Problems Index Fri Oct 05 07:08:46 EDT 2018

BOSPRE 2018 PROBLEMS
------ ---- --------

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

This year we have introduced generate programs for many problems: see ‘Generate Programs’ in ACM Contest Help.

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/permutation
Match me if you can.

problems/positive
Don’t let factions get in the way.

problems/goodpath
The high and low of it all.

problems/dropstick
Aims the thing.

problems/alignment
Shifty is good sometimes.

problems/maxparallel
What if you had a million lives?

problems/flowsecurity
What will sabotage do to us?

problems/driving
Dividing up transportation labor.

problems/overlap
Counting for the numerous.

problems/walls
Are you stuck in a corner?

```
Permutation
-----------

Given two equal length strings of upper case letters,
find a permutation that maps the first string to the
second string.


Input
-----

For each of several test cases, first a line that gives
the test case name, and then two lines each containing
one of the two strings.  All lines are at most 80 char-
acters long.  Input ends with an end of file.


Output
------

For each test case, first an exact copy of the test case
name line.  Then N lines where N is the common length of
the input strings.  Each of the N lines has the form:

        L: I -> J

where L is a letter that is in position I in the first
string and position J in the second string.  Positions
are numbered 1, 2, ..., N.  Each position must appear
EXACTLY ONCE as I on the N lines, and also exactly once
as J in the N lines, so that the N lines define a
permutation (i.e., a 1-1 map).

If this is not possible, output a single line contain-
ing just the word ‘impossible’ in lower case letters,
in place of the N lines.

For example, if the input is:           ABC
                                        ABA

the output:                             A: 1 -> 1
                                        B: 2 -> 2
                                        A: 1 -> 3

is incorrect because 3 does not appear as I in any of
the 3 lines, and also because 1 appears as I more than
once.  In fact, as the second string has more A’s than
the first string, there are no permutations that map
the first string to the second string, and the correct
answer is ‘impossible’.

There may be more than one correct solution.  If so,
give only one solution.


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

-- SAMPLE 1 --
ABC
CBA
-- SAMPLE 2 --
ABC
ABA
-- SAMPLE 3 --
ABCDEFG
GFEDCBA
-- SAMPLE 4 --
ABABAB
BBBAAA
-- SAMPLE 5 --
ABABAB
BBBAAC
[see sample.in for more sample input]
```

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

-- SAMPLE 1 --
A: 1 -> 3
B: 2 -> 2
C: 3 -> 1
-- SAMPLE 2 --
impossible
-- SAMPLE 3 --
A: 1 -> 7
B: 2 -> 6
C: 3 -> 5
D: 4 -> 4
E: 5 -> 3
F: 6 -> 2
G: 7 -> 1
-- SAMPLE 4 --
A: 1 -> 4
B: 2 -> 1
A: 3 -> 5
B: 4 -> 2
A: 5 -> 6
B: 6 -> 3
-- SAMPLE 5 --
impossible
[see sample.test for more sample output]
```

File:      permutation.txt
Author:    Shai Simonson <shai@stonehill.edu>
Editor:    Bob Walton <walton@seas.harvard.edu>
Date:      Fri Oct  5 21:35:32 EDT 2018

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

Diophantine Equations With Positive Solutions
----------- --------- ---- -------- ---------

You are to find a strictly positive integral solution (x,y) to

```
        A*x + B*y = C
```

given that A, B, C are strictly positive integers.

Polynomial equations with integer coefficients in which only integer solutions are sought are called Diophantine equations, so we are asking for positive solutions of one of the simplest Diophantine equations.

Input
-----

For each of several test cases, a single line of the form:

```
        A B C
```

where

```
        10 <= A,B <= 1,000
        10 <= C <= 1,000,000
```

Input ends with an end of file.

Output
------

For each test case, one line of the form

```
        A * x + B * y = C
```

where all the letters are replaced by non-zero positive integers so as to make the equation true. The integers replacing A, B, C are taken from the test case input line. You must find the solution values of x and y.

Input will be such that there is a unique solution.

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

```
10 25 100
29 17 300
33 66 99
128 241 34647
128 241 34648
128 241 34649
113 197 21952
113 197 21953
113 197 21954
113 197 21955
113 197 30001
```

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

10 * 5 + 25 * 2 = 100
29 * 8 + 17 * 4 = 300
33 * 1 + 66 * 1 = 99
128 * 137 + 241 * 71 = 34647
128 * 105 + 241 * 88 = 34648
128 * 73 + 241 * 105 = 34649
113 * 67 + 197 * 73 = 21952
113 * 135 + 197 * 34 = 21953
113 * 6 + 197 * 108 = 21954
113 * 74 + 197 * 69 = 21955
113 * 133 + 197 * 76 = 30001
```

File:      positive.txt
Author:    Shai Simonson <shai@stonehill.edu>
Editor:    Bob Walton <walton@seas.harvard.edu>
Date:      Thu Oct  4 20:12:03 EDT 2018

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

```
Finding A Good Path
------- - ---- ----

Given a 2-dimensional array representing elevations of
a 2-dimensional map, determine the lowest and highest
points and whether there is a path between them that
does not go both up and down.


Input
-----

For each test case, first a line that gives the test
case name.  Then a line of the form

        M N

giving the number of rows (M) and columns (N) in the
array.  Then M lines each containing one row of N num-
bers, which are the elevations at the corresponding map
point.  The rows are numbered 1 through M from top to
bottom, and the columns are numbered 1 through N from
left to right.

        5 <= M,N <= 100
        0 <= array element <= 100

The test case name line is 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 a single line of the form:

        Lr Lc Hr Hc YESNO

where (Lr,Lc) is the (row,column) of the lowest point in
the array, (Hr,Hc) is the (row,column) of the highest
point in the array, and YESNO is the word:

    yes   If there is a path from the lowest point to
          the highest point that never goes down.

    no    If there is no such path.

Specifically, the path starts at the lowest point and
can only go from a point to one of its 4-neighbors
(left, right, up, or down) that does NOT have a lower
elevation than the point.

Indices in the array begin with 1, so

        1 <= Lr,Hr <= M
        1 <= Lc,Hc <= N

Input will be such that there is a unique solution.
```

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

-- SAMPLE 1 --
5 5
2 3 3 4 5
2 4 4 1 4
2 1 0 2 1
4 2 1 3 4
3 1 2 2 3
-- SAMPLE 2 --
5 5
2 3 3 4 5
3 4 4 1 4
2 1 0 2 1
4 2 1 3 4
3 1 2 2 3


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

-- SAMPLE 1 --
3 3 1 5 yes
-- SAMPLE 2 --
3 3 1 5 no
```

File:      goodpath.txt
Author:    Shai Simonson <shai@stonehill.edu>
Editor:    Bob Walton <walton@seas.harvard.edu>
Date:      Sat Oct  6 02:48:31 EDT 2018

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

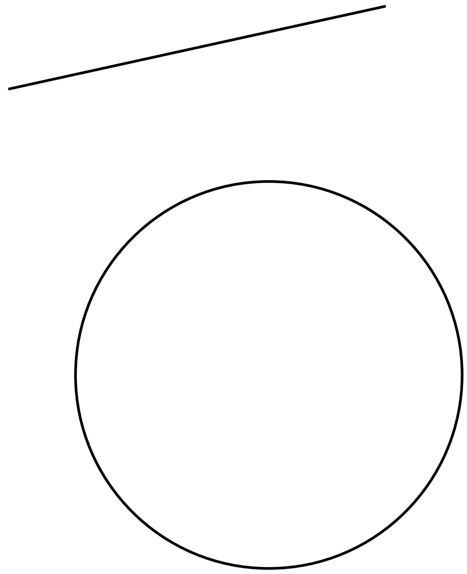
```
Drop Stick
---- -----

Tiny Joe and his friends play a game they call Drop
Stick.  They go to a high place and drop a straight
stick to see if they can hit a circular target far
below them.

However, Tiny Joe and his friends live in the year
2057 and all games are in virtual reality.  So to
play this game, they got you to program it.

You have available subroutines to create the high and
low places and define the circle and even drop the
stick.  But you need a new subroutine to determine
if the dropped stick touches the circle.

The subroutines you are given tell you the diameter
of the circle and the distances from its center of
the two stick ends after the stick has been dropped.
You also know the length of the stick.

You must determine if the stick touches any part of
the circle.


Input
-----

For each test case, one line of the form

        R L D1 D2

giving the radius R of the circle, the length L of the
stick, and the distances D1 and D2 of the two endpoints
of the stick from the center of the circle.  Numbers are
floating point.

    1 <= R,L <= 1,000
    0 <= D1,D2 <= 10,000
    D1 + D2 >= L
    D1 + L >= D2
    L + D2 >= D1


Input ends with an end of file.


Output
------

For each test case, one line containing

        R L D1 D2 ANS

where R, L, D1, D2 are copied from the input and must be
printed with at least 3 decimal places, and ANS is
either ‘TOUCH’ or ‘NO-TOUCH’.

The input will be such that the distance from the center
of the circle to the nearest point on the stick will
differ from R by at least 0.001.
```

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

100 200 200 200
100 200 110 110
100 200 150 150
100 200 140 140
1 2 1.412 1.412
1 2 1.417 1.417
1 1.4142135 1 1
1 3 1.1 3
1 2 1.1 1.1
1 2 0.9 2
1 2 1.02 2
1 2 1.1 2


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

100.000 200.000 200.000 200.000 NO-TOUCH
100.000 200.000 110.000 110.000 TOUCH
100.000 200.000 150.000 150.000 NO-TOUCH
100.000 200.000 140.000 140.000 TOUCH
1.000 2.000 1.412 1.412 TOUCH
1.000 2.000 1.417 1.417 NO-TOUCH
1.000 1.414 1.000 1.000 TOUCH
1.000 3.000 1.100 3.000 NO-TOUCH
1.000 2.000 1.100 1.100 TOUCH
1.000 2.000 0.900 2.000 TOUCH
1.000 2.000 1.020 2.000 TOUCH
1.000 2.000 1.100 2.000 NO-TOUCH
```

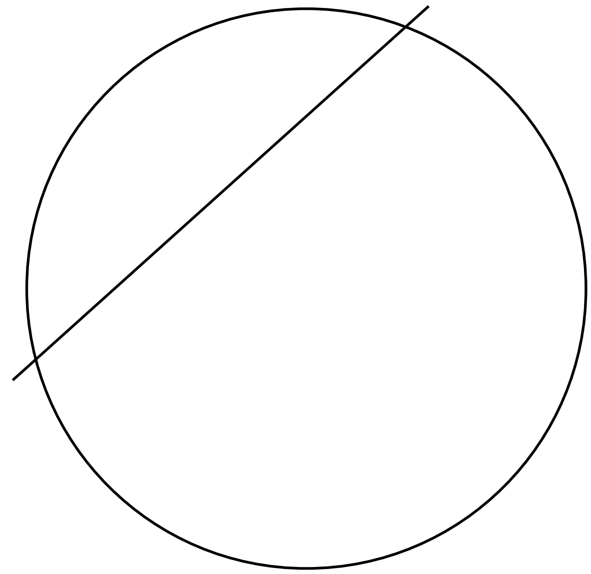
```
Display
-------

The commands

        display_dropstick -X dropstick.out

        display_dropstick -X dropstick.in

may be useful.  The command ‘display_dropstick -doc’
documents the display_dropstick program.


File:      dropstick.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sat Oct  6 02:52:02 EDT 2018
```

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

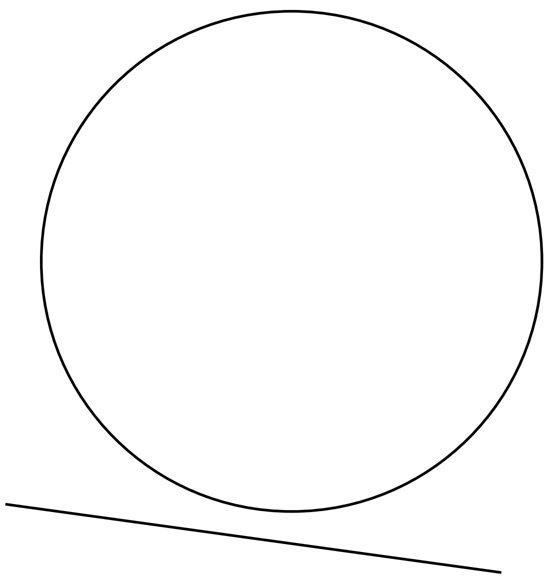
**100.000 200.000 200.000 200.000 NO-TOUCH**



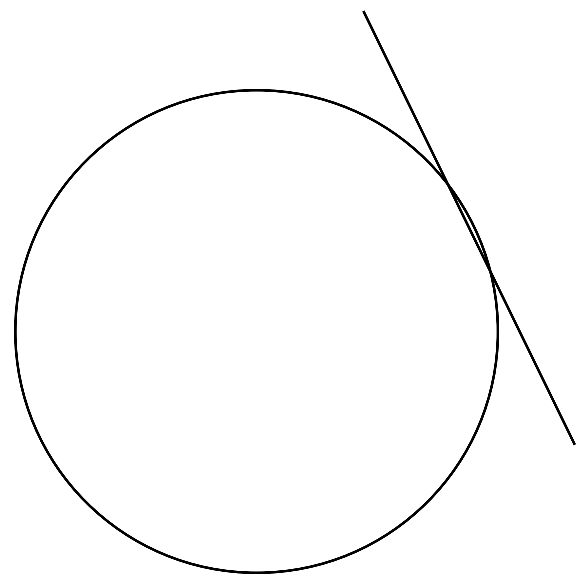
**100.000 200.000 110.000 110.000 TOUCH**



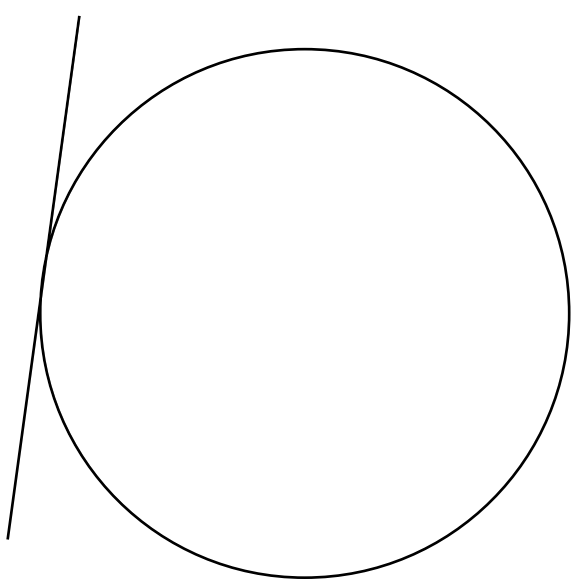
**100.000 200.000 150.000 150.000 NO-TOUCH**



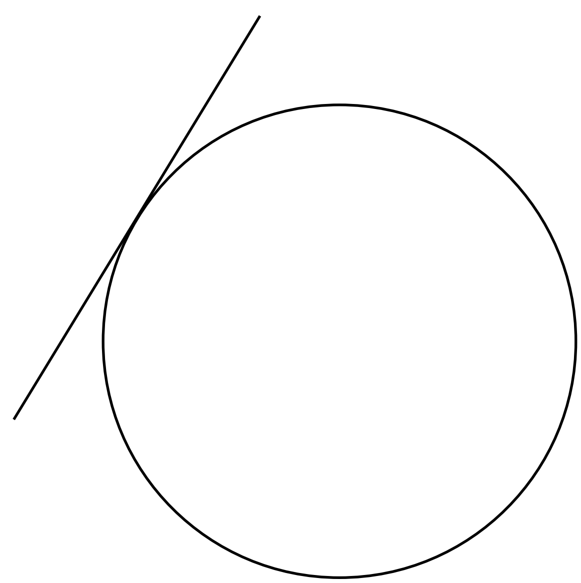
**100.000 200.000 140.000 140.000 TOUCH**



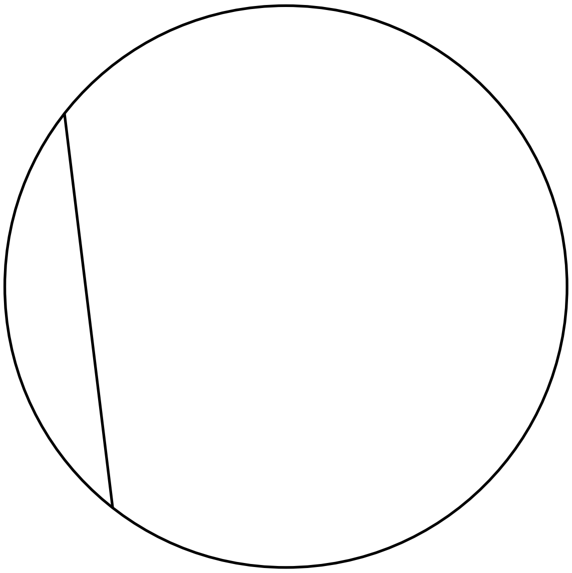
**1.000 2.000 1.412 1.412 TOUCH**



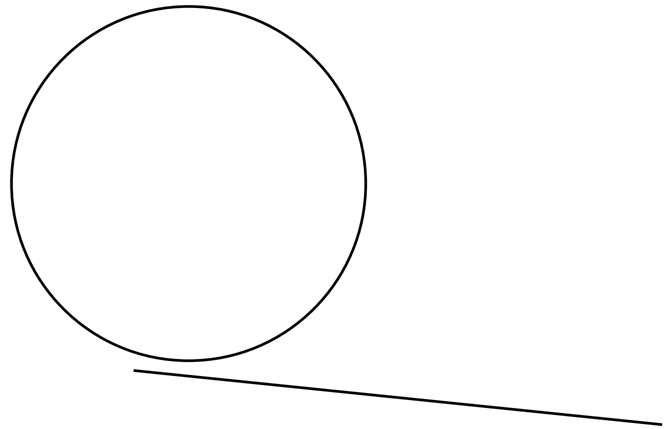
**1.000 2.000 1.417 1.417 NO-TOUCH**



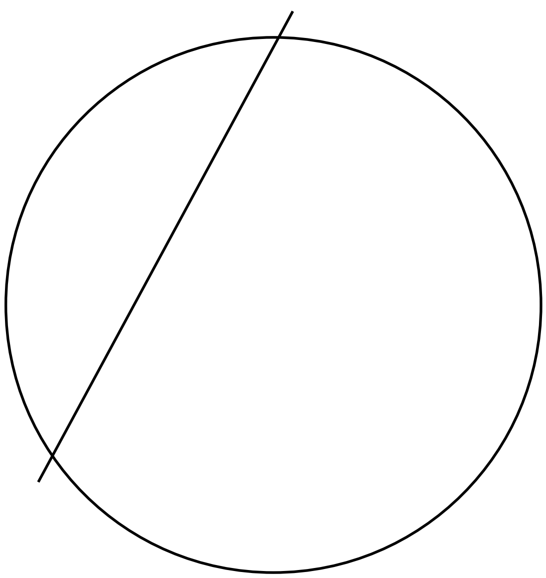
**1.000 1.414 1.000 1.000 TOUCH**



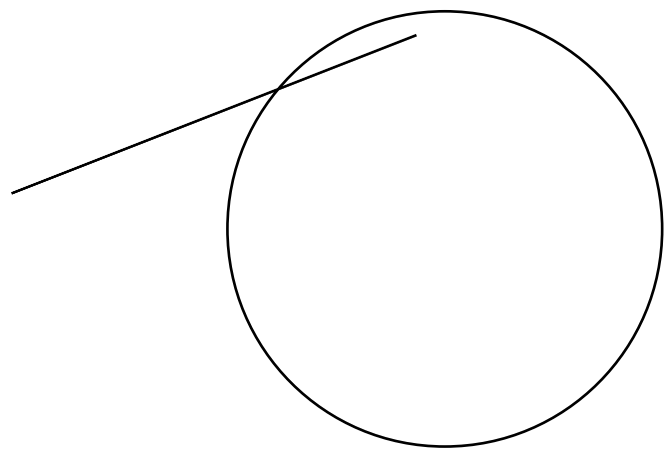
**1.000 3.000 1.100 3.000 NO-TOUCH**



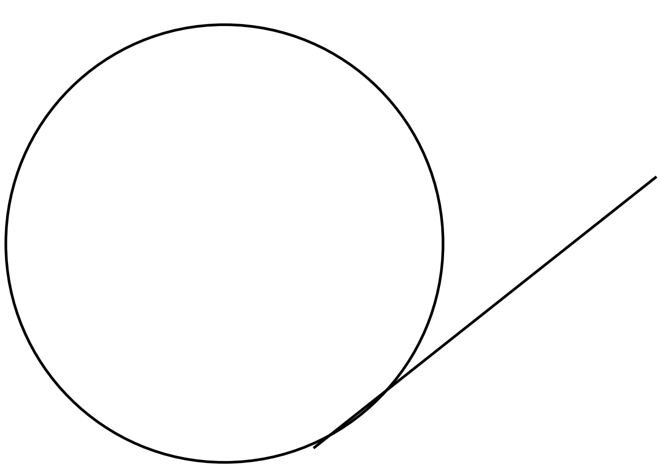
**1.000 2.000 1.100 1.100 TOUCH**



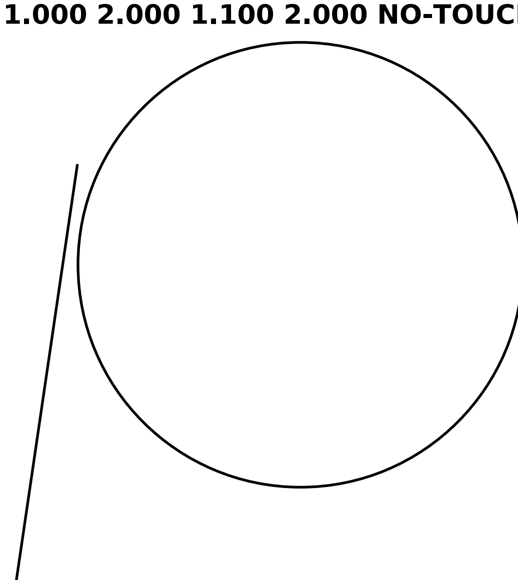
**1.000 2.000 0.900 2.000 TOUCH**



**1.000 2.000 1.020 2.000 TOUCH**



**1.000 2.000 1.100 2.000 NO-TOUCH**



```
Sequence Alignment
-------- ---------

Sequence alignment algorithms are at the heart of DNA
sequencing.  Here we give a simplified sequence
alignment problem.

You are given two sequences, each a string of capital
letters.  To align them, you insert dashes (-) into the
strings to make them the same length, and then put them
next to each other and compute a score for the align-
ment.  You seek an alignment with the best score.

The dashes can be put before or after a string, or
between letters.  Several dashes may be put between
two letters.

After inserting dashes the positions in each string are
numbered 1, 2, 3, ..., and the two modified strings have
identical lengths.  Then the score is computed from
three parameters, A, B, and C, as follows

  + A   For every position in which the two strings
        have matching letters.

  - B   For every position in which one string has
        '-' and the other a letter and the '-' is
        not before all letters in its string or after
        all letters in its string.

  - C   For every position in which the two strings
        have different letters.

Alignments that have a position where both aligned
strings have '-' are NOT permitted.

For example, given the strings

    ABCXDEA
    ACYDEDE

and the alignment

    ABCXDEA--
    A-CYDE-DE

the score of the alignment is +A -B +A -C +A +A -B or
4A-2B-C.


Input
-----

For each of several test cases, a line containing just
the test case name, followed by a line containing

    N A B C

followed by N pairs of lines, each pair containing two
strings.  Strings are at most 1,000 characters long and
contain only upper case letters.  A, B, and C are float-
ing point numbers.

  0 <= A,B,C <= 1
  sum over all string pairs in an input file of (I+J)^2
      <= 100,000,000
  where I and J are the lengths of the two strings

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

Output
------

For each test case, first an exact copy of the test case name line, then for each string pair two lines containing the alignment that has the best score. The input will be such that this alignment is always unique.

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

```
** SAMPLE 1 **
1 1.0 0.7 0.9
ABCXDEA
ACYDEDE
** SAMPLE 2 **
1 1.0 0.7 0.9
ABCDEFGHIJK
ABCXEFHIJKIJK
** SAMPLE 3 **
2 0.5 1.0 0.5
CGATGTATCGAATGTATACG
CGATGTATGGATTGATACG
CGTGAGAGTACGCTATGCTCGA
CTTGGAGTACCTATGTCGA
```

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

```
** SAMPLE 1 **
ABCXDEA--
A-CYDE-DE
** SAMPLE 2 **
ABCDEFGHIJK---
ABCXEF-HIJKIJK
** SAMPLE 3 **
CGATGTATCGAATGTATACG
CGATGTATGGATTG-ATACG
CGTGAGAGTACGCTATGCTCGA
CTTG-GAGTAC-CTATG-TCGA
```

File:      alignment.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sat Oct  6 03:01:08 EDT 2018

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

```
Maximum Parallel Machine
------- -------- -------

The maximum parallel machine (MAXPAR) has an extremely
wide instruction that can perform in parallel one opera-
tion for each register.  More specifically, in a single
instruction, each register can be set to the output of
an arithmetic operation applied to values read from two
register operands.

You have been asked to write a compiler for this
machine, which given a program written in a typical
programming language plus sufficient input data to
determine the entire control flow of the program when
it is run, will compile a MAXPAR program to perform
the same computation.

As an initial exercise, you are going to solve this
problem in a simplified case.  In this simplified case
all data are 32 bit signed integers, the only memory are
26 registers named A, B, C, ..., Z, and instructions can
only reference registers.  The registers are given
initial values before the program starts and register
values when the program ends are the program output.


Input Programming Language
----- ----------- --------

The input programming language has the syntax:

    program ::= statements
    statements ::= statement ; | statements statement ;
    statement ::= arithmetic-statement
                | loop
                | exit-statement
    arithmetic-statement ::=
        register = register op register
    register ::= A | B | C | D | E | F | G | H | I
               | J | K | L | M | N | O | P | Q | R
               | S | T | U | V | W | X | Y | Z
    op ::= + | - | *
    loop ::= begin; statements end
    exit-statement ::= exit
                     | exit if register cop register
    cop ::= <= | == | !=

If execution arrives at the end of a loop, execution
continues at the beginning of the loop.  An exit-state-
ment can only appear in a loop and exits the smallest
loop containing the exit-statement.

Whitespace, including line breaks, is optional ANYWHERE,
and can be completely deleted before syntax analysis.

Registers that appear in exit-statements are called
‘control’ registers, and registers input to arithmetic
statements that compute control registers are also
control registers.  All other registers are called
‘data’ registers.  In an arithmetic-statement, ALL
registers must be control or ALL must be data.  You are
being asked to compile the input program into a MAXPAR
program, given the initial values of the control
registers, without knowing the values of the data
registers.
```

The MAXPAR program contains nothing but data register arithmetic statements, and is designed to be run by giving all data register initial values, running the program, and then taking output from the register values at the end of the program. However you are just compiling the MAXPAR program, and not running it.

MAXPAR Programming Language
------ ----------- --------

The MAXPAR programming language has the syntax:

```
    program ::= statements
    statements ::= statement ; | statements statement ;
    statement ::= substatements
    substatements ::= substatement
                    | substatements , substatement
    substatement ::= arithmetic-statement
    arithmetic-statement ::=
        see Input Programming Language above
```

Whitespace, including line breaks, is optional ANYWHERE, and can be completely deleted before syntax analysis.

The substatements of a single statement are executed in parallel. That is, all the operands of all the substatements are read first, then all the operations are performed, then all the operation values are written last.

All the registers appearing in the MAXPAR program are data registers.

Input
-----

For each test case, first a line that gives the test case name. Then lines containing one input program followed by a line containing just '$'. Then one or more sets of control register values. Each set consists of zero or more lines of the form

```
        register value
```

where value is an integer. Each set ends with a line containing just '$'. The test case ends with an additional line containing just '#'.

Each set of control register values will have exactly one value for EACH control register in the input program. Some input programs may have no control registers.

Input ends with an end of file. No input line is longer than 80 characters, and the input program in its entirety contains no more than 10,000 non-whitespace characters.

The input is guaranteed to be syntactically correct and execute no more than 100,000 statements when run on any set of initial control register values. Registers store 32-bit signed integers and integer arithmetic is modulo 2^32 (as in C, C++, and JAVA). Control register input values are guaranteed to have no more than 9 digits.

Output
------

For each test case, first an exact copy of the test case name line. Then for each control register set a MAXPAR program that computes the same thing as the input program when it is run with the initial control register values given by the register set. Each MAXPAR program is followed by a line containing just '$'. Lastly, at the end of the test case, output one more line containing just '#'.

In the execution of either the input program or the MAXPAR program the different values of a register can be tagged with the order of their computation, so that the 0'th value is the initial value, the 1'st value is the first value computed, etc. Given this we require that substatements of the MAXPAR program correspond 1-1 to data register arithmetic-statement executions in the input program such that:

(1) The arithmetic-statement of an execution is syntactically identical to its corresponding MAXPAR substatement (with whitespace removed).

(2) If a substatement computes the n'th value of its destination register, its corresponding input arithmetic-statement execution computes the n'th value of its destination register.

(3) If a substatement inputs the m'th value of its first (or second) operand register, its corresponding input arithmetic-statement execution inputs the m'th value of its first (or second) operand register.

Among all possible MAXPAR programs, you must output one with the fewest possible number of statements. If there are several answers, any one will do.

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

```
-- SAMPLE 1 --
Z = Z + Y;
Z = Z + Y;
X = Z + Y;
Y = Y + W;
$
$
#
-- SAMPLE 2 --
X = Y + Z;
Z = Y + X;
begin;
exit if C == D;
C = C + B;
Z = Z + X;
Y = Y + X;
end;
$
B 1
C 1
D 4
$
#
```

```
-- SAMPLE 3 --
begin;
exit if A == B;
R = X * Y;
exit if A <= B;
S = X - Y;
exit if A != B;
T = X + Y;
exit;
end;
$
A 0
B 0
$
A 0
B 1
$
A 1
B 0
$
#


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

-- SAMPLE 1 --
Z=Z+Y;
Z=Z+Y;
X=Z+Y,Y=Y+W;
$
#
-- SAMPLE 2 --
X=Y+Z;
Y=Y+X,Z=Y+X;
Y=Y+X,Z=Z+X;
Y=Y+X,Z=Z+X;
Z=Z+X;
$
#
-- SAMPLE 3 --
$
R=X*Y;
$
R=X*Y,S=X-Y;
$
#
```

Note
----

Similar techniques have been used to translate traces of real program executions into MAXPAR-like code in order to determine the maximum parallelism possible in these executions. One such is the ORACLE code in the ‘Limits of control flow parallelism’, ISCA ’92, Monica S. Lam and Robert P. Wilson, ACM Digital Library. The ORACLE machine is like the MAXPAR machine except that ORACLE has an unbounded number of registers and never writes a register more than once. Typical results are that gcc and latex executions translate into ORACLE code with an average of 100 to 200 substatements per statement, thus giving an upper bound of 200 on the factor that parallelism can speed up gcc and latex (without reorganizing the code completely).

File:      maxparallel.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sat Oct  6 11:55:17 EDT 2018

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

```
Flow Security
---- --------

You are responsible for getting water from Lake Achu to
the city of Bzurt.  Unfortunately rebels are threatening
to blow up one of the pipes through which the water
flows.   You must estimate the worst amount of damage
they can do.


Input
-----

For each test case, first a line that gives the test
case name.  Then a line of the form

        N P

giving the number N of nodes in the water network and
the number of pipes P connecting them.  Nodes are
numbered 1, 2, ..., N.  After this line there are P
pipe description lines of the form

        n1 n2 f

that state that there is a pipe from node n1 to node n2
with a capacity of f cubic meters per second in either
direction.

Node 1 is Lake Achu and node N is Bzurt.  All numbers
are integers.

    2 <= N <= 10,000
    1 <= P <= 20,000
    1 <= n1, n2 <= N
    n1 != n2
    1 <= f <= 100,000

    sum over all test cases of P * sum f over all pipes
    <= 100,000,000

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

Because of the age of the system of pipes, some pipes
are not functional, and these are not listed in the
input.  Because of this, the pipe system may not be
connected, and some nodes may have no pipes into them,
but Lake Achu is always connected to Bzurt.


Output
------

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

        F0 F1

where F0 is the maximum flow possible from node 1 to
node N if all pipes are left intact, and F1 is the
smallest maximum flow that will be possible after the
rebels destroy some single pipe.  In other words,
F0 - F1 is the maximum amount the rebels can reduce
the maximum flow of the system by destroying just one
pipe.

All maximum flows are in cubic meters per second.
```

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

-- SAMPLE 1 --
5 7
1 2 10
2 3 10
3 4 10
4 5 10
1 3 5
2 4 5
3 5 5
-- SAMPLE 2 --
11 15
2 3 8
3 4 8
5 6 6
6 7 6
8 9 4
9 10 4
1 2 3
1 5 2
1 8 4
4 11 4
7 11 3
10 11 2
2 6 5
9 7 5
7 4 3


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

-- SAMPLE 1 --
15 5
-- SAMPLE 2 --
9 5
```

File:      flowsecurity.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sat Oct  6 11:57:26 EDT 2018

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

## Driving

Its 2200 and the Earth is very much too hot so the good old USA has moved to D-dimensional space, or D-space, somewhere in the Ort Cloud. However, things are not that different from today. There are still towns and cities and roads (running through road-tubes) in between. Actually, its a little more like 1922, before Route 66 was established (in 1926), in that there are no super-highways. You just drive from town to town. Distances are still measured in miles, and as automatic driving did not survive the (lidar) Chaff Anarchists, your car must have a driver.

You are driving with friends from New San Francisco to New New York City. You agree that you will only change drivers at a town and you define a ‘shift’ to be the part of the route that one driver drives at a time. You further agree that

(1) each shift will drive a shortest route from its initial town to its final town

(2) each shift but the last will drive at least M0 miles

(3) each shift will drive at most M1 miles

(4) the number of shifts S will be as small as possible, given the above

(5) S will not be greater than S0

(6) M1 will be as small as possible given the above

You want to figure out good values for M0, S0, M1, S, and also plan your route. You have decided to write a program that given M0 and S0 will compute M1 and S and a route.

## Input

For each test case, first a line that gives the test case name. Then a line of the form

```
        T R Q D
```

where T is the number of towns, R the number of roads, Q the number of queries, and D the dimension of D-space. Towns are numbered 1, 2, ..., T, where 1 is New San Francisco and T is New New York City.

Then R road description lines follow, each of the form:

```
        I J M
```

describing a road from Town I to Town J that is M miles long and does not go through any other towns. There is at most one road between any two towns, and no road connects a town to itself.

Lastly Q query lines follow, each of the form

```
        M0 S0
```

where M0 and S0 are as describe above.

The data is such that there is a path between any two towns, i.e., the graph of towns and roads is connected. The road lengths are close to straight line distances, but in D-space of course.

All input numbers are integers.

```
    2 <= T <= 5,000
    1 <= R <= 20,000
    1 <= Q <= 100
    2 <= D <= 10
    1 <= M0 <= M1 <= 1,000
    1 <= S0 <= 100
    1 <= I,J <= T
    I != J
    1 <= M <= 200

    Sum R*T+Q*T^2 over all
    test cases in one file     <= 40,000,000
```

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

Output
------

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

```
        M0 S0 M1 S T1 T2 ... TS
```

where M0 and S0 are copied from the query description line, M1 is the smallest possible value of M1 given M0 and S0, S is the smallest possible number of shifts given M0 and M1, and T1, T2, ..., TS are the numbers of the DESTINATION towns of the consecutive shifts in the route. Necessarily S <= S0 and TS == T.

If there is more than one route for a given M1 and S, any one will do.

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

```
-- SAMPLE 1 --
5 5 12 2
1 2 10
2 3 10
3 4 10
4 5 10
1 3 15
5 10
5 20
10 1
10 2
10 3
10 4
20 1
20 2
20 3
30 1
30 2
30 3
```

```
-- SAMPLE 2 --
12 15 11 2
 1 12 100
 1  2  90
 2 12  90
 1  3  80
 3  4  80
 4 12  80
 1  5  70
 5  6  70
 6  7  70
 7 12  70
 1  8  60
 8  9  60
 9 10  60
10 11  60
11 12  60
 50 1
 50 2
 50 3
 50 4
 50 5
 50 6
 60 6
 70 6
 80 6
 90 6
100 6
```

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

-- SAMPLE 1 --
5 10 10 4 2 3 4 5
5 20 10 4 2 3 4 5
10 1 35 1 5
10 2 20 2 3 5
10 3 15 3 3 4 5
10 4 10 4 2 3 4 5
20 1 35 1 5
20 2 25 2 4 5
20 3 25 2 4 5
30 1 35 1 5
30 2 35 1 5
30 3 35 1 5
-- SAMPLE 2 --
50 1 100 1 12
50 2 90 2 2 12
50 3 80 3 3 4 12
50 4 70 4 5 6 7 12
50 5 60 5 8 9 10 11 12
50 6 60 5 8 9 10 11 12
60 6 60 5 8 9 10 11 12
70 6 70 4 5 6 7 12
80 6 80 3 3 4 12
90 6 90 2 2 12
100 6 100 1 12
```

File:      driving.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Wed Aug  8 04:14:51 EDT 2018

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

Overlap
-------

Your boss at Applied Counting Mechanisms has signed a contract to count the number of pairs of rectangles that overlap each other within very large sets of such rectangles. You have been given the job of writing the program to do this, with an emphasis on optimization due to the large sizes of the sets.

Input
-----

A sequence of test cases. Each test case begins with a line containing the test case name. This is followed by rectangle description lines each of the form

```
        Xmin Xmax Ymin Ymax
```

denoting the rectangle [Xmin,Xmax] x [Ymin,Ymax].

The last line of the test case contains just ‘*’.

All numbers are integers. If N is the number of rectangles in a test case,

```
        2 <= N <= 1,000,000
        -10^15 <= Xmin < Xmax <= +10^15
        -10^15 <= Ymin < Ymax <= +10^15
```

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 number of pairs of rectangles which overlap.

Two rectangles overlap if and only if they have an interior point in common. For example, [0,2]x[4,5] and [1,3]x[2,8] overlap as they have the intersection [1,2]x[4,5] which has an interior, but [0,2]x[4,5] and [2,4]x[2,8] do not overlap as they have the intersection {2}x[4,5] which has no interior.

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

-- SAMPLE 1 --
1 3 0 4
2 4 1 5
3 5 2 6
*
-- SAMPLE 2 --
0 10 -10 -0
1 11 -11 -1
2 12 -12 -2
3 13 -13 -3
4 14 -14 -4
5 15 -15 -5
6 16 -16 -6
7 17 -17 -7
8 18 -18 -8
9 19 -19 -9
*
-- SAMPLE 3 --
-10 +10 -5 +5
-10 +10 -5 +5
-10 +10 -5 +5
-10 +10 -5 +5
-10 +10 -6 +6
*
```

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

-- SAMPLE 1 --
2
-- SAMPLE 2 --
45
-- SAMPLE 3 --
10
```

File:      overlap.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Fri Oct  5 05:41:13 EDT 2018

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

## Walls

The Grand Mazzz of Zurrr has the prettiest walls. Some are made of colored glass, some of rubies, some of silver, and even one of gold.

Every morning Fritzzz must run through the Mazzz to wait on the King of Zurrr. This is annoying and time consuming, not to mention that if Fritzzz gets lost he might never get out!

Fortunately Fritzzz has a map of the Mazzz, although it is so big its hard to use it to plan a strategy. But he has encoded the map as computer data and hopes you can help him find the shortest path through the Mazzz.

## Input

For each test case, first a line that gives the test case name. Then a line of the form

```
        X Y N
```

giving the X and Y dimensions of the Mazzz and the number of walls N. Then there are N lines each containing

```
        X1 Y1 X2 Y2
```

defining a wall whose ends are at (X1,Y1) and (X2,Y2). All walls are straight. Two or more walls may have a common endpoint, but otherwise walls do not intersect or overlap. Two walls that have a common endpoint may be parallel. All walls are of non-zero length. Four of the walls bound the Mazzz, and are called boundary walls. No other walls have endpoints on the Mazzz boundary. All input numbers are integers.

```
    4 <= N <= 500
    100 <= X,Y <= 10,000
    0 <= X1, X2 <= X
    0 <= Y1, Y2 <= Y
```

Fritzzz starts at (0,0) and has to get to (X,Y).

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

## Output

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

```
        L
```

which is the shortest distance that Fritzzz must travel to get from (0,0) to (X,Y) while staying inside the Mazzz. L must be accurate to 2 decimal places.

Fritzzz can go arbitrarily close to walls. If walls join at a common endpoint, or corner, Fritzzz can go around the corner but not through it. Since only boundary walls touch the boundary, Fritzzz could run along the boundary, so necessarily L < X+Y.

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

-- SAMPLE 1 --
100 100 24
0 0 0 100
0 100 100 100
100 100 100 0
100 0 0 0
30 30 40 40
25 25 30  5
50 50 60 60
20 40 30 30
60 60 70 70
70 70 80 80
50 30 60 60
89 10 60 60
 5 20 10 15
10 15 15 20
20 10 10  5
85 85 90 90
75 95 90 90
90 80 90 90
10 10 20 20
10 10 20 10
20 20 15 20
20 40 20 80
 5 20 10 80
30  5 50 10


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

-- SAMPLE 1 --
148.31
```

```
Display
-------

The command

    display_walls -X walls.in

can be used to display test case input.  See

    display_walls -doc

for more documentation.


File:      walls.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sun Sep 23 12:09:24 EDT 2018
```

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

## -- SAMPLE 1 --