ellipse would be:

```
((x-cx)/major)^2 + ((y-cy)/minor)^2 = 1
```

There is exactly one building for each grid square, but the building description lines are in arbitrary order.

All input numbers are integers.

```
     1 <= M,N            <= 20
 1,000 <= L              <= 10,000
     0 <= cy             <= M*L
     0 <= cx             <= N*L
- 180 <  angle       <=  + 180
    50 <= minor <= major <= L/2 - 50
```

No building part is within 50 feet of a North-South or East-West line running through any waypoint.

Input ends with an end of file. The test case name line is at most 80 characters.

Output

For each test case, first an exact copy of the test case name line. Then just one line giving the length of the shortest route that obeys the rules. The length should be accurate to 1 part in 10^5.

Assume the size of a solo-pedal is negligible so if the route includes part of a building perimeter, only the exact length of that perimeter part counts as part of the length.

Sample Input

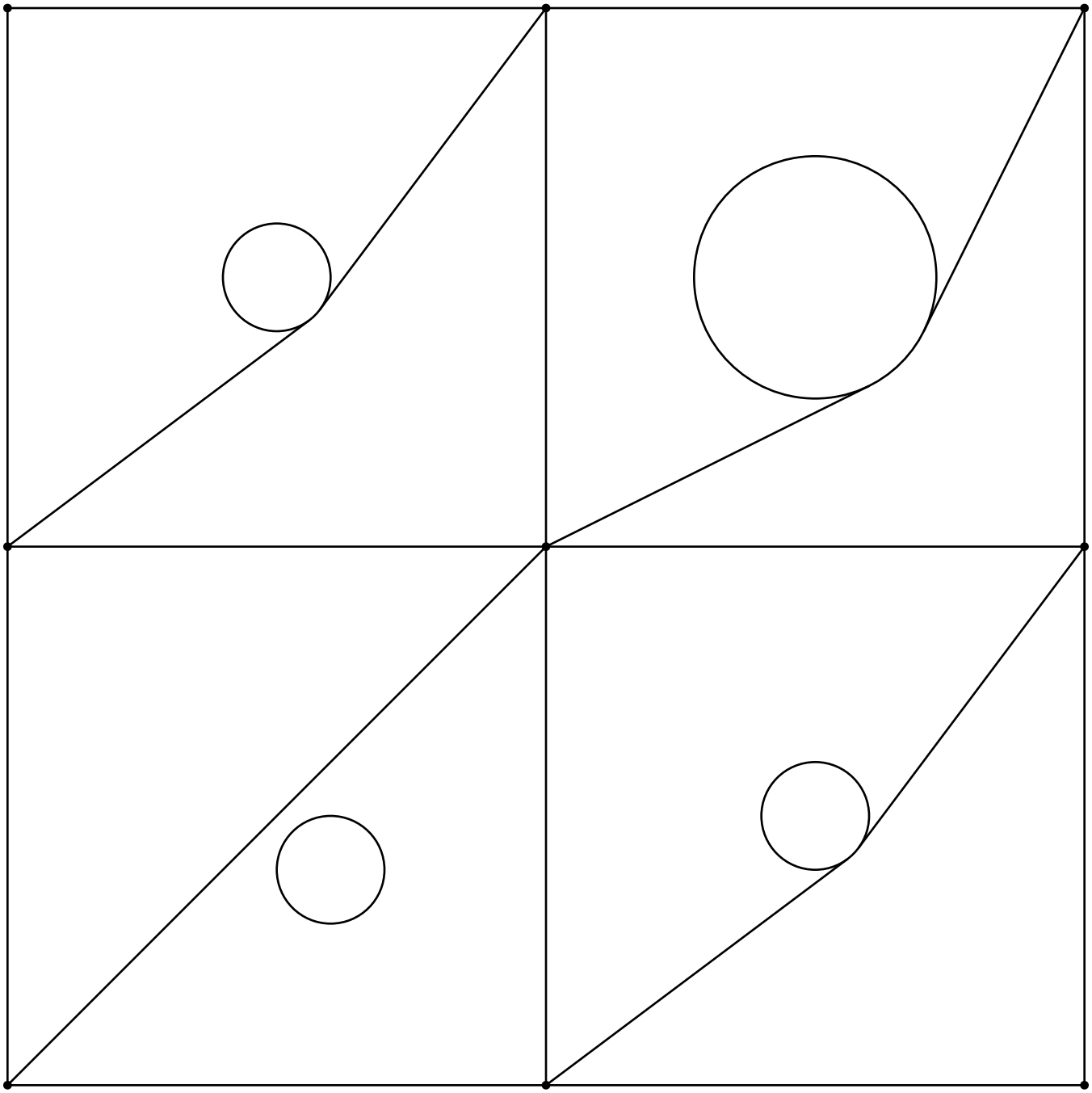
```
-- SAMPLE 1 --
2 2 1000
 600  400 0 100 100
1500  500 0 100 100
 500 1500 0 100 100
1500 1500 0 225 225
-- SAMPLE 2 --
2 3 1000
 500  500   0 100 150
1500  500   0 150 200
2500  500  45  75 200
 500 1510  30 100 150
1510 1500  60 150 200
2510 1510 120  75 200
```

Sample Output

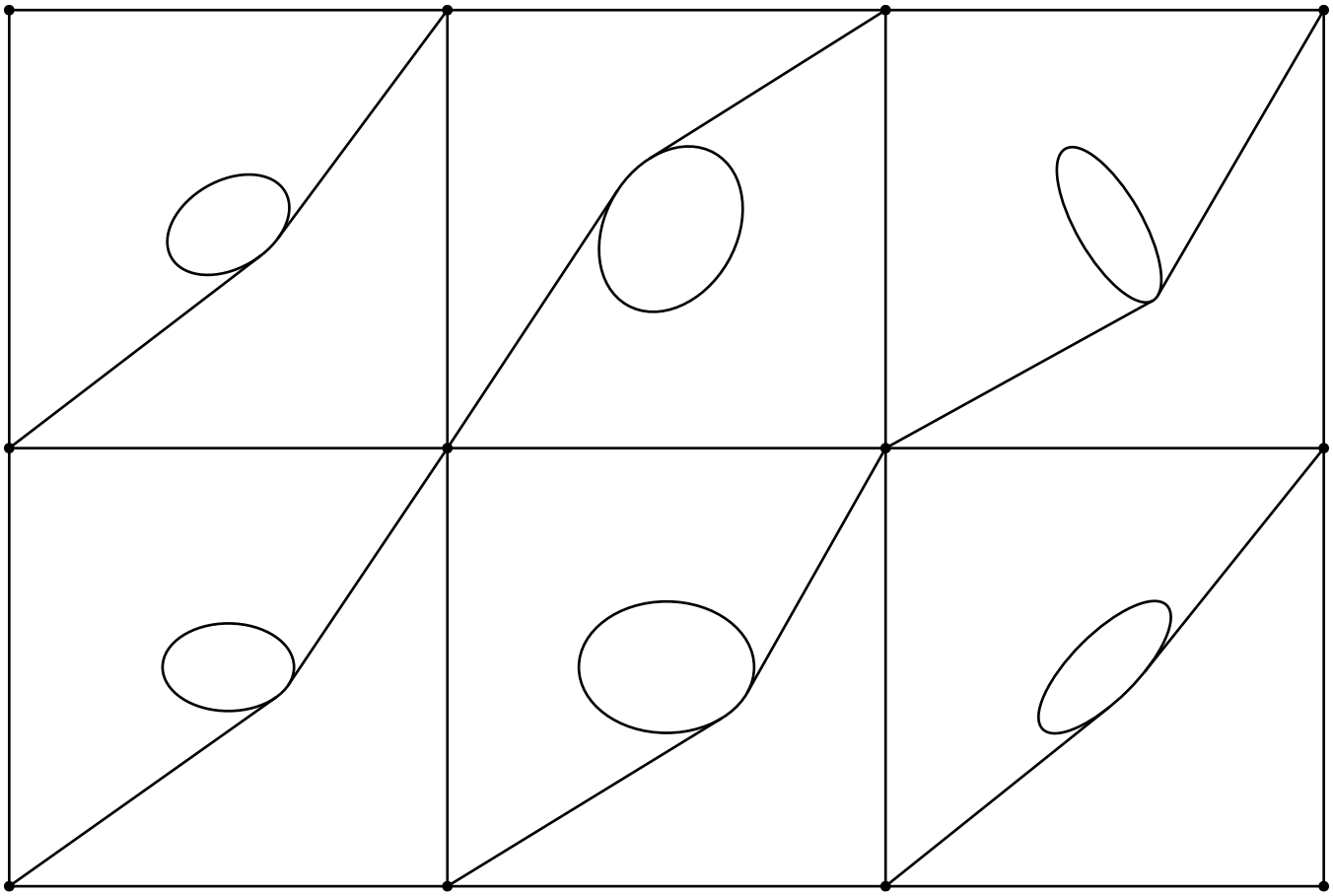
```
-- SAMPLE 1 --
2900.64
-- SAMPLE 2 --
3882.51
```

```
File:      whoosh.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Oct 15 14:09:19 EDT 2019
```

-- SAMPLE 1 --



-- SAMPLE 2 --



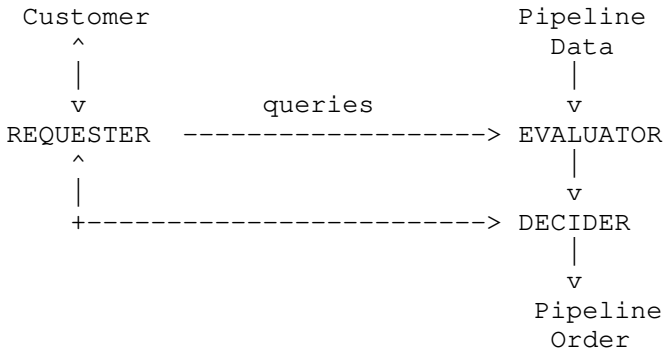
Flow Management
---- ----------

You work for a company that manages the flow of gasoline from refineries in the Southwest to depots in the Northeast.

The year is 2473, and things have changed. Gone are all the large companies, and all companies are very small. There are thousands of small companies that each operate a single pipeline from one depot to another, with depots all across the country. Each charges its own price per gallon for transport from the beginning of its pipe to the end of its pipe.

The good news is that everything is computerized, so your customers put orders into your computer, your computer puts orders into pipeline company computers, and pipeline company computers turn valves on and off to make things happen. Also, there is no minimum charge for using a pipeline, and the charge is linear in the amount of gasoline per hour, because gasoline is actually batched in tanks at depots and sent down a pipeline in batches, so the cost per gallon is held constant by twiddling the size of the batches.

You are your company's software department, and as we said, everything is done automatically by computer. So in a sense, you company is just a computer program. Its called FLOWMAN, the flow manager. Its block diagram is:



Your immediate job is to write a new and better EVALUATOR. This is given queries of the form:

> What is the maximum flow rate of gasoline in gallons per hour that we can achieve from depot QS to depot QD if we spend up to QC dollars per hour?

Given such a query and the current pipeline data, your EVALUATOR must return a number that is the answer to the query. As some point the DECIDER will place an order based on your answers to several queries, and then the pipeline operators will change the pipeline data to reflect the new order.

Input

A sequence of test cases. Each test case begins with a line containing the test case name. The next line contains

D P Q

where D is the number of depots, P the number of pipelines, and Q the number of queries. Depots have ID numbers 1, 2, ..., D.

Then next P lines each describe one pipeline and have the form

PS PD PC PP

where PS is the source depot ID of the pipeline, PD is the destination depot ID, PC is the capacity of the pipeline given as in integer in gallons per hour, and PP is the price in dollars per gallon to the nearest hundreth of a cent (0.0001 dollars).

Pipelines only work in one direction and cannot maintain a flow greater than their capacity. But they can work with any flow value less than their capacity, including NON-INTEGRAL numbers of gallons per hour.

The next Q lines each describe one query and have the form

QS QD QC

where QS is the source depot ID, QD is the destination depot ID, and QC is the maximum expenditure rate in dollars per hour, given as an integer.

For each query you must compute a network flow such that the net flow into every depot but QS and QD is zero, as the flow must be sustained indefinitely. The net flow out of QS must equal the net flow into QD, and this must be maximized given the pipeline capacities and the maximum total expenditure constraint QC.

2 <= D <= 1,000
1 <= P <= 1,000
1 <= Q <= 10
1 <= PC <= 500
0.0100 <= PP <= 1.0000
1 <= QC <= 100,000

sum D*P*Q over all test cases <= 1,000,000

Input ends with an end of file. The test case name line is at most 80 characters.

Output

For each test case, first an exact copy of the test case name line. Then for each query, just one line giving the maximum flow possible, in gallons per hour, accurate to 0.1 gallon. Each query is evaluated independently of the other queries.

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

-- SAMPLE 1 --
4 5 3
1 2 100 0.1000
1 3  50 0.3000
2 4 100 0.2000
2 3  50 0.0300
3 4  50 0.0700
1 4  10
1 4  25
1 4  40
-- SAMPLE 2 --
4 5 2
1 2 100 0.1000
2 3 100 0.2000
3 4 100 0.1000
2 1  50 0.0100
4 3  50 0.0200
1 4 25
4 1 25


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

-- SAMPLE 1 --
50.00
100.00
131.91
-- SAMPLE 2 --
62.50
0.00

[See sample.flow for pipe flows]
```

```
File:      flowman.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Oct 15 02:59:55 EDT 2019

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

(This Page Intentionally Left Blank)

(This Page Intentionally Left Blank)

Railgun

Its URGENT! You MUST get a message to the Princess!

She's in the station on the other side of the twin suns, and because of the suns, you cannot get an electromagnetic or photonic message through to her. You must use a message capsule fired from a rail gun.

The velocity of the capsule just after it is fired from the rail gun is fixed. The good news is that both you and the Princess's station are holding constant position relative to the suns, and the velocity is fast enough that the path of the message capsule will be close to the straight line between you and the Princess.

The suns are of course rotating about each other, and as you and the Princess are fixed relative to the suns, the coordinate system you are using is rotating with the suns. Therefore there are Coriolis, centrifugal, and Euler forces that are not negligible. However, you are to IGNORE THESE (treat them as zero), as they significantly slow the computation and thus make judging more difficult.

Input

A sequence of test cases. Each test case has four lines of the form:

```
TEST-CASE-NAME
M1 X1 M2 X2
X3 Y3 Z3 X4 Y4 Z4
V D
```

The first line is a test case name line, that has at most 80 characters. The other lines contain the floating point numbers defined as follows:

- M1 is the mass of sun 1, and (X1,0,0) is its center
- M2 is the mass of sun 2, and (X2,0,0) is its center
- (X3,Y3,Z3) is your position
- (X4,Y4,Z4) is the Princess's station position
- V is the capsule velocity when it leaves the rail gun
- D is the distance within which the capsule must come to the Princess's station in order for the station to capture the capsule

Units are kilograms, meters, meters per second.

Coordinates are chosen so that (0,0,0) is the center of gravity of the 2-sun system, that is,

$$M1*X1 + M2*X2 = 0$$

Acceleration caused by the gravity of a sun is

$$G*M/R**2$$ pointed at the center of the sun

where M is the mass of the sun, R the distance to the sun, and

$$G = 6.67408e-11\ m**3/(kg*sec**2)$$

All spacial coordinates are in the range [-1e15,+1e15] meters and masses are in the range [1e30,1e32] kilograms. V will be large enough to get the capsule to the Princess within 1e4 seconds (3 hours). D will be in the range [1e3,1e5].

Input ends with an end of file.

Output

For each test case, first an exact copy of the test case name line. Then just one line of the form

```
VX VY VZ
```

giving the initial velocity of capsule. To be judged correct, the numbers should be output with 10 digits of precision.

The capsule must go to the Princess's station without going completely around either sun. The capsule must come within D meters of the Princess's station, and the total initial velocity, sqrt (VX**2 + VY**2 + VZ**2), must be within 0.0001% of V.

There will of course be many slightly different solutions; any one will do.

Display

There is an interactive display command that may help you visualize the problem. To see its documentation use

```
display_trajectory -doc
```

and to run it use

```
display_trajectory sample.in
```

replacing sample.in with any test case input file.
The display command requires X windows.

Sample Input

```
-- SAMPLE 1 --
1e30 -1e10   1e30 +1e10
2e10 1e10 +2.0e9   0 -1e10 +5.0e9
5e6 1e4
-- SAMPLE 2 --
1e30 -1e10   1e30 +1e10
-0.99e10 1e10 0   -0.99e10 -1e10 0
1e7 1e4
```

Sample Output

```
-- SAMPLE 1 --
-3515346.385 -3515696.573 531241.3722
-- SAMPLE 2 --
45647.43196 -9999895.815 0.01604674784
```

File: railgun.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Tue Oct 15 05:05:25 EDT 2019

Triangle Survey
-------- ------

You are the chief programmer for the Triangle Survey Company, and you have a problem. Your company surveys very large properties to accurately locate points on the property, and half the data for a recent job has gone missing.

Your survey team did its usual thing: it organized the points into a simple polygon whose vertices are points to be located with all the remaining points to be located inside the polygon, and then triangulated the polygon so that the set of all triangle vertices was exactly the set of points to be located. Then the team measured the lengths of all the triangle edges precisely. Lastly the team should have reported:

1) A length list of triples <I,J,L> each specifying that the distance from point I to point J is L.

2) A triangle list of triples <I,J,K> each specifying that points I, J, K are vertices of a triangle and the order I, J, K is clockwise order.

The team reported the length list as usual, but then the team LOST the triangle list.

You are being asked to find all the point locations from the length list, without knowing the triangle list a priori.

Input

A sequence of test cases. Each test case begins with a line containing the test case name. The next line has the form

P E

where P is the number of points and E is the number of lengths (triangle edges) measured. Points are numbered 1, 2, ..., P.

Next come E length list lines each of the form

I J L

where I and J are numbers of distinct points connected by a triangle edge and L is the length of the edge.

Lengths are positive floating point numbers.

Points 1 and 2 are at the front of the property, and the first length line has the form:

1 2 L12

You are to assign point 1 the coordinates (0,0) and point 2 the coordinates (0,L12). All other points are to be assigned Y-coordinate > 0, i.e., the property is in a half-plane bounded by its front, and you are to put the property points in the upper half-plane.

In addition, to simplify scoring this problem, the X and Y coordinates of all points have been chosen to be integers.

3 <= P <= 10,000
E <= 3*P - 3
1 <= L <= 100,000

The test case name lines are at most 80 characters. Input ends with an end of file.

The input will be such that all the point X and Y values are integers such that

```
- 30,000 <= X <= + 30,000
       0 <= Y <= + 30,000
```

where Y == 0 only for points 1 and 2
and X == 0 for point 1 (and maybe other points)

Output

For each test case, first an exact copy of the test case name line. This is followed by P lines of the form

```
X Y
```

where the I'th such line contains the integer coordinates (X,Y) of point I. The first of these lines is therefore '0 0' and the second is 'L12 0'.

X and Y must be exactly accurate (as integers).

Display

You can display or print test cases with the commands:

```
display_survey sample.in sample.test
print_survey sample.in sample.test
```

where sample.in/sample.test can be replaced by any test case input file and its corresponding output file.

Sample Input

```
-- SAMPLE 1 --
5 8
         1         2          100.000000
         2         3          100.498756
         3         4           22.360680
         4         1          142.126704
         5         1           64.031242
         5         2           64.031242
         5         3           84.852814
         5         4           80.622577
```

[See sample.in for more sample input.]

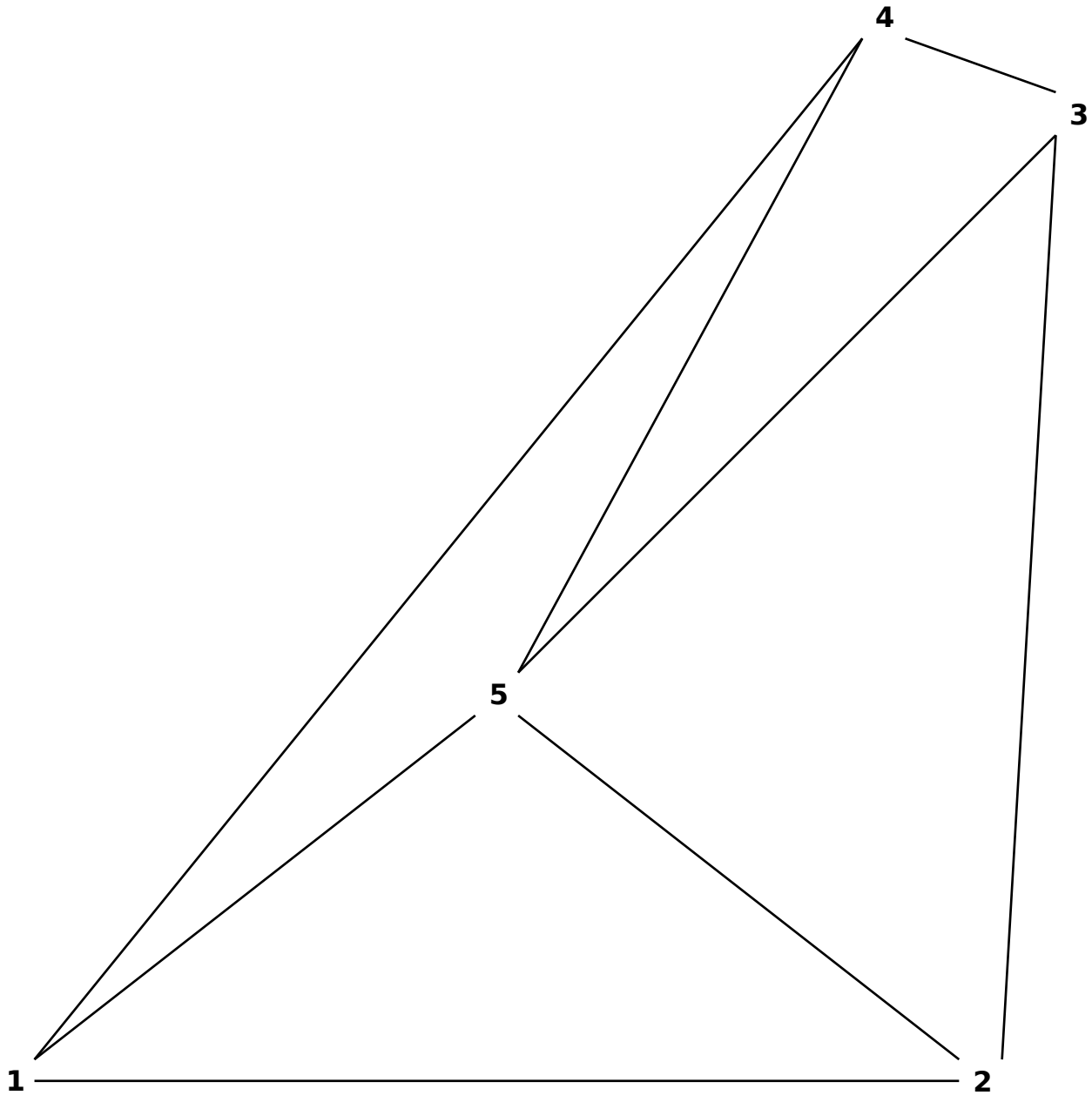
Sample Output

```
-- SAMPLE 1 --
         0         0
       100         0
       110       100
        90       110
        50        40
```

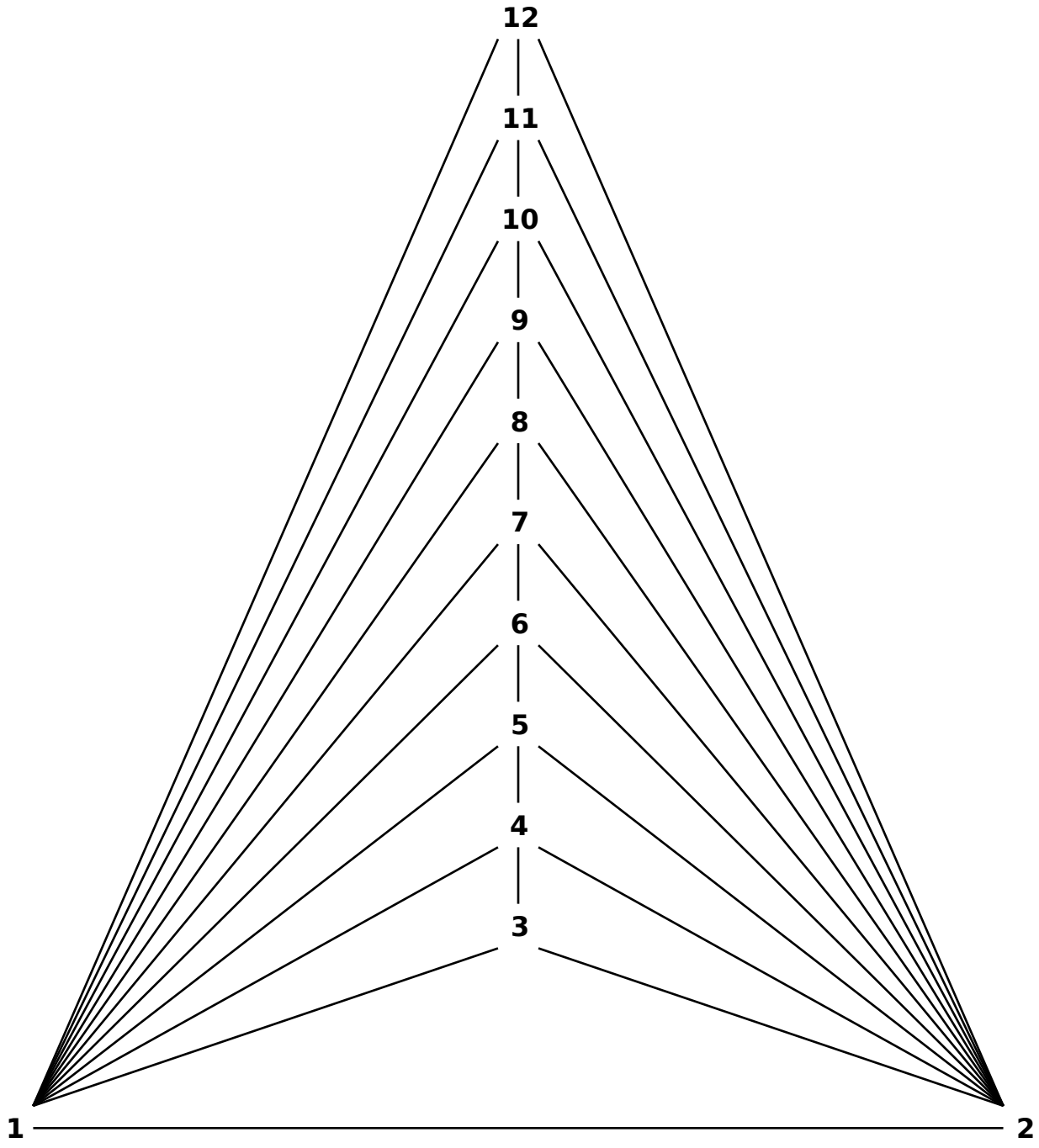
[See sample.test for more sample output.]

File: trisurvey.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Tue Oct 15 14:50:52 EDT 2019

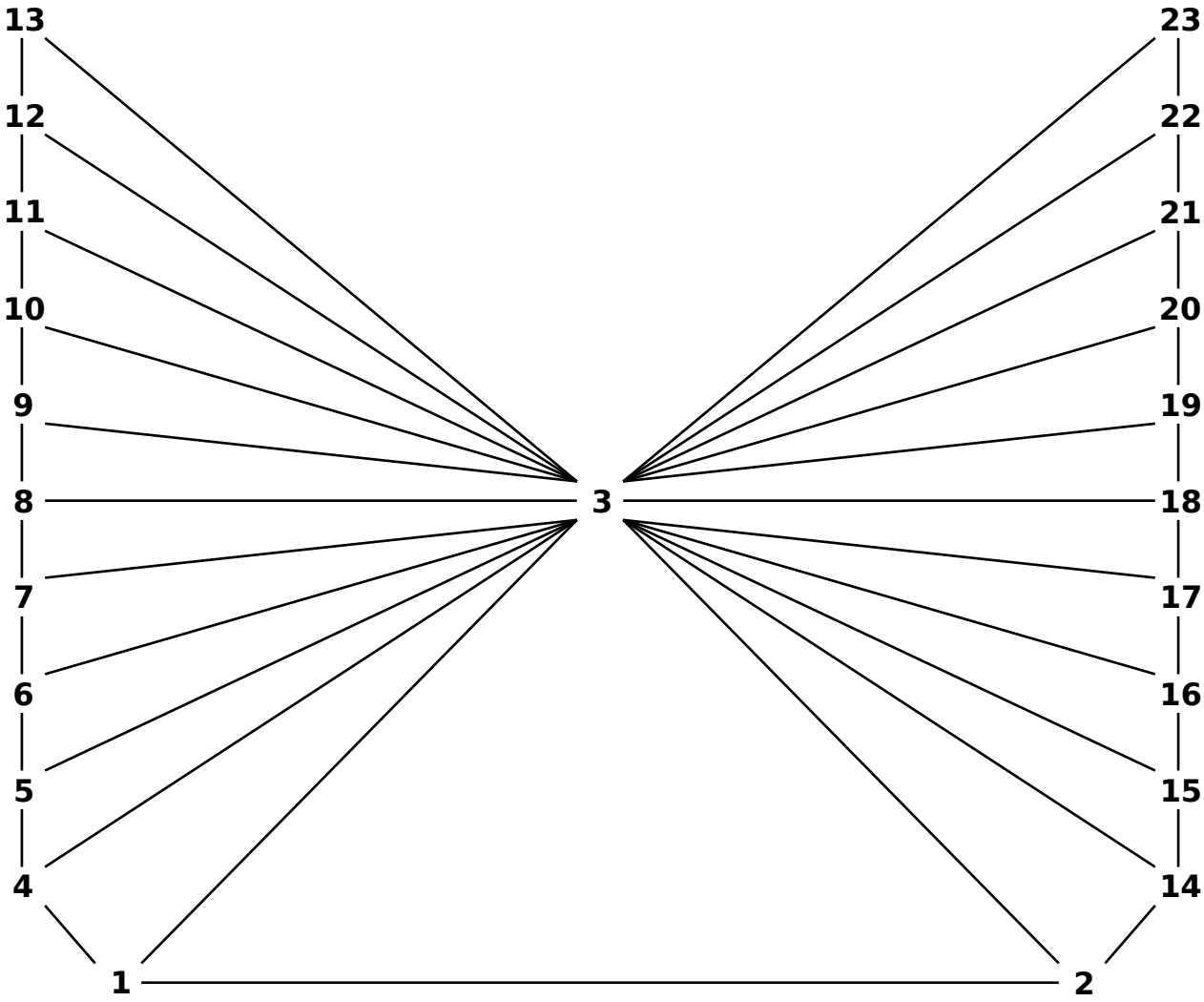
-- SAMPLE 1 --



-- SAMPLE 2 --



-- SAMPLE 3 --



-- SAMPLE 4 --