2009 Mid-Atlantic Regional Programming Contest

Welcome to the 2009 Programming Contest. Before you start the contest, please be aware of the following notes:

The Contest

1. There are eight (8) problems in the packet, using letters A–H. These problems are NOT sorted by difficulty. As a team's solution is judged correct, the team will be awarded a balloon. The balloon colors are as follows:

Problem	Problem Name	Balloon Color
A	Euclid	Yellow
B	Block Game	Green
C	Parlay Wagering	Silver
D	The Ninja Way	Black
E	Extended Manhattan Distance	Orange
F	Off the Wall	Purple
G	Stringer	Pink
H	Word Ladder	Red

2. Solutions for problems submitted for judging are called runs. Each run will be judged.

   The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
Correct	Your submission has been judged correct.
Wrong Answer	Your submission generated output that is not correct or is incomplete.
Output Format Error	Your submission's output is not in the correct format or is misspelled.
Excessive Output	Your submission generated output in addition to or instead of what is required.
Compilation Error	Your submission failed to compile.
Run-Time Error	Your submission experienced a run-time error.
Time-Limit Exceeded	Your submission did not solve the judges' test data within 30 seconds.

3. A team's score is based on the number of problems they solve and penalty points, which reflect the amount of time and number of incorrect submissions made before the problem is solved. For each problem solved correctly, penalty points are charged equal to the time at

which the problem was solved plus 20 minutes for each incorrect submission. No penalty points are added for problems that are never solved. Teams are ranked first by the number of problems solved and then by the fewest penalty points.

4. This problem set contains sample input and output for each problem. However, you may be assured that the judges will test your submission against several other more complex datasets, which will not be revealed until after the contest. Your major challenge is designing other input sets for yourself so that you may fully test your program before submitting your run. Should you receive an incorrect judgment, you should consider what other datasets you could design to further evaluate your program.

5. In the event that you feel a problem statement is ambiguous or incorrect, you may request a clarification. Read the problem carefully before requesting a clarification. If the judges believe that the problem statement is sufficiently clear, you will receive the response, "The problem statement is sufficient; no clarification is necessary." If you receive this response, you should read the problem description more carefully. If you still feel there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found. If the problem statement is ambiguous in specifying the correct output for a particular input, please include that input data in the clarification request.

   You may not submit clarification requests asking for the correct output for inputs that you provide. Sample inputs *may* be useful in explaining the nature of a perceived ambiguity, e.g., "There is no statement about the desired order of outputs. Given the input: …, would not both this: … and this: … be valid outputs?".

   If a clarification is issued during the contest, it will be broadcast to all teams.

6. Runs for each particular problem will be judged in the order they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for B followed by a run for C, but receive the response for C first.

   Do not request clarifications on when a response will be returned. If you have not received a response for a run within 30 minutes of submitting it, **you may have a runner ask the site judge to determine the cause of the delay. Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.**

   If, due to unforeseen circumstances, judging for one or more problems begins to lag more than 30 minutes behind submissions, a clarification announcement will be issued to all teams. This announcement will include a change to the 30 minute time period that teams are expected to wait before consulting the site judge.

7. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification.

Your Programs

8. All solutions must read from standard input and write to standard output. In C this is scanf/printf, in C++ this is cin/cout, and in Java this is System.in/System.out. The judges will ignore all output sent to standard error (cerr in C++ or System.err in Java). You may wish to use standard error to output debugging information. From your workstation you may test your program with an input file by redirecting input from a file:

   ```
   program < file.in
   ```

9. All lines of program input and output should end with a newline character (`\n`, `endl`, or `println()`).

10. All input sets used by the judges will follow the input format specification found in the problem description. You do not need to test for input that violates the input format specified in the problem.

11. Unless otherwise specified, all lines of program output should be left justified, with no leading blank spaces prior to the first non-blank character on that line.

12. Unless otherwise specified, all numbers in your output should begin with the − if negative, followed immediately by 1 or more decimal digits. If it is a real number, then the decimal point should appears, followed by the appropriate number of decimal digits. For output of real numbers, the number of digits after the decimal point will be specified in the problem description as the "precision").

    All real numbers printed to a given precision should be rounded to the nearest value. Rounding should be carried out so that trailing digits of 5 of higher are rounded up, tr4ailing digits of 4 or less are rounded down. For example, if a precision of 2 decimal digits is requested, then 0.0152 would round to 0.02, but 0.0149 would round to 0.01.

    In simpler terms, neither scientific notation nor commas will be used for numbers, and you should ensure you use a printing technique that rounds to the appropriate precision.

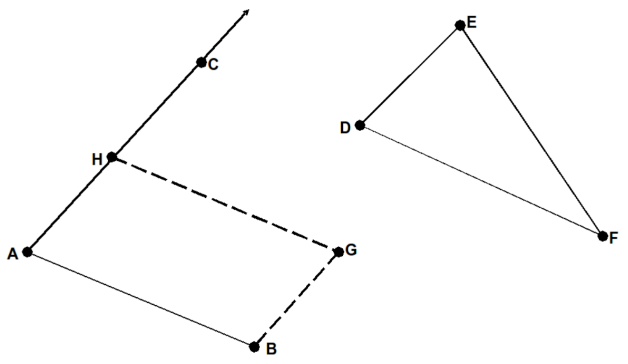
13. If a problem specifies that an input is a floating point number, the input will be presented according to the rules stipulated above for output of real numbers, except that decimal points and the following digits may be omitted for numbers with no non-zero decimal portion. Scientific notation will not be used in input sets unless a problem explicitly allows it.

Good luck, and HAVE FUN!!!

Problem A: Euclid

In one of his notebooks, Euclid gave a complex procedure for solving the following problem. With computers, perhaps there is an easier way.

In a 2D plane, consider a line segment AB, another point C which is not collinear with AB, and a triangle DEF. The goal is to find points G and H such that:



- H is on the ray AC (it may be closer to A than C or further away, but angle CAB is the same as angle HAB)
- $ABGH$ is a parallelogram (AB is parallel to GH, AH is parallel to BG)
- The area of parallelogram ABGH is the same as the area of triangle DEF

Input

Input consists of multiple datasets. Each dataset will consist of twelve real numbers, with no more than 3 decimal places each, on a single line. Those numbers will represent the x and y coordinates of points A through F, as follows:

$$x_A\ y_A\ x_B\ y_B\ x_C\ y_C\ x_D\ y_D\ x_E\ y_E\ x_F\ y_F$$

Points A, B and C are guaranteed to **not** be collinear. Likewise, D, E and F are also guaranteed to be non-collinear. Every number is guaranteed to be in the range from $-1000.0 \ldots 1000.0$ inclusive.

End of the input will be a line with twelve zero values.

Output

For each input set, print a single line with four floating point numbers. These represent points G and H, like this:

$$x_G\ y_G\ x_H\ y_H$$

Print all values to a precision of 3 decimal places. Print a single space between numbers.

Example

Given the input

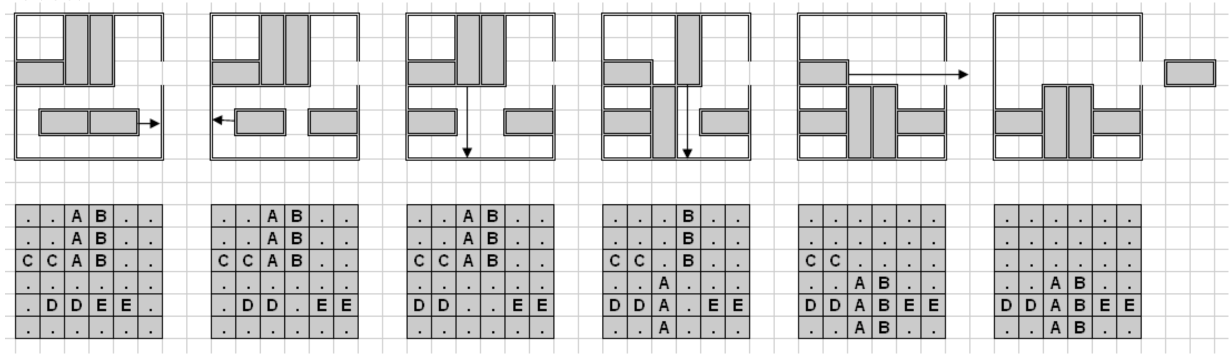
```
0 0 5 0 0 5 3 2 7 2 0 4
1.3 2.6 12.1 4.5 8.1 13.7 2.2 0.1 9.8 6.6 1.9 6.7
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

the output would be

```
5.000 0.800 0.000 0.800
13.756 7.204 2.956 5.304
```

Problem B: Block Game

Bud bought this new board game. He is hooked. He has been playing it over and over again such that he thinks can finish the game with the minimum number of moves, but he is uncertain. He wants you to help him check whether the moves he has listed are indeed the minimum number of moves.



You are given a 6x6 board, and a set of 2x1 or 3x1 (vertical) or 1x2 or 1x3 (horizontal) pieces. You can slide the horizontal pieces horizontally only, and the vertical pieces vertically only. You are only allowed to slide a piece if there is no other piece, nor a wall, obstructing its path.

There will be one special 1x2 horizontal piece. There will also be a gap in the wall, on the right side, on the same row as the special piece, that *only* the special piece can fit through. The goal of the game is to get that one special horizontal piece out of the gap on the right side.

A "move" in this game is when you take a piece and slide it any number of squares (i.e. if you slide a piece horizontally one square, that is one move, and sliding it 2 squares at once is also considered one move).

Input

Input will consist of multiple datasets. Each data set will begin with a line with a single capital letter, indicating the special piece which must move off of the board.

The next 6 lines will consist of 6 characters each. These characters will either be a '.' (period), indicating an empty square, or a capital letter, indicating part of a piece.

The letters are guaranteed to form pieces that are 1x2, 1x3, 2x1 or 3x1, and no letter will be used to represent more than one piece on any given board. The letter indicating the special piece is guaranteed to correspond to a 1x2 piece somewhere on the board.

The end of data is indicated by a single '*' (asterisk) on its own line.

Output

For each data set, print a single integer, indicating the smallest number of moves necessary to remove the given piece, or -1 if it isn't possible. Print each integer on its own line. There should be no blank lines between outputs.

Example

Given the input

```
C
..AB..
..AB..
CCAB..
......
.DDEE.
......
A
......
......
......
......
AA....
......
Z
.ZZ..X
.....X
.....X
.....Y
.....Y
.....Y
*
```

the output would be

```
5
1
-1
```

Problem C: Parlay Wagering

Parlay wagering offers sports bettors the opportunity to win a large sum of money from a small initial wager. A parlay wager is a combination of individual independent wagers that only pays if no individual wager loses. The payout from each wager is applied or "parlayed" to the next wager in turn. If any individual wager loses, the bettor receives nothing. If any individual wager is a tie or "push", that wager is effectively ignored, reducing the ultimate payout.

The sports book quotes the payout rate for an individual wager as a "money line", a non-zero integer in the range -2000 to 2000. To compute the payout for a successful wager, the money line is converted to a decimal multiplier as follows: if the money line is positive, it is divided by 100 to obtain the multiplier. If the money line is negative, the absolute value is divided into 100 to obtain the multiplier. The multiplier is always *truncated* to three digits after the decimal point. The wager is multiplied by this multiplier to determine the amount won. The amount won is *truncated* to the cent (the sports book keeps the fractional cents).

Consider the following example for a five-way parlay wager:

Money Line	Wager	Result	Multiplier	Amount Won
-170	$10.00	Win	100/170	$5.88
-160	$15.88	Win	100/160	$9.92
125	$25.80	Win	125/100	$32.25
-135	$58.05	Win	100/135	$42.95
-140	$101.00	Win	100/140	$72.11
			Subtotal:	$163.11
			Original Wager:	$10.00
			Total Returned:	$173.11

The maximum payout for any parlay wager is $1 million. If the calculated total exceeds that amount, the actual total returned will be $1 million.

Write a program that will calculate the total amount returned for a series of parlay wagers.

For each parlay wager, your program is to print the total amount returned in dollars and cents on a single line starting in the first column without embedded or trailing whitespace. Print the leading dollar sign and insert commas at the millions and thousands positions as needed.

Input

Input will consist of several wagers. The first line of input to your program will contain the total number of parlay wagers as a single positive integer.

Each wager that follows will be represented by a series of lines.

The first line of each parlay wager contains the initial bet and the count of individual wagers as integers separated from each other by a single space.

The following lines represent the individual wagers, one per line. Each individual wager is given as its money line followed by a single space and the result of the wager ("Win", "Tie", or "Loss").

Output

For each parlay wager, your program should print one line containing the total amount returned in dollars and cents. Print the leading dollar sign and insert commas at the millions and thousands positions as needed.

Example

Given the input

```
3
10 5
-170 Win
-160 Win
125 Win
-135 Win
-140 Win
15 8
100 Win
-100 Tie
-250 Win
135 Tie
265 Tie
1500 Win
120 Win
130 Win
10 2
100 Loss
300 Tie
```

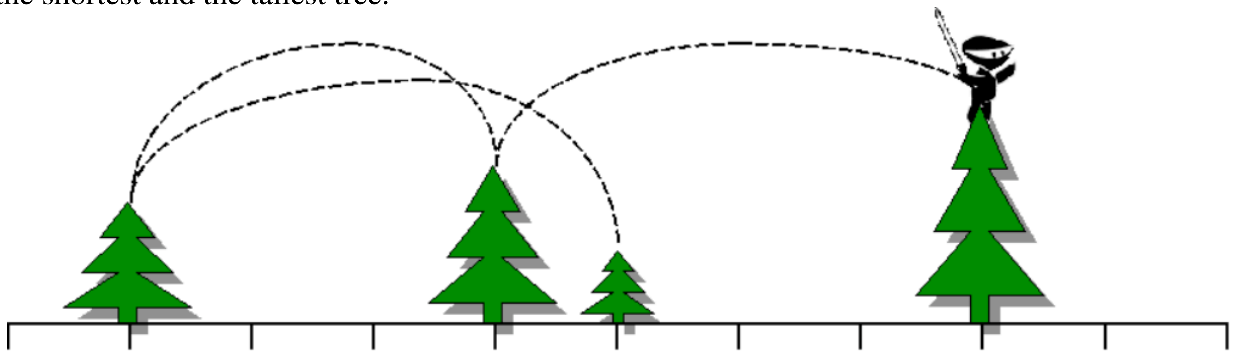
the output would be

```
$173.11
$3,400.32
$0.00
```

Problem D: The Ninja Way

As we all know, ninjas travel by jumping from treetop to treetop. A clan of ninjas plan to use N trees to hone their tree hopping skills. They will start at the shortest tree and make N-1 jumps, with each jump taking them to a taller tree than the one they're jumping from. When done, they will have been on every tree exactly once, and they will end up on the tallest tree.

The ninjas can travel for at most a certain horizontal distance D in a single jump. To make this as much fun as possible, the Ninjas want to maximize the distance between the planting positions of the shortest and the tallest tree.



The ninjas are going to plant the trees subject to the following constraints:

- All trees are to be planted along a one-dimensional path, which we can regard as the number line.
- Trees must be planted at integer locations along the path, with no two trees at the same location.
- Trees must be arranged so their planted ordering from left to right is the same as their ordering in the input: from $1, 2, \ldots, N$. They must **not** be sorted by height, or reordered in any way. They must be kept in their stated order.
- The Ninjas can only jump so far, so every tree must be planted close to the next tallest tree. In fact, they must be no further than D apart on the ground (the difference in their heights doesn't matter).

Given N trees, numbered $1, 2, \ldots, N$, each with a distinct integer height, help the ninjas figure out the maximum possible distance between the shortest tree and the tallest tree.

Input

Input will consist of multiple datasets. Each dataset begins with a line containing two integers N ($1 \leq N \leq 1000$) and D ($1 \leq D \leq 10^6$).

The next N lines each contains a single integer, giving the heights of the N trees.

The last test case is followed by a line with two zeros.

Output

For each test case, output a line with a single integer representing the maximum possible distance between the planted positions of the shortest and tallest tree, subject to the constraints above, or -1 if it is impossible to lay out the trees. Do not print any blank lines between answers.

Example

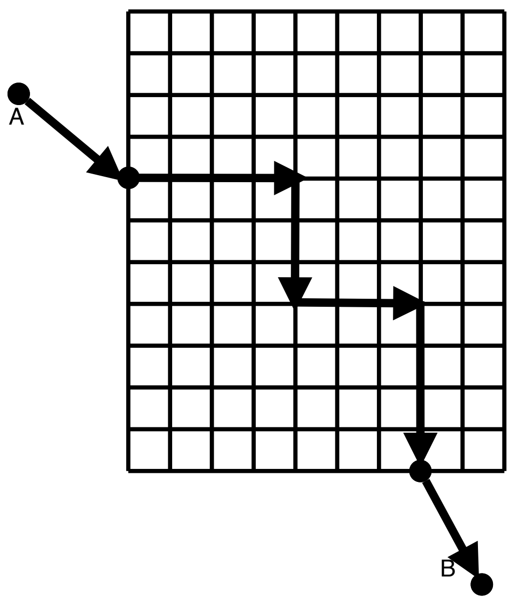
Given the input

```
4 4
20
30
10
40
5 6
20
34
54
10
15
4 2
10
20
16
13
0 0
```

the output would be

```
3
3
-1
```

Problem E: Extended Manhattan Distance



The streets in Manhattan form a grid. If the grid is aligned such that the grid lines are parallel to the x- and y-coordinate axes, the distance one needs to walk or drive from one point to the other, assuming you can only move along streets and cannot take short cuts through buildings, equals $\Delta x + \Delta y$. This is called the *Manhattan distance*.

Now assume that the land outside the city grid is completely flat with no obstacles that prevent moving anywhere. Suppose we want to move from point A to point B where these points can be on the grid or outside the grid. When traveling outside the city, the shortest distance between the two points in this case will not necessarily be the Manhattan distance. It will be the Manhattan distance if the two points are both on the grid. If both points are, for example, north of the grid, the shortest distance between the two points will be the straight-line (Euclidean) distance between them. In other cases, calculating the distance may be more complicated.

In this problem, two opposite corners of the city grid will be specified. It will be assumed that the grid lines are parallel to the coordinate axes, and that the distance between any two consecutive grid lines, horizontal and vertical, is 1 unit. Two points A and B on the plane with integer coordinates will also be specified. Write a program to calculate the shortest distance between the two points, given that we can only move along the grid lines (i.e. in the city streets) within the city grid.

Input

Input will consist of multiple datasets. Each dataset will consist of a single line with eight integers, as follows:

$$x_L \; y_L \; x_U \; y_U \; x_A \; y_A \; x_B \; y_B$$

describing the points L, U, A, and B. L and U are the lower-left corner and the upper-right corner of the city grid, respectively. A and B are the two points between which we wish to travel.

All input integers will be in the range from -1000 to 1000 (inclusive), with $x_L < x_u$ and $y_L < y_U$. End of data will be signified by a line with eight zeros.

Output

For each data set, print one line containing the distance of the shortest path between the A and B, printed to to three decimal places of precision.

Example

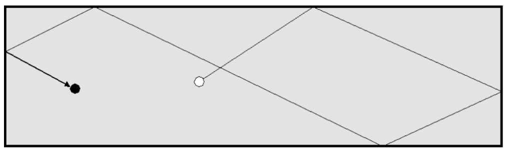
Given the input

```
0 0 4 4 -1 0 5 3
0 0 4 4 2 2 5 3
0 0 0 0 0 0 0 0
```

the output would be

```
7.650
3.414
```

Problem F: Off the Wall



Consider a pool table, and the positions of a cue ball and a target ball. The cue ball must bounce off of a certain number of cushions (i.e. edges of the table), and then hit the target ball. What is the minimum distance that the cue ball has to travel?

Assume ideal cushions with perfect reflection (i.e., the angle at which a ball strikes the cushion is equal to the angle at which it bounces away), and a negligible ball diameter. The coordinate system uses a corner of the table as the origin, and the edges of the table are aligned with the coordinate axes. If the cue ball hits in a corner, it is considered to be hitting two cushions. The cue ball must hit exactly the right number of cushions first, before hitting the target ball.

Input

Input will consist of multiple datasets. Each dataset is on a single line containing seven integers:

$$L\ W\ x_C\ y_C\ x_T\ y_T\ N$$

The first two integers, L and W ($2 \leq L, W \leq 100$), are the dimensions of the table.

The next two pairs of integers are the x,y coordinates of the cue and target balls, respectively. You are guaranteed that $0 < x_C < L$, $0 < x_T < L$, $0 < y_C < W$, and $0 < y_T < W$. C and T are distinct points.

The final integer N, ($0 \leq N \leq 100$), is the number of cushions that must be hit prior to striking the target ball.

End of input will be indicated by a line with seven zeros.

Output

For each dataset, print a line with a single real number to 3 decimal digits precision, representing the shortest distance the cue ball must travel.

Example

Given the input

```
20 15 10 1 12 1 1
10 20 1 2 7 16 2
100 100 2 50 1 50 1
0 0 0 0 0 0 0
```

the output would be

```
2.828
19.698
100.005
```

Problem G: Stringer

Imagine a list of strings that are all built from the first N letters of the alphabet, which all have a predetermined number of a's, a predetermined (but possibly different) number of b's, and so on. Imagine that it's sorted in alphabetical order, and numbered, starting at 0. What's the Kth string in the list?

For example, look at all strings of a's and b's (N=2) with 2 a's and 3 b's:

0: `aabbb`	5: `babab`
1: `ababb`	6: `babba`
2: `abbab`	7: `bbaab`
3: `abbba`	8: `bbaba`
4: `baabb`	9: `bbbaa`

If K=5, then the K^th string in the list would be `babab`.

Input

Input will consist of multiple datasets. Each dataset consists of two lines.

On the first line are two integers, N ($1 \leq N \leq 20$) and K ($0 \leq K < m$), where N is the number of letters of the alphabet used, K is the index of the list element that should be found, and m (not given explicitly in the input) denotes the number of strings make up the list.

m, the number of strings in the list may be very, very large (too large to permit generating the whole list), but the input will be chosen so that m and K will each fit in a 32 bit integer.

On the second line will be N non-negative integers, which represent the number of a's, the number of b's and so on. The sum of these integers is guaranteed to be at least 1 and no greater than 50.

End of input is indicated by a line with two zeros.

Output

Output each answer string on its own line. Do not print any extra white space. Do not print any blank lines between answers.

Example

Given the input

```
2 5
2 3
3 0
2 3 1
0 0
```

the output would be

```
babab
aabbbc
```

Problem H: Word ladder

You now work for a puzzle company. They have a puzzle they call a *Word Ladder*. A solver starts with a given starting word, and makes changes one letter at a time until s/he reaches a target word, with no word in the chain appearing more than once. There are three ways to take a single step from one word to another:

- Change one letter
- Add one letter
- Remove one letter

So, it's one step from COT to CAT, one step from CAT to SCAT, and one step from SCAT to SAT. Here's a word ladder from COT to SCAT:

COT $\Rightarrow$ CAT $\Rightarrow$ SAT $\Rightarrow$ SCAT

Here's another word ladder from COT to SCAT:

COT $\Rightarrow$ CAT $\Rightarrow$ SCAT

The length of a word ladder is the number of words in it, so the examples above show a word ladder of length 4, and one of length 3. The second is the shortest possible between COT and SCAT. Shorter ladders are considered better than longer ladders.

The puzzle company knows that, given two words, a smart solver will always find the best ladder, which is the shortest ladder, between them. They want to give their solvers a challenge, so they are looking for long word ladders. Given a limited vocabulary, you need to tell them the length of the longest word ladder that a smart solver would find using only words in that vocabulary - that is, the longest of all best ladders.

Input

Input will consist of multiple datasets. Each dataset starts with an integer N ($1 \leq N \leq 500$) which indicates the number of words in the vocabulary. The words follow, one per line.

Each word will consist only of 1 to 50 lower-case letters. There will be no other characters or white space.

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

Output

For each input set, print a line containing a single integer representing the length of the longest ladder that a smart solver would find.

Example

Given the input

```
4
cat
cot
scat
sat
7
welcome
to
the
acm
regional
programming
contest
0
```

the output would be

```
3
1
```