

2011-1004 UM Online Team Programming Contest
October 4, 2011

Hosted by:
University of Michigan

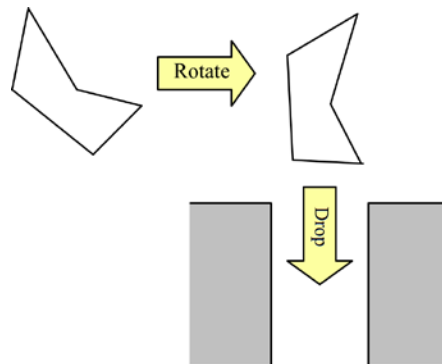
Rules:

1. There are a few questions to be solved any time during the contest.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
6. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
7. All communication with the judges will be handled by the PC2 environment.
8. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: Trash Removal

Allied Chute Manufacturers is a company that builds trash chutes. A trash chute is a hollow tube installed in buildings so that trash dropped in at the top will fall down and be collected in the basement. Designing trash chutes is actually highly nontrivial. Depending on what kind of trash people are expected to drop into them, the trash chute needs to have an appropriate size. And since the cost of manufacturing a trash chute is proportional to its size, the company always would like to build a chute that is as small as possible. Choosing the right size can be tough though.

We will consider a 2-dimensional simplification of the chute design problem. A trash chute points straight down and has a constant width. Objects that will be dropped into the trash chute are modeled as polygons. Before an object is dropped into the chute it can be rotated so as to provide an optimal fit. Once dropped, it will travel on a straight path downwards and will not rotate in flight. The following figure shows how an object is first rotated so it fits into the trash chute.



Your task is to compute the smallest chute width that will allow a given polygon to pass through.

Input

The input contains several test cases. Each test case starts with a line containing an integer n ($3 \leq n \leq 100$), the number of points in the polygon that models the trash item.

The next n lines then contain pairs of integers x_i and y_i ($0 \leq x_i, y_i \leq 10^4$), giving the coordinates of the polygon vertices in order. All points in one test case are guaranteed to be mutually distinct and the polygon sides will never intersect. (Technically, there is one inevitable exception of two neighboring sides sharing their common vertex. Of course, this is not considered an intersection.)

The last test case is followed by a line containing a single zero.

Output

For each test case, display its case number followed by the width of the smallest trash chute through which it can be dropped. Display the minimum width with exactly two digits to the right of the decimal point, rounding up to the nearest multiple of $1/100$. Answers within $1/100$ of the correct rounded answer will be accepted.

Follow the format of the sample output.

Sample Input

```
3
0 0
3 0
0 4
4
0 10
10 0
20 10
10 20
0
```

Sample Output

```
Case 1: 2.40
Case 2: 14.15
```

Problem B: Going Tubing

Mark is going tubing in a large waterpark. The park consists of N pools ($2 \leq N \leq 5000$) connected by M one-way chutes ($1 \leq M \leq 10000$). Each chute i takes T_i ($1 \leq T_i \leq 10^{12}$) seconds to use, while crossing each pool takes no time. Because the chutes are powered by jets shooting water, it is possible to have cycles in the graph. He starts at pool 1 and would like to reach the end, pool N , as quickly as possible so he can go again.

However, he also wants to splash the park operators when he arrives at the ending pool. Every K seconds ($1 \leq K \leq 10^{18}$), starting at the K th second, the park operators change shifts. During the R seconds ($1 \leq R \leq 20$) it takes them to change shifts, no operator is within range of Mark's splashing prowess.

Find the length T of the shortest path from pool 1 to pool N such that T is not of the form $(a \cdot K + b)$ for integer a and b , where $0 \leq b < R$.

Input

There will be multiple input cases. Each test case will have:

Line 1: Four space-separated integers: N , M , K , and R .

Lines 2.. $M + 1$: Line $i+1$: three integers, A_i , B_i , and T_i , denoting a chute from pool A_i to pool B_i of length T_i .

Last line will contain 0 0 0 0 to indicate end of input.

Output

For each test case A single integer, the length of the shortest path that does not have length in $\{0, 1, \dots, R-1\}$ when reduced modulo K . If no such path exists, output 0.

Sample Input

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

Sample Output

```
8
```

Sample Explanation

Mark's path is 1 -> 3 -> 2 -> 4.

Problem C: Rectalicious!

Given a set of adjacent rectangles with a common base, find the smallest set of rectangles that will cover them without covering anything else.

We signify areas belonging to each rectangle by the rectangle number, such as 1, 2, 3, and so on, and arrange them as in example below:

<p>If the rectangles are arranged as follows...</p> <pre> 44 445 2 445 1233445 1233445</pre>	<p>one cover using the least number of rectangles is:</p> <pre> 22 223 4 223 1111223 1111223</pre>
--	--

Input

There are multiple test cases. For each test case you will have:

Line 1: n , number of rectangles ($1 \leq n \leq 250000$)

Lines 2.. $n+1$: n pairs of integers, line $i+1$ contains the width and height of i^{th} rectangle ($1 \leq w, h \leq 10^9$)

Last line will have 0 to signify end of input.

Output

For each test case, you will have:

Line 1: The minimum number of rectangles needed to cover input rectangles.

Magnificent rectangular yellow birdie tells you that processing complexity using second power of N will result in much too lengthy units of time to compose certain case outputs.

Sample Input

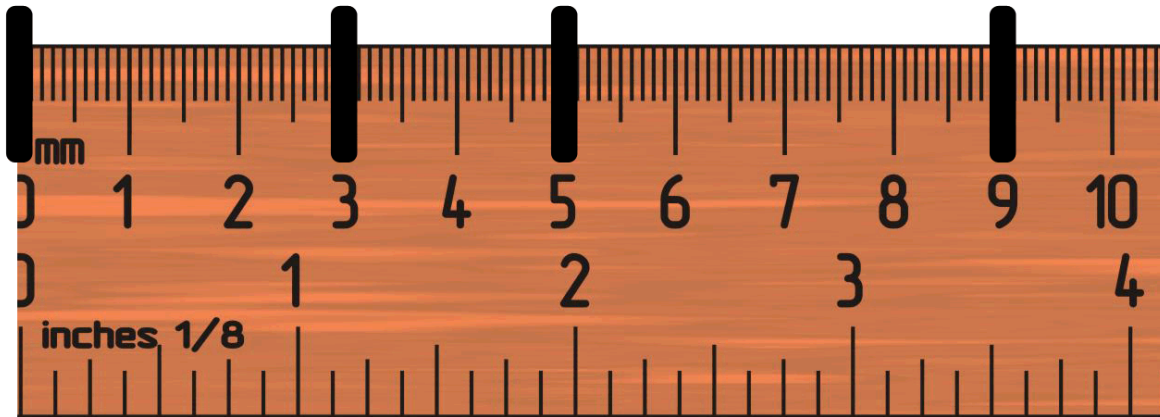
```
5
1 2
1 3
2 2
2 5
1 4
0
```

Sample Output

```
4
```

Problem D: Reconstruction Construction

You and your friend play a game. You are given a ruler and sliders that you can set on any particular ruler notch. For example, if the sliders are set in positions 0, 3, 5, and 6, you get the image below:



If you compute all possible distances between each pair of the sliders, you will get the numbers 3, 2, 4, 5, 6 and 9.

Your friend secretly puts on the sliders, computes all the distances, which he writes down on paper and then takes all the sliders off of the ruler. It is now your turn to reconstruct the original slider placement. Your friend says it may take too long time complexity-wise to figure this out, so use caution!

Input

There are multiple test cases. First line of each test case will have n , $1 < n \leq 500$, the number of sliders. Next line after that will list these distances, separated by spaces. Last line will contain the number 0 to end the input.

Output

Output a listing of slider placements, assuming the first one is always placed at position 0. If there is more than one such placement, output the first one lexicographically.

Sample Input

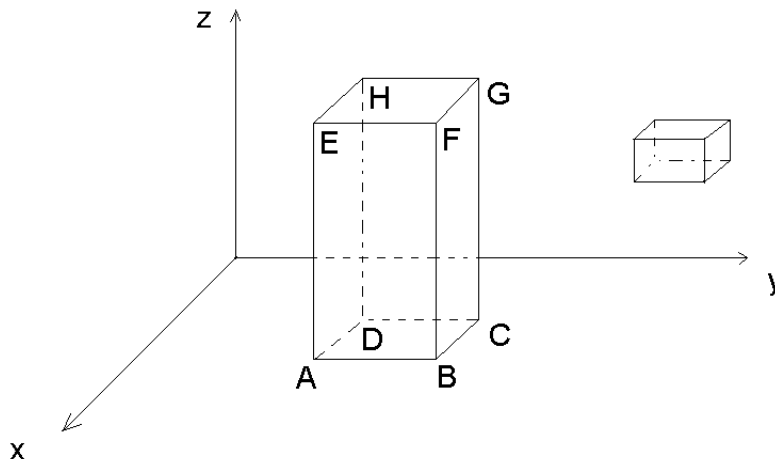
```
4
2 3 4 5 6 9
0
```

Sample Output

```
0 3 5 9
```

Problem E. Think Inside the Box

Dennis is bored and lonely during one of his advanced Calculus classes. One of his assignments features a 3-dimensional coordinate system with two 3-dimensional rectangular parallelepipeds. Dennis calls them 'boxes' to save on typing space. To make things more interesting for himself and to avoid boredom, Dennis decides to answer the age old question – is it possible to fit one of the boxes inside the other. For example, in the graph below there are two boxes. The first box has dimensions of $2 \times 3 \times 8$, while the second box has dimensions of $1 \times 3 \times 1$. The second box fits inside the first one, but the first one does not fit inside the second one.



In case the boxes are of identical sizes, Dennis considers them to be able to fit within each other. Dennis does not consider rotating boxes to make them fit. So, for example, according to Dennis, boxes with dimensions $2 \times 3 \times 4$ and $4 \times 3 \times 2$ do not fit within the each other, even though they could be considered identical otherwise. After doing a few of these exercises Dennis is no longer bored. But he is still lonely. To cheer him up and to keep him company, will you help him solve the rest of the problems?

Input

First line will contain an integer n , signifying the number of test cases to follow. Each of the following n lines will contain a description of two boxes by listing a series of 16 points in (x, y, z) format. This makes for 48 numbers total per line. The first 3 numbers signify point A of box 1 with coordinates (x_A, y_A, z_A) , the next 3 numbers signify point B for box 1 with coordinates (x_B, y_B, z_B) , and so on. The first 8 points (24 numbers) belong to the first box, and the next 8 points belongs to the second box. You may assume that each of the box's sides will be parallel to the coordinate system axis. The order of the box's corners will always be given in order of points A, B, C, D, E, F, G, and H as in the diagram above.

Output

For each test case, output a line in the following form: "Case i: box X fits inside box Y.", where i is the test case number that starts with 1, and X and Y are either 1 or 2. If boxes are identical, output "Case i: boxes are identical." If neither box fits inside the other, output "Case i: neither box fits inside the other." You are also required to separate test cases with a blank line.

Sample Input

```
2
6 5 1 6 8 1 4 8 1 4 5 1 6 5 9 6 8 9 4 8 9 4 5 9 6 6 7 6 9 7 5 9 7 5 6
7 6 6 8 6 9 8 5 9 8 5 6 8
2 0 0 2 2 0 0 2 0 0 0 0 2 0 2 2 2 2 0 2 2 0 0 2 2 0 0 2 2 0 0 2 0 0 0
0 2 0 2 2 2 2 0 2 2 0 0 2
```

Sample Output

Case 1: box 2 fits inside box 1.

Case 2: boxes are identical.