Problem Set for the 2nd Day of Summer Camp 2007

Japanese Alumni Group

Contest Held on 23 Sep 2007

Status of Problems

All problems were newly created by the members of Japanese Alumni Group.

Some portion of the problem statement was modified for clarifications.

The second figure and the sample output in Problem E were found to be incorrect. It is corrected in this PDF file.

This problem set was typeset by LaTeX 2_ε with a style file made by a member of JAG from scratch, so that the statement looks like those used in ACM-ICPC Regional Contest held in Japan.

Terms of Use

Problem A
Hit and Blow
Input: A.txt

Hit and blow is a popular code-breaking game played by two people, one codemaker and one codebreaker. The objective of this game is that the codebreaker guesses correctly a secret number the codemaker makes in his or her mind.

The game is played as follows. The codemaker first chooses a secret number that consists of four different digits, which may contain a leading zero. Next, the codebreaker makes the first attempt to guess the secret number. The guessed number needs to be legal (i.e. consist of four different digits). The codemaker then tells the numbers of *hits* and *blows* to the codebreaker. Hits are the matching digits on their right positions, and blows are those on different positions. For example, if the secret number is 4321 and the guessed is 2401, there is one hit and two blows where 1 is a hit and 2 and 4 are blows. After being told, the codebreaker makes the second attempt, then the codemaker tells the numbers of hits and blows, then the codebreaker makes the third attempt, and so forth. The game ends when the codebreaker gives the correct number.

Your task in this problem is to write a program that determines, given the situation, whether the codebreaker can logically guess the secret number within the next two attempts. Your program will be given the four-digit numbers the codebreaker has guessed, and the responses the codemaker has made to those numbers, and then should make output according to the following rules:

- if only one secret number is possible, print the secret number;
- if more than one secret number is possible, but there are one or more critical numbers, print the *smallest* one;
- otherwise, print "`????`" (four question symbols).

Here, critical numbers mean those such that, after being given the number of hits and blows for them on the next attempt, the codebreaker can determine the secret number uniquely.

Input

The input consists of multiple data sets. Each data set is given in the following format:

> N
> *four-digit-number*$_1$ *n-hits*$_1$ *n-blows*$_1$

...
four-digit-number$_N$ *n-hits*$_N$ *n-blows*$_N$

N is the number of attempts that has been made. *four-digit-number*$_i$ is the four-digit number guessed on the i-th attempt, and *n-hits*$_i$ and *n-blows*$_i$ are the numbers of hits and blows for that number, respectively. It is guaranteed that there is at least one possible secret number in each data set.

The end of input is indicated by a line that contains a zero. This line should not be processed.

Output

For each data set, print a four-digit number or "`????`" on a line, according to the rules stated above.

Sample Input

```
2
1234 3 0
1245 3 0
1
1234 0 0
2
0123 0 4
1230 0 4
0
```

Output for the Sample Input

```
1235
????
0132
```

Problem B
Turn Left
Input: B.txt

Taro got a driver's license with a great effort in his campus days, but unfortunately there had been no opportunities for him to drive. He ended up obtaining a gold license.

One day, he and his friends made a plan to go on a trip to Kyoto with you. At the end of their meeting, they agreed to go around by car, but there was a big problem; none of his friends was able to drive a car. Thus he had no choice but to become the driver.

The day of our departure has come. He is going to drive but would never turn to the right for fear of crossing an opposite lane (note that cars keep left in Japan). Furthermore, he cannot U-turn for the lack of his technique. The car is equipped with a car navigation system, but the system cannot search for a route without right turns. So he asked to you: "I hate right turns, so, could you write a program to find the shortest left-turn-only route to the destination, using the road map taken from this navigation system?"

Input

The input consists of multiple data sets. The first line of the input contains the number of data sets. Each data set is described in the format below:

$m\ n$
$name_1\ x_1\ y_1$
...
$name_m\ x_m\ y_m$
$p_1\ q_1$
...
$p_n\ q_n$
$src\ dst$

m is the number of intersections. n is the number of roads. $name_i$ is the name of the i-th intersection. (x_i, y_i) are the integer coordinates of the i-th intersection, where the positive x goes to the east, and the positive y goes to the north. p_j and q_j are the intersection names that represent the endpoints of the j-th road. All roads are bidirectional and either vertical or horizontal. *src* and *dst* are the names of the source and destination intersections, respectively.

You may assume all of the followings:

- $2 \leq m \leq 1000$, $0 \leq x_i \leq 10000$, and $0 \leq y_i \leq 10000$;
- each intersection name is a sequence of one or more alphabetical characters at most 25 character long;
- no intersections share the same coordinates;
- no pair of roads have common points other than their endpoints;
- no road has intersections in the middle;
- no pair of intersections has more than one road;
- Taro can start the car in any direction; and
- the source and destination intersections are different.

Note that there may be a case that an intersection is connected to less than three roads in the input data; the rode map may not include smaller roads which are not appropriate for the non-local people. In such a case, you still have to consider them as intersections when you go them through.

Output

For each data set, print how many times *at least* Taro needs to pass intersections when he drive the route of the shortest distance without right turns. The source and destination intersections must be considered as "passed" (thus should be counted) when Taro starts from the source or arrives at the destination. Also note that there may be more than one shortest route possible.

Print "`impossible`" if there is no route to the destination without right turns.

Sample Input

```
2 1
KarasumaKitaoji 0 6150
KarasumaNanajo 0 0
KarasumaNanajo KarasumaKitaoji
KarasumaKitaoji KarasumaNanajo
3 2
KujoOmiya 0 0
KujoAburanokoji 400 0
OmiyaNanajo 0 1150
KujoOmiya KujoAburanokoji
KujoOmiya OmiyaNanajo
KujoAburanokoji OmiyaNanajo
10 12
KarasumaGojo 745 0
HorikawaShijo 0 870
ShijoKarasuma 745 870
ShijoKawaramachi 1645 870
```

```
HorikawaOike 0 1700
KarasumaOike 745 1700
KawaramachiOike 1645 1700
KawabataOike 1945 1700
KarasumaMarutamachi 745 2445
KawaramachiMarutamachi 1645 2445
KarasumaGojo ShijoKarasuma
HorikawaShijo ShijoKarasuma
ShijoKarasuma ShijoKawaramachi
HorikawaShijo HorikawaOike
ShijoKarasuma KarasumaOike
ShijoKawaramachi KawaramachiOike
HorikawaOike KarasumaOike
KarasumaOike KawaramachiOike
KawaramachiOike KawabataOike
KarasumaOike KarasumaMarutamachi
KawaramachiOike KawaramachiMarutamachi
KarasumaMarutamachi KawaramachiMarutamachi
KarasumaGojo KawabataOike
8 9
NishikojiNanajo 0 0
NishiojiNanajo 750 0
NishikojiGojo 0 800
NishiojiGojo 750 800
HorikawaGojo 2550 800
NishiojiShijo 750 1700
Enmachi 750 3250
HorikawaMarutamachi 2550 3250
NishikojiNanajo NishiojiNanajo
NishikojiNanajo NishikojiGojo
NishiojiNanajo NishiojiGojo
NishikojiGojo NishiojiGojo
NishiojiGojo HorikawaGojo
NishiojiGojo NishiojiShijo
HorikawaGojo HorikawaMarutamachi
NishiojiShijo Enmachi
Enmachi HorikawaMarutamachi
HorikawaGojo NishiojiShijo
0 0
```

Output for the Sample Input

```
2
impossible
13
```

4

Problem C
!
Input: C.txt

You are one of ICPC participants and in charge of developing a library for multiprecision numbers and radix conversion. You have just finished writing the code, so next you have to test if it works correctly. You decided to write a simple, well-known factorial function for this purpose:

$$M! = \prod_{i=1}^{M} i = M \times (M-1) \times \cdots \times 2 \times 1, \;\; 0! = 1.$$

Your task is to write a program that shows the number of trailing zeros when you compute $M!$ in base N, given N and M.

Input

The input contains multiple data sets. Each data set is described by one line in the format below:

N M

where N is a decimal number between 8 and 36 inclusive, and M is given in the string representation in base N. Exactly one white space character appears between them.

The string representation of M contains up to 12 characters in base N. In case N is greater than 10, capital letters A, B, C, ... may appear in the string representation, and they represent 10, 11, 12, ..., respectively.

The input is terminated by a line containing two zeros. You should not process this line.

Output

For each data set, output a line containing a decimal integer, which is equal to the number of trailing zeros in the string representation of $M!$ in base N.

Sample Input

```
10 500
16 A
0 0
```

Output for the Sample Input

```
124
2
```

Problem D
Speed
Input: D.txt

Do you know *Speed*? It is one of popular card games, in which two players compete how quick they can move their cards to tables.

To play *Speed*, two players sit face-to-face first. Each player has a deck and a tableau assigned for him, and between them are two tables to make a pile on, one in left and one in right. A tableau can afford up to only four cards.

There are some simple rules to carry on the game:

1. A player is allowed to move a card from his own tableau onto a pile, only when the rank of the moved card is a neighbor of that of the card on top of the pile. For example A and 2, 4 and 3 are neighbors. A and K are also neighbors in this game.
2. He is also allowed to draw a card from the deck and put it on a vacant area of the tableau.
3. If both players attempt to move cards on the same table at the same time, only the faster player can put a card on the table. The other player cannot move his card to the pile (as it is no longer a neighbor of the top card), and has to bring it back to his tableau.

First each player draws four cards from his deck, and places them face on top on the tableau. In case he does not have enough cards to fill out the tableau, simply draw as many as possible. The game starts by drawing one more card from the deck and placing it on the tables on their right simultaneously. If the deck is already empty, he can move an arbitrary card on his tableau to the table.

Then they continue performing their actions according to the rule described above until both of them come to a deadend, that is, have no way to move cards. Every time a deadend occurs, they start over from each drawing a card (or taking a card from his or her tableau) and placing on his or her right table, regardless of its face. The player who has run out of his card first is the winner of the game.

Mr. James A. Games is so addicted in this simple game, and decided to develop robots that plays it. He has just completed designing the robots and their program, but is not sure if they work well, as the design is so complicated. So he asked you, a friend of his, to write a program that simulates the robots.

The algorithm for the robots is designed as follows:

- A robot draws cards in the order they are dealt.
  - Each robot is always given one or more cards.
  - In the real game of *Speed*, the players first classify cards by their colors to enable them to easily figure out which player put the card. But this step is skipped in this simulation.
  - The game uses only one card set split into two. In other words, there appears at most one card with the same face in the two decks given to the robots.
- As a preparation, each robot draws four cards, and puts them to the tableau from right to left.
  - If there are not enough cards in its deck, draw all cards in the deck.
- After this step has been completed on both robots, they synchronize to each other and start the game by putting the first cards onto the tables in the same moment.
  - If there remains one or more cards in the deck, a robot draws the top one and puts it onto the right table. Otherwise, the robot takes the rightmost card from its tableau.
- Then two robots continue moving according to the basic rule of the game described above, until neither of them can move cards anymore.
  - When a robot took a card from its tableau, it draws a card (if possible) from the deck to fill the vacant position after the card taken is put onto a table.
  - It takes some amount of time to move cards. When a robot completes putting a card onto a table while another robot is moving to put a card onto the same table, the robot in motion has to give up the action immediately and returns the card to its original position.
  - A robot can start moving to put a card on a pile at the same time when the neighbor is placed on the top of the pile.
  - If two robots try to put cards onto the same table at the same moment, only the robot moving a card to the left can successfully put the card, due to the position settings.
  - When a robot has multiple candidates in its tableau, it prefers the cards which can be placed on the right table to those which cannot. In case there still remain multiple choices, the robot prefers the weaker card.
- When it comes to a deadend situation, the robots start over from each putting a card to the table, then continue moving again according to the algorithm above.
- When one of the robots has run out the cards, i.e., moved all dealt cards, the game ends.
  - The robot which has run out the cards wins the game.
  - When both robots run out the cards at the same moment, the robot which moved the stronger card in the last move wins.

The strength among the cards is determined by their ranks, then by the suits. The ranks are strong in the following order: A > K > Q > J > X (10) > 9 > $\cdots$ > 3 > 2. The suits are strong in the following order: S (Spades) > H (Hearts) > D (Diamonds) > C (Cloves). In other words, SA is the strongest and C2 is the weakest.

The robots require the following amount of time to complete each action:

- 300 milliseconds to draw a card to the tableau,
- 500 milliseconds to move a card to the right table,
- 700 milliseconds to move a card to the left table, and
- 500 milliseconds to return a card to its original position.

Cancelling an action always takes the constant time of 500ms, regardless of the progress of the action being cancelled. This time is counted from the exact moment when the action is interrupted, not the beginning time of the action.

You may assume that these robots are well-synchronized, i.e., there is no clock skew between them.

For example, suppose Robot A is given the deck "`S3 S5 S8 S9 S2`" and Robot B is given the deck "`H7 H3 H4`", then the playing will be like the description below. Note that, in the description, "the table A" (resp. "the table B") denotes the right table for Robot A (resp. Robot B).

- Robot A draws four cards S3, S5, S8, and S9 to its tableau from right to left. Robot B draws all the three cards H7, H3, and H4.
- Then the two robots synchronize for the game start. Let this moment be 0ms.
- At the same moment, Robot A starts moving S2 to the table A from the deck, and Robot B starts moving H7 to the table B from the tableau.
- At 500ms, the both robots complete their moving cards. Then Robot A starts moving S3 to the table A 1(which requires 500ms), and Robot B starts moving H3 also to the table A (which requires 700ms).
- At 1000ms, Robot A completes putting S3 on the table A. Robot B is interrupted its move and starts returning H3 to the tableau (which requires 500ms). At the same time Robot A starts moving S8 to the table B (which requires 700ms).
- At 1500ms, Robot B completes returning H3 and starts moving H4 to the table A (which requires 700ms).
- At 1700ms, Robot A completes putting S8 and starts moving S9 to the table B.
- At 2200ms, Robot B completes putting H4 and starts moving H3 to the table A.
- At 2400ms, Robot A completes putting S9 and starts moving S5 to the table A.
- At 2900ms, The both robots are to complete putting the cards on the table A. Since Robot B is moving the card to the table left to it, Robot B completes putting H3. Robot A is interrupted.
- Now Robot B has finished moving all the dealt cards, so Robot B wins this game.

Input

The input consists of multiple data sets, each of which is described by four lines. The first line of each data set contains an integer N_A, which specifies the number of cards to be dealt as a deck to Robot A. The next line contains a card sequences of length N_A. Then the number N_B and card sequences of length N_B for Robot B follows, specified in the same manner.

In a card sequence, card specifications are separated with one space character between them. Each card specification is a string of 2 characters. The first character is one of '`S`' (spades), '`H`' (hearts), '`D`' (diamonds) or '`C`' (cloves) and specifies the suit. The second is one of '`A`', '`K`', '`Q`', '`J`', '`X`' (for 10) or a digit between '`9`' and '`2`', and specifies the rank. As this game is played with only one card set, there is no more than one card of the same face in each data set.

The end of the input is indicated by a single line containing a zero.

Output

For each data set, output the result of a game in one line. Output "`A wins.`" if Robot A wins, or output "`B wins.`" if Robot B wins. No extra characters are allowed in the output.

Sample Input

```
1
SA
1
C2
2
SA HA
2
C2 C3
5
S3 S5 S8 S9 S2
3
H7 H3 H4
10
H7 CJ C5 CA C6 S2 D8 DA S6 HK
10
C2 D6 D4 H5 DJ CX S8 S9 D3 D5
0
```

Output for the Sample Input

```
A wins.
B wins.
B wins.
A wins.
```

Problem E
Spirograph
Input: E.txt

Some of you might have seen instruments like the figure below.

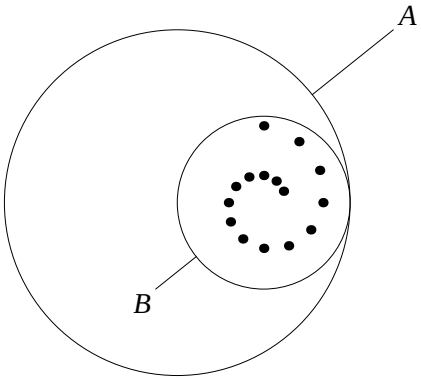

Figure 1: Spirograph

There are a fixed circle (indicated by A in the figure) and a smaller interior circle with some pinholes (indicated by B). By putting a pen point through one of the pinholes and then rolling the circle B without slipping around the inside of the circle A, we can draw curves as illustrated below. Such curves are called *hypotrochoids*.

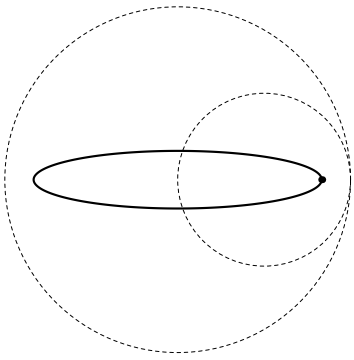

Figure 2: An Example Hypotrochoid

Your task is to write a program that calculates the length of hypotrochoid, given the radius of the fixed circle A, the radius of the interior circle B, and the distance between the B's centroid and the used pinhole.

Input

The input consists of multiple test cases. Each test case is described by a single line in which three integers P, Q and R appear in this order, where P is the radius of the fixed circle A, Q is the radius of the interior circle B, and R is the distance between the centroid of the circle B and the pinhole. You can assume that $0 \leq R < Q < P \leq 1000$. P, Q, and R are separated by a single space, while no other spaces appear in the input.

The end of input is indicated by a line with $P = Q = R = 0$.

Output

For each test case, output the length of the hypotrochoid curve. The error must be within $10^{-2}(= 0.01)$.

Sample Input

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

Output for the Sample Input

```
13.36
6.28
```

Problem F
Mysterious Dungeons
Input: F.txt

The Kingdom of Aqua Canora Mystica is a very affluent and peaceful country, but around the kingdom, there are many evil monsters that kill people. So the king gave an order to you to kill the master monster.

You came to the dungeon where the monster lived. The dungeon consists of a grid of square cells. You explored the dungeon moving up, down, left and right. You finally found, fought against, and killed the monster.

Now, you are going to get out of the dungeon and return home. However, you found strange carpets and big rocks are placed in the dungeon, which were not there until you killed the monster. They are caused by the final magic the monster cast just before its death! Every rock occupies one whole cell, and you cannot go through that cell. Also, each carpet covers exactly one cell. Each rock is labeled by an uppercase letter and each carpet by a lowercase. Some of the rocks and carpets may have the same label.

While going around in the dungeon, you observed the following behaviors. When you enter into the cell covered by a carpet, all the rocks labeled with the corresponding letter (e.g., the rocks with '`A`' for the carpets with '`a`') disappear. After the disappearance, you can enter the cells where the rocks disappeared, but if you enter the same carpet cell again or another carpet cell with the same label, the rocks revive and prevent your entrance again. After the revival, you have to move on the corresponding carpet cell again in order to have those rocks disappear again.

Can you exit from the dungeon? If you can, how quickly can you exit? Your task is to write a program that determines whether you can exit the dungeon, and computes the minimum required time.

Input

The input consists of some data sets.

Each data set begins with a line containing two integers, W and H ($3 \leq W, H \leq 30$). In each of the following H lines, there are W characters, describing the map of the dungeon as $W \times H$ grid of square cells. Each character is one of the following:

- '`@`' denoting your current position,
- '`<`' denoting the exit of the dungeon,

- A lowercase letter denoting a cell covered by a carpet with the label of that letter,
- An uppercase letter denoting a cell occupied by a rock with the label of that letter,
- '`#`' denoting a wall, and
- '`.`' denoting an empty cell.

Every dungeon is surrounded by wall cells ('`#`'), and has exactly one '`@`' cell and one '`<`' cell. There can be up to eight distinct uppercase labels for the rocks and eight distinct lowercase labels for the carpets.

You can move to one of adjacent cells (up, down, left, or right) in one second. You cannot move to wall cells or the rock cells.

A line containing two zeros indicates the end of the input, and should not be processed.

Output

For each data set, output the minimum time required to move from the '`@`' cell to the '`<`' cell, in seconds, in one line. If you cannot exit, output -1.

Sample Input

```
8 3
########
#<A.@.a#
########
8 3
########
#<AaAa@#
########
8 4
########
#<EeEe@#
#FG.e#.#
########
8 8
########
#mmm@ZZ#
#mAAAbZ#
#mABBBZ#
#mABCCd#
#aABCDD#
#ZZcCD<#
########
0 0
```

Output for the Sample Input

```
7
-1
7
27
```

Problem G
Repeated Subsequences
Input: G.txt

You are a treasure hunter traveling around the world. Finally, you've got an ancient text indicating the place where the treasure was hidden. The ancient text looks like a meaningless string of characters at first glance. Actually, the secret place is embedded as the *longest repeated subsequence* of the text.

Well, then, what is the *longest repeated subsequence* of a string? First, you split the given string S into two parts F and R. Then, you take the longest common subsequence L of F and R (longest string L that is a subsequence of both F and R). Since there are many possible ways to split S into two parts, there are many possible L's. The longest repeated subsequence is the longest one among them. For example, the longest repeated subsequence of "ABCABCABAB" is "ABAB", which is obtained when you split "ABCABCABAB" into "ABCABC" and "ABAB".

Now your task is clear. Please find the longest repeated subsequence and get the hidden treasure!

Input

The input consists of multiple data sets. Each data set comes with a single line that contains one string of up to 300 capital letters. It is guaranteed that there is at least one repeated subsequence in each string.

The end of input is indicated by a line that contains "`#END`". This line should not be processed.

Output

For each data set, print the longest repeated subsequence on a line. If there are multiple longest subsequence, print any one of them.

Sample Input

```
ABCABCABAB
ZZZZZZZZZZZZ
#END
```

Output for the Sample Input

```
ABAB
ZZZZZZ
```

Problem H
Petoris
Input: H.txt

You are playing a puzzle game named *petoris*. It is played with a board divided into square grids and square tiles each of which fits to a single grid.

In each step of the game, you have a board partially filled with tiles. You also have a block consisting of several tiles. You are to place this block somewhere on the board or to discard it, under the following restrictions on placement:

- the block can be rotated, but cannot be divided nor flipped;
- no tiles of the block can collide with any tiles existing on the board; and
- all the tiles of the block need to be placed inside the board.

Your task is to write a program to find the maximum score you can earn in this step. Here, the score is the number of the horizontal lines fully filled with tiles after the block is placed, or -1 in case of discard.

Input

The first line of the input is N, the number of data sets. Then N data sets follow.

Each data set consists of lines describing a block and a board. Each description (both for a block and a board) starts with a line containing two integer H and W, the vertical and horizontal dimension. Then H lines follow, each with W characters, where a '`#`' represents a tile and '`.`' a vacancy. You can assume that $0 < H \leq 64$ and $0 < W \leq 64$. Each block consists of one or more tiles all of which are connected. Each board contains zero or more tiles, and has no horizontal line fully filled with tiles at the initial state.

Output

For each data set, print in a single line the maximum possible score.

Sample Input

```
5
4 4
....
....
####
....
12 8
........
........
........
........
........
.......#
##.##..#
.#######
.#######
.#######
.#######
.####.#.
4 4
....
....
.###
...#
12 8
........
........
........
........
........
........
........
##...###
##.#####
#######.
#######.
#######.
4 4
####
#..#
#..#
####
12 8
........
```

```
........
........
........
........
.......#
##.##..#
##....##
##.##.##
##.##.##
##....##
.####.#.
2 2
##
#.
3 3
.##
.##
##.
4 4
....
.##.
.##.
....
2 2
..
..
```

Output for the Sample Input

```
4
1
4
-1
2
```

Problem I
Pythagoraslope
Input: I.txt

Alice, your girlfriend, is a student at an art school. She is in the final year, and now working hard to build a facture for fulfilling the requirement to graduate. Her work is a large pinball with many straight slopes. Before starting to build, she has made several plans, but is unsure if they work as expected. So she asked you, a professional programmer, for help.

You have modeled this situation by a two dimensional plane with some line segments on it. In this model, there is gravitation downward, i.e., in the decreasing direction of y-coordinate. Your task is to write a program that simulates the pinball, and compute the last position where the ball crosses the x-axis.

You may assume coefficient of restitution between the slopes and the ball is 0, i.e., when the ball collides a slope, it instantly loses the velocity component orthogonal to the slope. And since her pinball is so large, you may also assume that the volume of the ball is negligible.

Input

The input consists of multiple data sets. Each data set is given in the format below.

N
g
$x\ y$
$x_{1,1}\ y_{1,1}\ x_{1,2}\ y_{1,2}$
$\ldots$
$x_{N,1}\ y_{N,1}\ x_{N,2}\ y_{N,2}$

where N ($N \leq 100$) is the number of slopes, g is gravity acceleration, and (x, y) is the initial position of the ball. Each of the following N lines represents a slope, which is a line segment between $(x_{i,1}, y_{i,1})$ and $(x_{i,2}, y_{i,2})$.

You may assume that:

- all coordinates are more than or equal to 1, and less than or equal to 10,000;
- $x_{i,1} \neq x_{i,2}$ and $y_{i,1} \neq y_{i,2}$ for all $1 \leq i \leq N$;
- no two line segments cross each other;

- extending or shrinking a slope by the length of 0.0001 does not change the ball's trail, that is, do not change the set of slopes where the ball passes;
- the ball never collides to a slope at the angle of 90 ± 0.0001 degrees from the slope; and
- the initial position of the ball never lies on any slope.

The end of the input is indicated by a line containing a single zero. This is not a part of the data sets, and you must not process it.

Output

For each data set, output the x-coordinate of the final crossing point of the ball's trail and the x-axis. Your program may print any number of digits after the decimal point, but the output must not contain an error greater than $10^{-4}(= 0.0001)$.

Sample Input

```
3
1
120 1000
100 100 180 20
170 10 270 30
270 40 400 20
0
```

Output for the Sample Input

```
403.87458314
```