acm International Collegiate Programming Contest

IBM event sponsor

2009 ACM ICPC Southeast USA Regional Programming Contest

7 November, 2009

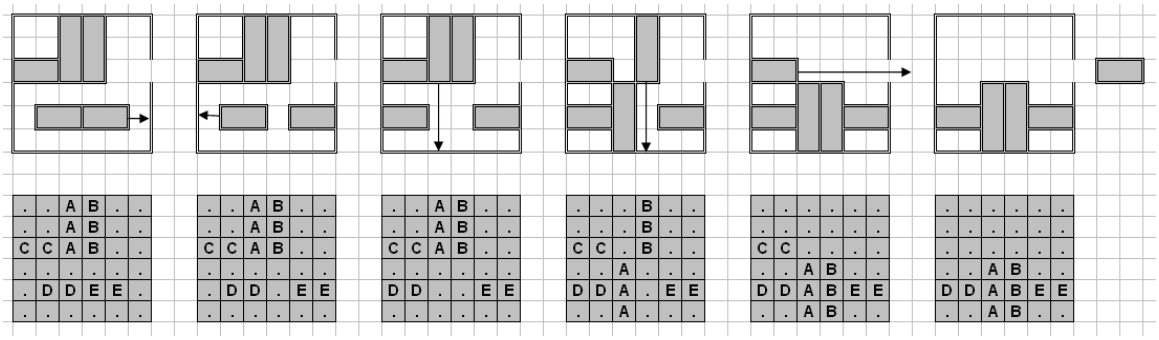
PROBLEMS

Hosted by:

Florida Institute of Technology
Armstrong Atlantic State University
University of South Alabama

A: Block Game

Bud bought a new board game. He is hooked. He has been playing it over and over again, and he thinks can solve any board with the minimum number of moves, but he is uncertain. He wants you to write a program to calculate the minimum number of moves required to solve different boards, so that he can double check his answers.



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 may slide a piece if there are no other pieces, nor walls, 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.

Sliding a piece any number of squares is considered one move. (i.e. sliding a piece horizontally one square is one move, and sliding it two squares at once is also considered one move).

Input

There will be several test cases. Each test case will begin with a line with a single capital letter, indicating the special piece which must be moved 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 test case, print a single integer, indicating the smallest number of moves necessary to remove the given special piece, or **-1** if it isn't possible. Print each integer on its own line. There should be no blank lines between answers.

Sample Input

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

Sample Output

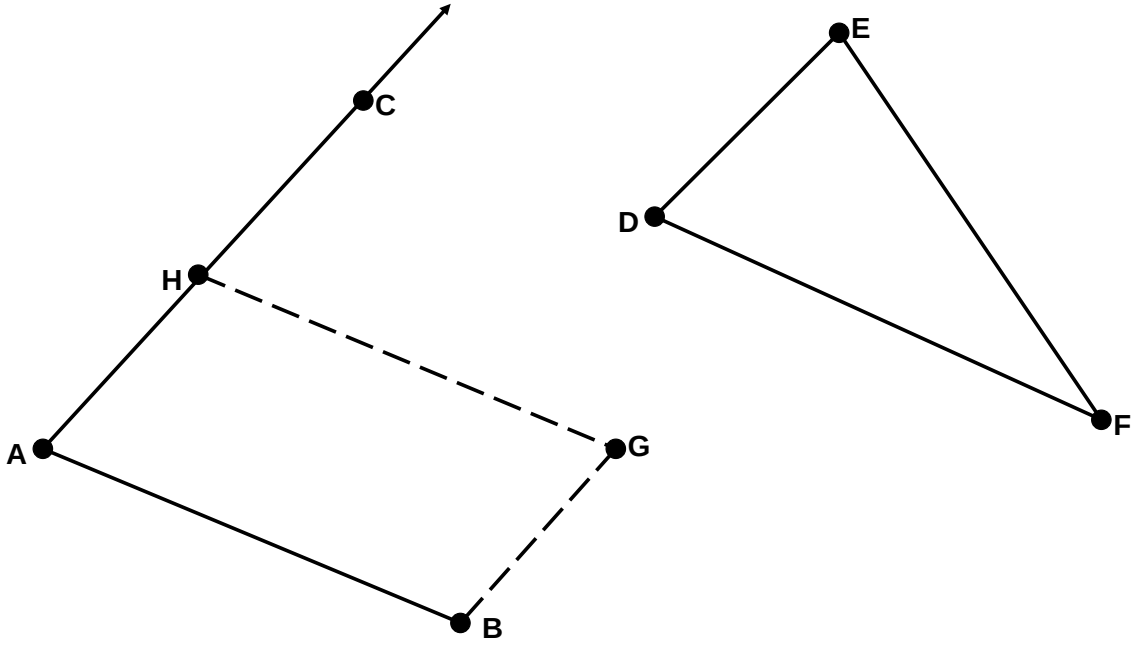
```
5
1
-1
```

B: 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 **HG**, **AH** is parallel to **BG**)
- The area of parallelogram **ABGH** is the same as the area of triangle **DEF**



The Input

There will be several test cases. Each test case will consist of twelve real numbers, with no more than 3 decimal places each, on a single line. Those numbers will represent, in order:

AX AY BX BY CX CY DX DY EX EY FX FY

where point **A** is (**AX**,**AY**), point **B** is (**BX**,**BY**), and so on. 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** to **1000.0** inclusive. End of the input will be signified by a line with twelve **0.0**'s.

The Output

For each test case, print a single line with four decimal numbers. These represent points **G** and **H**, like this:

GX GY HX HY

where point **G** is (**GX**,**GY**) and point **H** is (**HX**,**HY**). Print all values rounded to 3 decimal places of precision (NOT truncated). Print a single space between numbers. Do not print any blank lines between answers.

Sample 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
```

Sample Output

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

C: Museum Guards

A museum has hired some guards and is trying to build a schedule for them. The museum would like to build a 24-hour schedule for the guards, such that:

- Each guard works during the same time intervals every day.
- Each guard works within his/her time windows of availability, which s/he specifies.
- Each guard works at most the amount of time s/he is able, which s/he also specifies.
- The guards can only shift (start or stop working) on half hour boundaries. (e.g. 04:00 or 04:30, but not 04:15).
- The guards are only scheduled for shifts if they are available to work at all times during those shifts. (e.g. if a guard's window of availability opens at 03:05, they cannot be scheduled at 03:00.)
- The minimum number of guards on duty at any time during the day is maximized. This improves security of the museum.

Write a program to help the museum staff determine the maximum number of guards that they can maintain at all times throughout a 24-hour day, given the constraints of the guards' availability. You may assume that guard exchanges are instantaneous. That is, if 2 guards leave and 2 other guards arrive at the same time, the museum is guarded by 2 guards through this exchange.

Input

There will be multiple test cases. Each test case begins with a line containing a single integer **N** (**1 ≤ N ≤ 50**), the number of guards available. There will then be **N** blocks of data, one for each guard.

Each block provides the preferences of one guard. A block begins with two integers, **K** (**1 ≤ K ≤ 50**) and **M** (**1 ≤ M ≤ 1440**), in that order; **K** is the number of time intervals specifying when the guard is available for work, and **M** is the maximum number of minutes s/he is able to work each day. The next **K** lines each contains the starting and ending time, in that order, of a time interval where the guard is available, separated by whitespace. These time intervals may overlap. The *union* of all **K** time intervals provides the complete set of times at which the guard is available.

A starting or ending time is formatted as **HH:MM** (**00 ≤ HH ≤ 23**, **00 ≤ MM ≤ 59**). Midnight is represented by **00:00**. When the ending time is smaller than the starting time, it means the guard is available for working past midnight. For example, the interval "**23:00 03:00**" means the guard is available from 11pm at night to 3am in the morning. If the starting and ending times are equal, then the

guard is available for work during any time intervals throughout the day. The last test case is followed by a line with a single `0`.

Output

For each test case, output a single integer, representing the minimum number of guards on duty at any given time during the day, using a schedule that maximizes this value. That is, output the largest integer k, such that there is a schedule where at any given moment, there are k guards on duty at the museum, assuming instantaneous exchanges specified above. Do not print any blank lines between answers.

Sample Input

```
3
1 540
00:00 00:00
3 480
08:00 10:00
09:00 12:00
13:00 19:00
1 420
17:00 00:00
5
1 720
18:00 12:00
1 1080
00:00 23:00
1 1080
00:00 20:00
1 1050
06:00 00:00
1 360
18:00 00:00
3
1 1440
00:00 00:00
1 720
00:00 12:15
1 720
12:05 00:15
0
```

Sample Output

```
1
2
1
```

D: Knitting

Marcia loves to knit. As she knits, she wonders how many stitches the project she is working on will take to complete.

On every project, she starts with a row of a given number of stitches, and then adds more rows. Sometimes the next row will have the same number of stitches as the previous row and other times the next row will have more or less stitches than the previous row.

For example, a pattern for a triangular shawl may begin with just 3 stitches and add 2 stitches on each row. So, the first row will have 3 stitches, the second row will have 5 stitches, the third will have 7 stitches, and so on. If the project has a total of 3 rows, then it has a total of 15 stitches.

A more complex scarf project may have a 4 row repeating pattern that increases 6 stitches on the first row of the pattern, decreases 2 stitches on each of the next two rows, and has no change on the final row of the pattern. So, a scarf that has 50 stitches on the first row will have 56 on the second row, 54 on the third row, 52 on the fourth row, and 52 on the fifth row. On the sixth row, the pattern repeats, so there will be an increase of 6 stitches for a total of 58 stitches on that row. If the project stops there at 6 rows, then it will have a total of 322 stitches.

You will write a program to help Marcia figure out how many stitches a project will take to complete.

Input

The input to your program will be information about one or more projects. Each project's description will take up 2 lines. The first line contains three integers:

N M K

Where **N** (**1 ≤ N ≤ 100**) represents the number of stitches in the first row of the project, **M** (**1 ≤ M ≤ 1000**) represents the total number of rows in the project, and **K** (**1 ≤ K ≤ 100**) represents the number of rows in the repeating pattern. On the following line will be exactly **K** integers, each one in the range from **-100** to **100** (inclusive), indicating the repeating pattern, where negative values indicate a number of stitches to decrease, positive values indicate a number of stitches to increase, and 0 indicates no change. In any project, the pattern will never cause any row to have 0 or fewer stitches. End of input is indicated by a line with three **0**'s.

Output

For each project, give the total number of stitches in the completed project. Print each integer answer on its own line, with no blank lines between answers.

Sample Input

```
3 3 1
2
50 6 4
6 -2 -2 0
0 0 0
```

Sample Output

```
15
322
```

E: Minesweeper

Minesweeper is a game played on a **R** x **C** rectangular board. Some of the cells contain mines, and others are empty. For each empty cell, calculate the number of its adjacent cells that contain mines. Two cells are adjacent if they share a common edge or point. This means that each cell has a maximum of 8 neighbors (up, down, left, right, four diagonals).

Input

There will be multiple test cases. The first line of each test case will have two integers, **R** and **C** (**1 ≤ R,C ≤ 100**), indicating the number of rows and columns of the board. The next **R** lines each contain exactly **C** characters. Each character is either a '*****' (asterisk) indicating a mine, or a '**.**' (period) indicating an empty cell. The last data set is followed by a line containing two **0**'s.

Output

Print each board on **R** lines with **C** characters per line, and replace every '**.**' with the appropriate digit indicating the number of adjacent cells that contain mines. Leave the '*****' cells intact. Do not print any whitespace between cells. Do not print any blank lines between answers.

Sample Input

```
3 2
..
.*
..
5 5
*.*.*
..*..
*****
.....
..**.
0 0
```

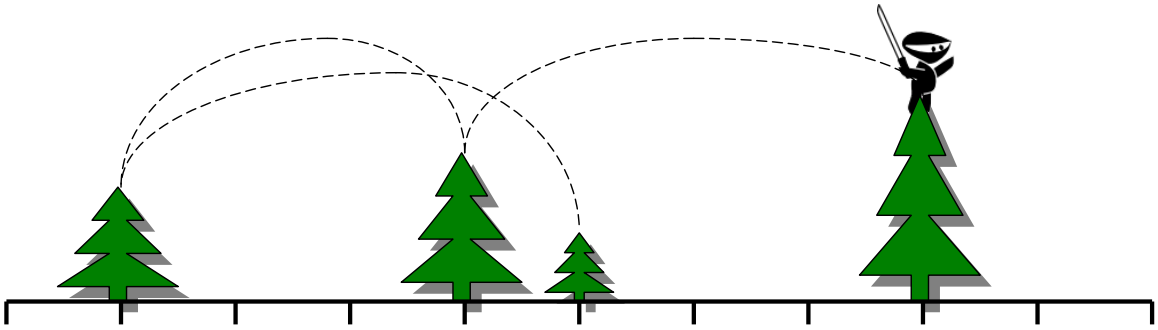
Sample output

```
11
1*
11
*3*3*
36*63
*****
24553
01**1
```

F: The Ninja Way

As we all know, Ninjas travel by jumping from treetop to treetop. A clan of Ninjas plans 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 finished, they will have been on every tree exactly once, traversing them in increasing order of height, and ending 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 positions of the shortest tree 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.
- 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. 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 enough to the next taller tree. Specifically, they must be no further than **D** apart on the ground (the difference in their heights doesn't matter).

Given **N** trees, in a specified order, each with a distinct integer height, help the ninjas figure out the maximum possible distance they can put between the shortest tree and the tallest tree, and be able to use the trees for training.

Input

There will be multiple test cases. Each test case begins with a line containing two integers **N** ($1 \le N \le 1000$) and **D** ($1 \le D \le 10^6$). The next **N** lines each contain a single integer, giving the heights of the **N** trees, in the order that they should be planted. Within a test case, all heights will be unique. The last test case is followed by a line with two **0**'s.

Output

For each test case, output a line with a single integer representing the maximum distance between 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.

Sample Input

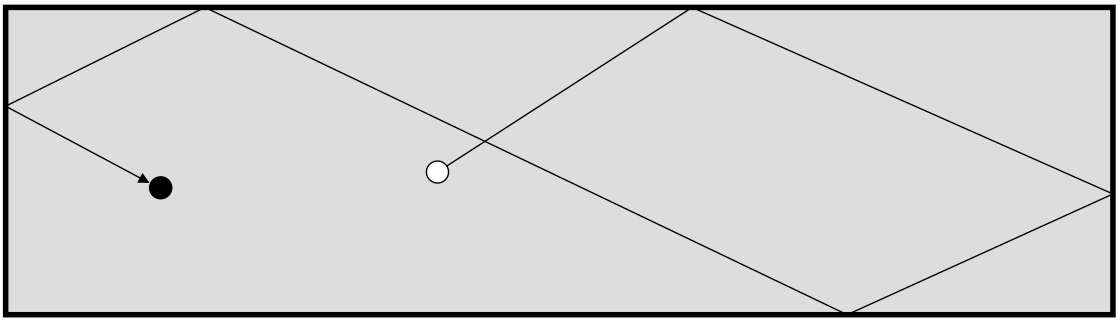
```
4 4
20
30
10
40
5 6
20
34
54
10
15
4 2
10
20
16
13
0 0
```

Sample Output

```
3
3
-1
```

G: Pool Table

Consider a pool table with 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 (i.e., laws of reflection apply), 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 correct number of cushions first, *before* hitting the target the ball.

Input

There will be multiple test cases. Each case is on a single line containing seven integers:

L W CX CY TX TY N

The first two integers, **L** and **W** (**2 ≤ L,W ≤ 100**), are the dimensions of the table. The next two pairs of integers are the coordinates (**X**, **Y**) of the cue and target balls, such that **0** < **CX,TX** < **L**, and **0** < **CY,TY** < **W**, and (**CX**,**CY**) is not the same as (**TX**,**TY**). The last integer **N**, (**0 ≤ N ≤ 100**), is the number of cushions that must be hit. The test cases will be followed by a line with seven **0**'s.

Output

For each test case, print a single decimal number, rounded (NOT truncated) to 3 decimal places, representing the shortest distance the cue ball must travel. Print each answer on its own line, with no blank lines between answers.

Sample Input

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

Sample Output

```
2.828
19.698
```

H: Robot Challenge

You have entered a robot in a Robot Challenge. A course is set up in a 100m by 100m space. Certain points are identified within the space as targets. They are ordered – there is a target 1, a target 2, etc. Your robot must start at (0,0). From there, it should go to target 1, stop for 1 second, go to target 2, stop for 1 second, and so on. It must finally end up at, and stop for a second on, (100,100).

Each target except (0,0) and (100,100) has a time penalty for missing it. So, if your robot went straight from target 1 to target 3, skipping target 2, it would incur target 2's penalty. Note that once it hits target 3, it cannot go back to target 2. It must hit the targets in order. Since your robot must stop for 1 second on each target point, it is not in danger of hitting a target accidentally too soon. For example, if target point 3 lies directly between target points 1 and 2, your robot can go straight from 1 to 2, right over 3, without stopping. Since it didn't stop, the judges will not mistakenly think that it hit target 3 too soon, so they won't assess target 2's penalty. Your final score is the amount of time (in seconds) your robot takes to reach (100,100), completing the course, plus all penalties. Smaller scores are better.

Your robot is very maneuverable, but a bit slow. It moves at 1 m/s, but can turn very quickly. During the 1 second it stops on a target point, it can easily turn to face the next target point. Thus, it can always move in a straight line between target points.

Because your robot is a bit slow, it might be advantageous to skip some targets, and incur their penalty, rather than actually maneuvering to them. Given a description of a course, determine your robot's best (lowest) possible score.

The Input

There will be several test cases. Each test case will begin with a line with one integer, **N** (**1 ≤ N ≤ 1000**) which is the number of targets on the course. Each of the next **N** lines will describe a target with three integers, **X**, **Y** and **P**, where (**X,Y**) is a location on the course (**1 ≤ X,Y ≤ 99**, **X** and **Y** in meters) and **P** is the penalty incurred if the robot misses that target (**1 ≤ P ≤ 100**). The targets will be given in order – the first line after **N** is target 1, the next is target 2, and so on. All the targets on a given course will be unique – there will be at most one target point at any location on the course. End of input will be marked by a line with a single **0**.

The Output

For each test case, output a single decimal number, indicating the smallest possible score for that course. Output this number rounded (NOT truncated) to three decimal places. Print each answer on its own line, and do not print any blank lines between answers.

Sample Input

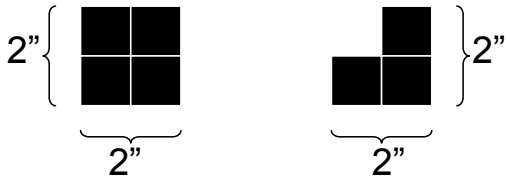
```
1
50 50 20
3
30 30 90
60 60 80
10 90 100
3
30 30 90
60 60 80
10 90 10
0
```

Sample Output

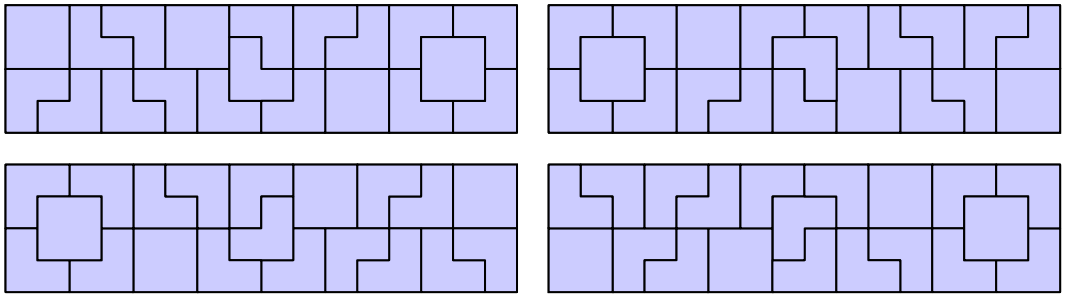
```
143.421
237.716
154.421
```

I: Mosaic

An architect wants to decorate one of his buildings with a long, thin mosaic. He has two kinds of tiles available to him, each 2 inches by 2 inches:



He can rotate the second kind of tile in any of four ways. He wants to fill the entire space with tiles, leaving no untiled gaps. Now, he wonders how many different patterns he can form. He considers two mosaics to be the same only if they have exactly the same kinds of tiles in exactly the same positions. So, if a rotation or a reflection of a pattern has tiles in different places than the original, he considers it a different pattern. The following are examples of 4” x 16” mosaics, and even though they are all rotations or reflections of each other, the architect considers them to be four different mosaics:



Input

There will be several test cases. Each test case will consist of two integers on a single line, **N** and **M** (**2 ≤ N ≤ 10**, **2 ≤ M ≤ 500**). These represent the dimensions of the strip he wishes to fill, in inches. The data set will conclude with a line with two 0’s.

Output

For each test case, print out a single integer representing the number of unique ways that the architect can tile the space, **modulo 10^6**. Print each integer on its own line, with no extra whitespace. Do not print any blank lines between answers.

Sample Input

```
2 10
4 16
4 50
0 0
```

Sample Output

```
25
366318
574777
```