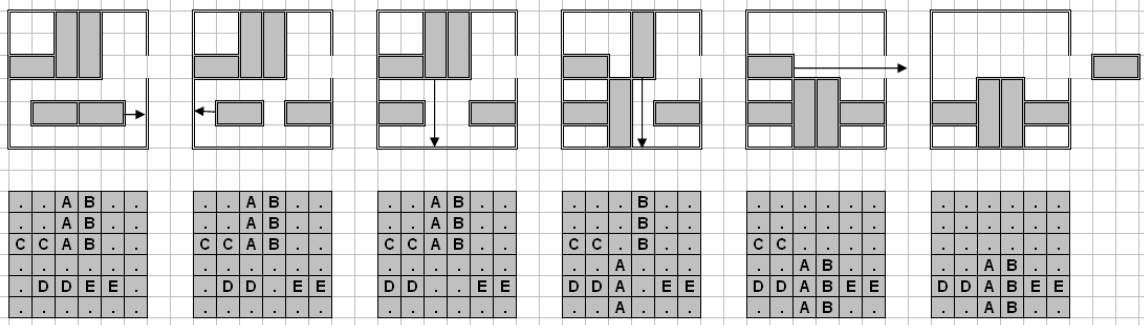




A: Block Game

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
```

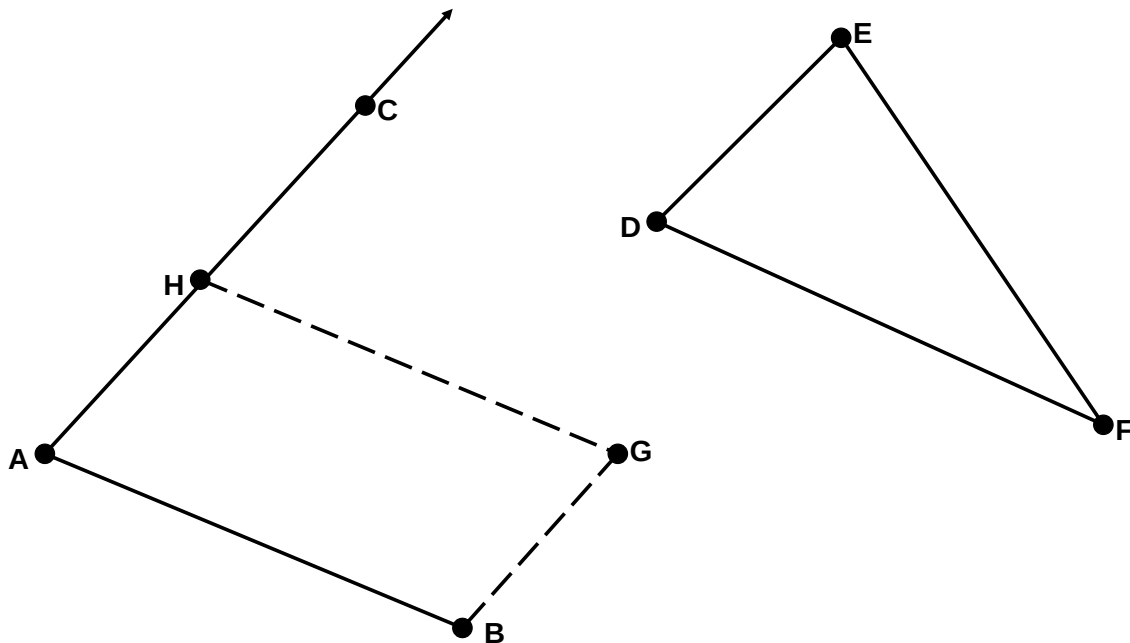


B: Euclid

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 $\mathbf{AB}$, another point $\mathbf{C}$ which is not collinear with $\mathbf{AB}$, and a triangle $\mathbf{DEF}$. The goal is to find points $\mathbf{G}$ and $\mathbf{H}$ such that:

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



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:

$\mathbf{AX\ AY\ BX\ BY\ CX\ CY\ DX\ DY\ EX\ EY\ FX\ FY}$

where point $\mathbf{A}$ is $(\mathbf{AX}, \mathbf{AY})$, point $\mathbf{B}$ is $(\mathbf{BX}, \mathbf{BY})$, and so on. Points $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are guaranteed to NOT be collinear. Likewise, $\mathbf{D}$, $\mathbf{E}$ and $\mathbf{F}$ 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

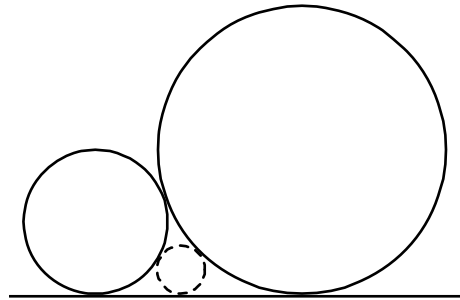
```
5.000 0.800 0.000 0.800
13.756 7.204 2.956 5.304
```

Hidden Circle

Filename: circle

Ali loves difficult circle problems. Whether training in the NASA Centrifuge for the astronaut corps, organizing his CD collection or building fancy clocks, nothing makes Ali happier than a circle problem! However, Raymond has decided that he wants to challenge Ali for the circular crown given to the person who makes the most difficult circle problem. Of course, Ali does not want to relinquish his crown. He wants you to succeed in solving Raymond's challenge, which is stated below. Don't let Ali down!

Given the radii of two tangent circles sitting on the ground, Ali wants you to determine the radius of the circle that is tangent to the two given circles and the ground as illustrated in the diagram below:



The Problem:

Given the radii of the two large circles in the diagram above, determine the radius of the circle that would fit between them, tangent to each large circle and the ground. If you need to use π , use a value of 3.1415926535.

The Input:

Input will begin with a single, positive integer, n , on a line by itself, representing the number of circle problems to solve. Each problem will be on a single line. Each line will contain only two positive integers separated by a single space, r ($r < 100$) and s ($s < 100$), representing the radii of the two circles, respectively.

The Output:

For each circle problem in the input, output a line with the following format:

Circle Problem # k : Radius of the small circle = c

where k ($1 \leq k \leq n$) represents the number of circle problem and c is the desired radius rounded to two decimal places. For example, 3.454 would round to 3.45 while 3.455 would round to 3.46.

Sample Input:

```
3
1 1
1 2
19 88
```

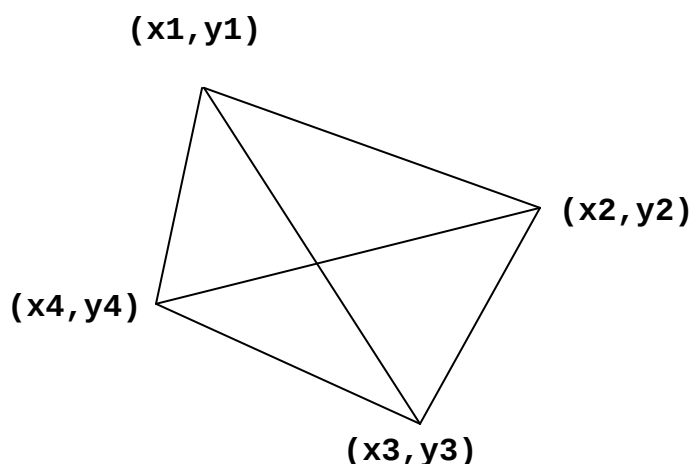
Sample Output:

```
Circle Problem #1: Radius of the small circle = 0.25
Circle Problem #2: Radius of the small circle = 0.34
Circle Problem #3: Radius of the small circle = 8.86
```

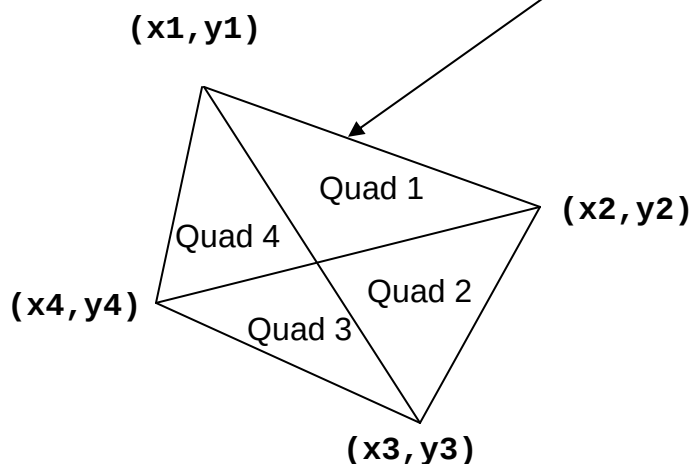


D: Shoring Up the Levees

The tiny country of Waterlogged is protected by a series of levees that form a quadrilateral as shown below:



The quadrilateral is defined by four vertices. The levees partition the country into four quadrants. Each quadrant is identified by a pair of vertices representing the outside edge of that quadrant. For example, Quadrant 1 shown below is defined by the points (x_1, y_1) and (x_2, y_2) .



It happens very often that the country of Waterlogged becomes flooded, and the levees need to be reinforced, but their country is poor and they have limited resources. They would like to be able to reinforce those levees that encompass the largest area first, then the next largest second, then the next largest third, and the smallest area fourth.



Help Waterlogged identify which quadrants are the largest, and the length of the levees around them

Input

There will be several sets of input. Each set will consist of eight real numbers, on a single line. Those numbers will represent, in order:

$X1$ $Y1$ $X2$ $Y2$ $X3$ $Y3$ $X4$ $Y4$

The four points are guaranteed to form a convex quadrilateral when taken in order – that is, there will be no concavities, and no lines crossing. Every number will be in the range from **-1000.0** to **1000.0** inclusive. No Quadrant will have an area or a perimeter smaller than **0.001**. End of the input will be a line with eight **0.0**'s.

Output

For each input set, print a single line with eight floating point numbers. These represent the areas and perimeters of the four Quadrants, like this:

$A1$ $P1$ $A2$ $P2$ $A3$ $P3$ $A4$ $P4$

Print them in order from largest area to smallest – so **$A1$** is the largest area. If two Quadrants have the same area when rounded to 3 decimal places, output the one with the largest perimeter first. Print all values with 3 decimal places of precision (rounded). Print spaces between numbers. Do not print any blank lines between outputs.

Sample Input

```
1 2 1 5 5 2 2 0
3.5 2.2 4.8 -9.6 -1.2 -4.4 -8.9 12.4
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

Sample Output

```
5.100 11.459 3.400 9.045 0.900 6.659 0.600 4.876
44.548 38.972 21.982 25.997 20.342 38.374 10.038 19.043
```

UCF Local Contest — September 3, 2011

Cut the Cake!

filename: polycake

On the faraway planet of Gastronomica, the native Gastronomes consider polygonal pancakes to be the highest form of art (as well as the tastiest). Every year, they celebrate the Polycake Festival, in which thousands of ninja chef monasteries perform the Thousand Slices, a series of rituals of tremendous importance to Gastronome society. In each of these rituals, a polycake is carefully placed on a ceremonial altar, and a ninja chef will solemnly cut the cake with a single, perfectly straight slice, dividing it into two pieces, one on the North part of the altar (representing the past) and the other on the South part (representing the future). The two pieces are carefully separated, and their perimeters measured. The results will determine the proportions of the ingredients used in polycakes planetwide, until the next Polycake Festival.

As the foremost Novice at the Temple of the Promised Cosmic Polycake, you are entrusted the task of measuring the perimeters of the two slices. Take care, for any mistakes will be mercilessly mocked by the Brotherhood of the Illusory Polycake¹. It would be best if you wrote a program to do this.

The Problem:

Given a polygon representing the polycake in the Cartesian plane and a line parallel to the X-axis indicating the position of the slice, you are to compute the perimeter of each of the two pieces thus formed. The perimeter of a polygon is defined as the sum of the lengths of all its sides. Assume that the input polygons will be convex (i.e., not concave) and simple (i.e., not complex, not intersecting).

The Input:

The first input line contains a positive integer, n , indicating the number of polycakes used in the ritual. This is followed by n data sets, each representing a single slicing of a polycake.

The first line of each set consists of two integers, V ($3 \leq V \leq 10$) and Y ($-1000 \leq Y \leq 1000$), representing the number of vertices in the polycake and the y-coordinate of the horizontal cut, respectively. (Recall that the equation of a line parallel to the X-axis is of the form $y=k$.)

The next V lines each contain two integers, x and y ($-1000 \leq x, y \leq 1000$), the Cartesian coordinates of the vertices of the polycake, in counter-clockwise order.

To minimize complications (and following the rules of the ritual), it is guaranteed that the cut will never pass through a vertex.

¹Reviled heretics who claim that the Cosmic Polycake is a lie.

The Output:

For each test case, first output “Case # i :” where i is the test case number, starting with 1. Then, output “ a b ”, the perimeters of the two slices, in increasing order. Output the results to 3 decimal places, rounded to the nearest thousandth (e.g., 77.0113 should round to 77.011, 88.0115 should round to 88.012, and 99.0117 should round to 99.012). Leave a blank line after the output for each test case. Follow the format illustrated in Sample Output.

Sample Input:

```
2
4 2
0 0
4 0
4 4
0 4
6 10
3 15
10 1
12 5
11 19
9 23
6 20
```

Sample Output:

```
Case #1: 12.000 12.000

Case #2: 25.690 35.302
```

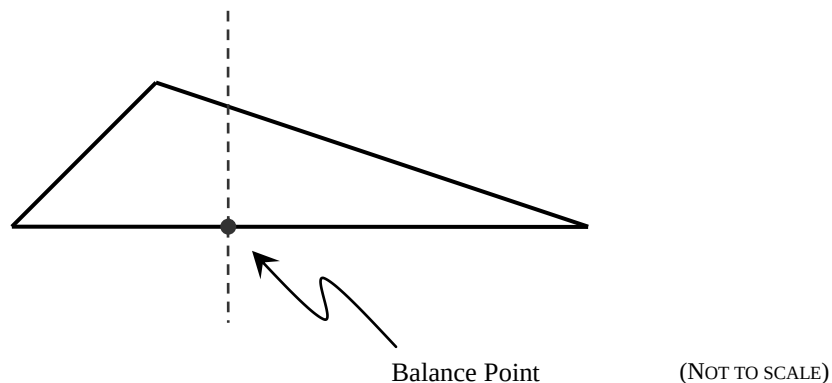
UCF Local Contest — September 5, 2009

Balance Point

filename: balance

Given an object, it's often fun to find the point at which you can balance it without it toppling over to one side. This is typically a very difficult problem, but we'd like to at least solve the problem in two dimensions for a simple shape: a scalene triangle. In particular, we will define the balance point of a scalene triangle as follows:

Find the longest side of the triangle. Then, determine the perpendicular line to this side that divides the triangle's area into two equal parts. The point at which this line intersects the longest side is known as the triangle's balance point.



The Problem:

Given the Cartesian coordinates of the three points of a scalene triangle, determine the x and y coordinates of its balance point (to 2 decimal places).

The Input:

There will be multiple triangles in the input file. The first input line contains a positive integer n , indicating the number of triangles to be processed. The triangles will be on the following n input lines, each on a separate line. Each triangle contains three pairs of x - y coordinates, denoting the three points of the triangle. Each of these six values will be real numbers in between -100 and 100, inclusive, separated by spaces. Assume that all three points are distinct and that they are not collinear, i.e., assume the points form a triangle.

The Output:

At the beginning of each test case, output "Triangle # t Balance Point: ", where t is the test case number (starting from 1). Following this header, for each triangle, print the x and y coordinates of that triangle's balance point, rounded to two decimal places, in parentheses,

separated by a comma. To clarify “rounded to two decimal places”: the output for 1.274 should be 1.27, for 1.275 should be 1.28, and for 1.276 should be 1.28.

Follow the format illustrated in Sample Output.

Sample Input:

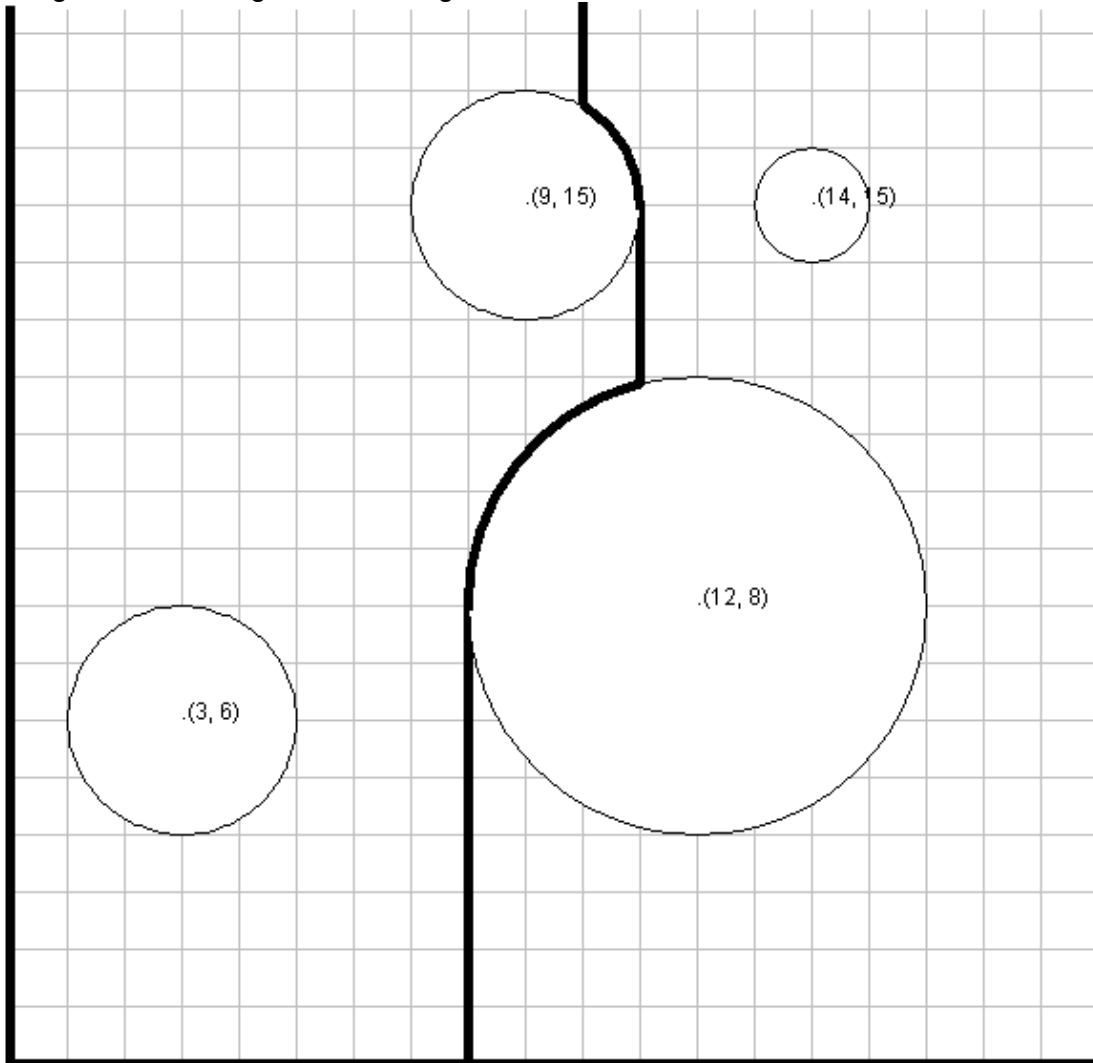
```
2
0.0 0.0 5.0 0.0 1.8 2.4
0.0 0.0 0.0 5.0 -2.4 3.2
```

Sample Output:

```
Triangle #1 Balance Point: (2.17,0.00)
Triangle #2 Balance Point: (0.00,2.83)
```

Raindrops (prob10)

There is a world filled with rods of varying thicknesses. A raindrop falls from above these rods and either falls directly to the ground (if there is no rod directly below it) or hit a rod and runs along it before falling to either the ground or another rod.



Your program will take a configuration of rods and will determine the total distance a raindrop will fall, given its starting position.

Input

The input to this program will consist of one or more test sets.

Each test set will begin with a pair of integers, *rods* and *drops*, $0 \leq rods \leq 100$ and $0 \leq drops \leq 100$. There will then be *rods* lines, each defining one rod. These lines will consist of three integers: *center-x*, *center-y*, *radius*, $0 \leq center-x, center-y \leq 1000$ and $1 \leq radius \leq 100$, where *center-x* and *center-y* represent the center of the rod and *radius* represents its radius. No rod will touch the ground and no two rods will touch each other (treat rods as solid disks so a rod will not be inside another rod) .

There will then be *drops* lines, each with a single integer representing the x position of the raindrop at the start. All drops should be assumed to start at height 1200.

The end of input is represented by a test set with *drops* = 0.

Output

For each test set, have a line with the number of the test set, starting at 1. Then, for each drop, give the total distance the drop falls before reaching the ground, rounded to 4 decimal places. Obviously, drops never fall upward. In the case a drop falls on the highest point of a rod, it will move to the left (smaller x) side of the rod.

Sample Input

```
4 1
3 6 2
9 15 2
12 8 4
14 15 1
10
0 2
17
19
3 0
```

Sample Output

```
Test set 1
1200.1975—1201.7618

Test set 2
1200.0000
1200.0000
```



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 ($1 \leq N \leq 1000$) 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 ($1 \leq x, y \leq 99$, x and y in meters) and p is the penalty incurred if the robot misses that target ($1 \leq p \leq 100$). 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
```