Problem A
To Add or to Multiply
Problem ID: addmul

The Industrial Computer Processor Company offers very fast, special purpose processing units tailored to customer needs. Processors of the a-C-m family (such as the 1-C-2 and the 5-C-3) have an instruction set with only two different operations:

`A` add a
`M` multiply by m

The processor receives an integer, executes a sequence of `A` and `M` operations (the program) that modifies the input, and outputs the result. For example, the 1-C-2 processor executing the program `AAAM` with the input 2 yields the output 10 (the computation is $2 \to 3 \to 4 \to 5 \to 10$), while the 5-C-3 processor yields 51 with the same program and input ($2 \to 7 \to 12 \to 17 \to 51$).

You are an a-C-m programmer assigned to a top secret project. This means that you have not been told the precise computation your program should perform. But you are given particular values p, q, r, and s and the following conditions:

1. The input is guaranteed to be a number between p and q.
2. The output must be some number between r and s.

Given an a-C-m processor and the numbers p, q, r, and s, your job is to construct the shortest a-C-m program which, for every input x such that $p \leq x \leq q$, yields some output y such that $r \leq y \leq s$. If there is more than one program of minimum length, choose the one that come first lexicographically, treating each program as a string of `A`s and `M`s.

Input

The input contains several test cases. Each test case is given by a line with the six integers a, m, p, q, r, and s as described above ($1 \leq a, m, p, q, r, s \leq 10^9$, $p \leq q$ and $r \leq s$).

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

Output

For each test case, display its case number followed by the best program as described above. Display the word "`empty`" if the best program uses no operations. Display the word "`impossible`" if there is no program meeting the specifications.

Display the program as a sequence of space-separated strings, alternating between strings of the form "n`A`" and strings of the form "n`M`", where $n > 0$. Strings of the former type indicate n consecutive `A` operations, and strings of the latter type indicate n consecutive `M` operations.

Follow the format of the sample output.

Sample input	Output for the Sample Input
1 2 2 3 10 20 1 3 2 3 22 33 3 2 2 3 4 5 5 3 2 3 2 3 0 0 0 0 0 0	Case 1: 1A 2M Case 2: 1M 2A 1M Case 3: impossible Case 4: empty

Problem B
Affine Mess
Problem ID: affine

Tess L. Ation ran into a little problem last week when she demonstrated the beta version of her new drawing software. On the screen she had an elegant demonstration design that illustrated every feature of her program; it had taken her hours to produce it. She was just putting the finishing touches on it as a group of potential investors entered the room to see the demonstration.

The presentation went well. Near the end, Tess clicked on a control panel button and told her audience, "This is the 'snap to grid' control. It forces control points, such as vertices, to jump to the nearest grid point. Here, let me show you," and she placed three bright red dots on the screen. Each one appeared at the grid point nearest to where she clicked. ("Luckily all control points in my demo design were already at integer coordinates. But I will have to remember to delete these three red dots before I save my diagram," she thought to herself.) "Now I'll step into the next room and get out of your way so you can discuss the system among yourselves and get a closer look at the screen, but please don't touch anything, since I haven't saved that file yet."

A few minutes later, the group joined Tess. One of the visitors stepped up to Tess and said, "I hope you don't mind, but I wanted to try it myself. Don't worry, I just played with the x-scale and y-scale controls a little bit." The next person said, "Sorry if this is a problem, but I really wanted to get a feel for the speed of display, so I just played around with the translation tool." And a third person said, "I couldn't resist just one tiny test: I rotated the image just so I could see all of the vertices snap to the nearest grid points after the rotation."

The person who played with the rotation tool remembered going first, but the other two could not recall their order. The three remembered only a few details of the changes. The x- and y-scaling factors had been (possibly negative) non-zero integers; the center of scaling was the origin $(0, 0)$. The x- and y-translation amounts had been integers. Rotation had been specified by a point with integer coordinates (x, y) on the perimeter of a square of width 20 centered at the origin (hence, $-10 \leq x, y \leq 10$ and the absolute value of x or y or both was 10). The tool rotated the drawing around the origin such that the positive x-axis would pass through (x, y) afterwards. Snapping took place after this rotation (coordinates with a fractional part of 0.5 were rounded away from zero).

After they left, Tess looked at her design – it was completely changed! She had not yet implemented the "undo" feature, and she had not saved the diagram prior to giving the demonstration. However, the three identical red dots were still there (transformed to other integer grid locations, of course), and Tess could remember the integer coordinates where she had originally placed them. Obviously, someone else might have altered the drawing without saying anything to her, but she could write a program to see if it was possible to reconstruct the sequence of alterations. Can you too?

Input

The input contains several test cases. Each test case consists of six pairs of integers x_i and y_i ($-500 \leq x_i, y_i \leq 500$ for $1 \leq i \leq 6$), three pairs per input line. The first three pairs represent the distinct initial locations of the three red dots. The last three pairs represent the distinct final locations of the three dots. The indexing of the pairs in each group of three is not significant: for example, (x_1, y_1) could have been mapped to any of (x_4, y_4), (x_5, y_5) or (x_6, y_6).

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

Output

For each test case, display its case number followed by one of the following three messages:

- "`equivalent solutions`" to indicate that there are one or more valid transformations, and all of them have the same effect on the whole drawing (no matter what the whole drawing looks like).
- "`inconsistent solutions`" to indicate that there are several valid transformations, but in general not all of them map the entire drawing in the same way (some drawing is mapped differently by two valid transformations).
- "`no solution`" to indicate that neither of the first two cases occurs.

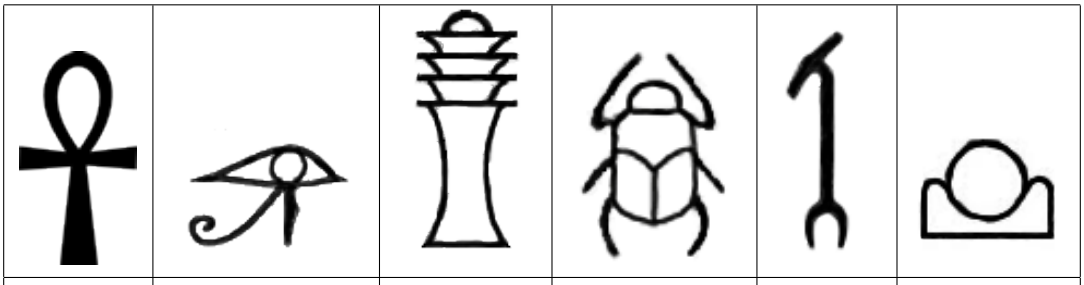
A valid transformation is a combination of rotation, translation and scaling (or rotation, scaling and translation) which satisfies the restrictions described above and maps the initial set of red dots to the final set (occupying all three final locations).

Follow the format of the sample output.

Sample input	Output for the Sample Input
3 0 4 0 1 4 -2 -4 -1 3 3 -4 0 1 1 1 2 1 1 2 2 2 3 2 1 0 2 0 3 0 3 3 1 1 2 2 1 0 2 0 3 0 3 2 1 1 2 2 2 3 0 6 1 2 2 3 0 6 1 2 0 0 0 0 0 0	Case 1: equivalent solutions Case 2: inconsistent solutions Case 3: no solution Case 4: inconsistent solutions Case 5: equivalent solutions

Problem C
Ancient Messages
Problem ID: ancient

In order to understand early civilizations, archaeologists often study texts written in ancient languages. One such language, used in Egypt more than 3000 years ago, is based on characters called hieroglyphs. Figure C.1 shows six hieroglyphs and their names. In this problem, you will write a program to recognize these six characters.



Ankh	Wedjat	Djed	Scarab	Was	Akeht

Figure C.1: Six hieroglyphs

Input

The input consists of several test cases, each of which describes an image containing one or more hieroglyphs chosen from among those shown in Figure C.1. The image is given in the form of a series of horizontal scan lines consisting of black pixels (represented by 1) and white pixels (represented by 0). In the input data, each scan line is encoded in hexadecimal notation. For example, the sequence of eight pixels `10011100` (one black pixel, followed by two white pixels, and so on) would be represented in hexadecimal notation as `9c`. Only digits and lowercase letters `a` through `f` are used in the hexadecimal encoding. The first line of each test case contains two integers, H and W. H ($0 < H \leq 200$) is the number of scan lines in the image. W ($0 < W \leq 50$) is the number of hexadecimal characters in each line. The next H lines contain the hexadecimal characters of the image, working from top to bottom. Input images conform to the following rules:

- The image contains only hieroglyphs shown in Figure C.1.
- Each image contains at least one valid hieroglyph.
- Each black pixel in the image is part of a valid hieroglyph.
- Each hieroglyph consists of a connected set of black pixels and each black pixel has at least one other black pixel on its top, bottom, left, or right side.
- The hieroglyphs do not touch and no hieroglyph is inside another hieroglyph.
- Two black pixels that touch diagonally will always have a common touching black pixel.
- The hieroglyphs may be distorted but each has a shape that is topologically equivalent to one of the symbols in Figure C.1[1].

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

[1] Two figures are topologically equivalent if each can be transformed into the other by stretching without tearing.

Output

For each test case, display its case number followed by a string containing one character for each hieroglyph recognized in the image, using the following code:

Ankh: `A`
Wedjat: `J`
Djed: `D`
Scarab: `S`
Was: `W`
Akhet: `K`

In each output string, print the codes in alphabetic order. Follow the format of the sample output.

The sample input contains descriptions of test cases shown in Figures C.2 and C.3. Due to space constraints not all of the sample input can be shown on this page.

Figure C.2: AKW

Figure C.3: AAAAA

Sample input	Output for the Sample Input
100 25 00000000000000000000000000 00000000000000000000000000 ...(50 lines omitted)... 00001fe0000000000007c0000 00003fe0000000000007c0000 ...(44 lines omitted)... 00000000000000000000000000 00000000000000000000000000 150 38 00000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000 ...(75 lines omitted)... 0000000003fffffffffffffff00000000000 0000000003fffffffffffffff00000000000 ...(69 lines omitted)... 00000000000000000000000000000000000000000000 00000000000000000000000000000000000000000000 0 0	Case 1: AKW Case 2: AAAAA

Problem D
Chips Challenge
Problem ID: chips

A prominent microprocessor company has enlisted your help to lay out some interchangeable components (widgets) on some of their computer chips. Each chip's design is an $N \times N$ square of slots. One slot can hold a single component, and you are to try to fit in as many widgets as possible.

Modern processor designs are complex, of course. You unfortunately have several restrictions:

- Some of the slots are disabled.
- Some of the slots are already occupied by other components and cannot be used for widgets.
- There are sibling memory buses connected to the horizontal and vertical edges of the chip and their bandwidth loads need to match. As such, there must be exactly as many components in the first row as in the first column, exactly as many in the second row as in the second column, and so on. Component counts include both the components already specified on the chip and the added widgets.
- Similarly, the power supply is connected at the end of each row and column. To avoid hot spots, any given row or column must have no more than A/B of the total components on the chip for a given A and B.

A specification for a chip is N lines of N characters, where '`.`' indicates an open slot, '`/`' indicates a disabled slot, and '`C`' indicates a slot already occupied by a component. For example:

```
CC/..
././/
..C.C
/.C..
/./C/
```

If no more than $3/10$ of the components may be in any one row or column, the maximum number of widgets that can be added to this 5×5 chip is 7. A possible arrangement is below, where '`W`' indicates a widget added in an open slot.

```
CC/W.
W/W//
W.C.C
/.CWW
/W/C/
```

Input

The input consists of several test cases. Each case starts with a line containing three integers: The size of the chip N ($1 \leq N \leq 40$), and A and B ($1 \leq B \leq 1000$, $0 \leq A \leq B$) as described above. Each of the following N lines contains N characters describing the slots, one of '`.`', '`/`' or '`C`', as described above.

The last test case is followed by a line containing three zeros.

Output

For each test case, display a single line beginning with the case number. If there is a solution, display the maximum number of widgets that can be added to the chip. Display "`impossible`" if there is no solution.

Follow the format of the sample output.

Sample input	Output for the Sample Input
2 1 1 /. // 2 50 100 /. C/ 2 100 100 ./ C. 5 3 10 CC/.. ./.// ..C.C /.C.. /./C/ 5 2 10 CC/.. ./.// ..C.C /.C.. /./C/ 0 0 0	Case 1: 0 Case 2: 1 Case 3: impossible Case 4: 7 Case 5: impossible

Problem E
Coffee Central
Problem ID: coffee

Is it just a fad or is it here to stay? You're not sure, but the steadily increasing number of coffee shops that are opening in your hometown has certainly become quite a draw. Apparently, people have become so addicted to coffee that apartments that are close to many coffee shops will actually fetch higher rents.

This has come to the attention of a local real-estate company. They are interested in identifying the most valuable locations in the city in terms of their proximity to large numbers of coffee shops. They have given you a map of the city, marked with the locations of coffee shops. Assuming that the average person is willing to walk only a fixed number of blocks for their morning coffee, you have to find the location from which one can reach the largest number of coffee shops. As you are probably aware, your hometown is built on a square grid layout, with blocks aligned on north-south and east-west axes. Since you have to walk along streets, the distance between intersections (a, b) and (c, d) is $|a - c| + |b - d|$.

Input

The input contains several test cases. Each test case describes a city. The first line of each test case contains four integers dx, dy, n, and q. These are the dimensions of the city grid $dx \times dy$ ($1 \leq dx, dy \leq 1000$), the number of coffee shops n ($0 \leq n \leq 5 \cdot 10^5$), and the number of queries q ($1 \leq q \leq 20$). Each of the next n lines contains two integers x_i and y_i ($1 \leq x_i \leq dx$, $1 \leq y_i \leq dy$); these specify the location of the i^{th} coffee shop. There will be at most one coffee shop per intersection. Each of the next q lines contains a single integer m ($0 \leq m \leq 10^6$), the maximal distance that a person is willing to walk for a cup of coffee.

The last test case is followed by a line containing four zeros.

Output

For each test case in the input, display its case number. Then display one line per query in the test case. Each line displays the maximum number of coffee shops reachable for the given query distance m followed by the optimal location. For example, the sample output shows that 3 coffee shops are within query distance 1 of the optimal location $(3, 4)$, 4 shops are within query distance 2 of optimal location $(2, 2)$, and 5 shops are within query distance 4 of optimal location $(3, 1)$. If there are multiple optimal locations, pick the location that is furthest south (minimal positive integer y-coordinate). If there is still a tie, pick the location furthest west (minimal positive integer x-coordinate).

Follow the format of the sample output.

Sample input	Output for the Sample Input
4 4 5 3 1 1 1 2 3 3 4 4 2 4 1 2 4 0 0 0 0	Case 1: 3 (3,4) 4 (2,2) 5 (3,1)

Problem F
Machine Works
Problem ID: works

You are the director of Arbitrarily Complex Machines (ACM for short), a company producing advanced machinery using even more advanced machinery. The old production machinery has broken down, so you need to buy new production machines for the company. Your goal is to make as much money as possible during the restructuring period. During this period you will be able to buy and sell machines and operate them for profit while ACM owns them. Due to space restrictions, ACM can own at most one machine at a time.

During the restructuring period, there will be several machines for sale. Being an expert in the advanced machines market, you already know the price P_i and the availability day D_i for each machines M_i. Note that if you do not buy machine M_i on day D_i, then somebody else will buy it and it will not be available later. Needless to say, you cannot buy a machine if ACM has less money than the price of the machine.

If you buy a machine M_i on day D_i, then ACM can operate it starting on day $D_i + 1$. Each day that the machine operates, it produces a profit of G_i dollars for the company.

You may decide to sell a machine to reclaim a part of its purchase price any day after you've bought it. Each machine has a resale price R_i for which it may be resold to the market. You cannot operate a machine on the day that you sell it, but you may sell a machine and use the proceeds to buy a new machine on the same day.

Once the restructuring period ends, ACM will sell any machine that it still owns. Your task is to maximize the amount of money that ACM makes during the restructuring.

Input

The input consists of several test cases. Each test case starts with a line containing three positive integers N, C, and D. N is the number of machines for sale ($N \leq 10^5$), C is the number of dollars with which the company begins the restructuring ($C \leq 10^9$), and D is the number of days that the restructuring lasts ($D \leq 10^9$).

Each of the next N lines describes a single machine for sale. Each line contains four integers D_i, P_i, R_i and G_i, denoting (respectively) the day on which the machine is for sale, the dollar price for which it may be bought, the dollar price for which it may be resold and the daily profit generated by operating the machine. These numbers satisfy $1 \leq D_i \leq D$, $1 \leq R_i < P_i \leq 10^9$ and $1 \leq G_i \leq 10^9$.

The last test case is followed by a line containing three zeros.

Output

For each test case, display its case number followed by the largest number of dollars that ACM can have at the end of day $D + 1$.

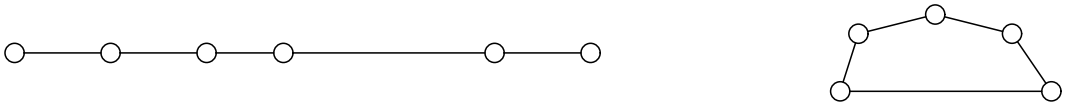
Follow the format of the sample output.

Sample input	Output for the Sample Input
6 10 20 6 12 1 3 1 9 1 2 3 2 1 2 8 20 5 4 4 11 7 4 2 10 9 1 0 0 0	Case 1: 44

Problem G
Magic Sticks
Problem ID: magicsticks

Magic was accepted by all ancient peoples as a technique to compel the help of divine powers. In a well-known story, one group of sorcerers threw their walking sticks on the floor where they magically appeared to turn into live serpents. In opposition, another person threw his stick on the floor, where it turned into a serpent which then consumed the sorcerers' serpents!

The only magic required for this problem is its solution. You are given a magic stick that has several straight segments, with joints between the segments that allow the stick to be folded. Depending on the segment lengths and how they are folded, the segments of the stick can be arranged to produce a number of polygons. You are to determine the maximum area that could be enclosed by the polygons formed by folding the stick, using each segment in at most one polygon. Segments can touch only at their endpoints. For example, the stick shown below on the left has five segments and four joints. It can be folded to produce a polygon as shown on the right.



Input

The input contains several test cases. Each test case describes a magic stick. The first line in each test case contains an integer n ($1 \leq n \leq 500$) which indicates the number of the segments in the magic stick. The next line contains n integers $S_1, S_2, \ldots, S_n$ ($1 \leq S_i \leq 1000$) which indicate the lengths of the segments in the order they appear in the stick.

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

Output

For each case, display its case number followed by the maximum total enclosed area that can be obtained by folding the magic stick at the given points. Answers within an absolute or relative error of 10^{-4} will be accepted.

Follow the format of the sample output.

Sample input	Output for the Sample Input
4 1 2 3 4 8 3 4 5 33 3 4 3 5 0	Case 1: 4.898979 Case 2: 19.311

This page is intentionally left blank.

Problem H
Mining Your Own Business
Problem ID: mining

John Digger is the owner of a large illudium phosdex mine. The mine is made up of a series of tunnels that meet at various large junctions. Unlike some owners, Digger actually cares about the welfare of his workers and has a concern about the layout of the mine. Specifically, he worries that there may a junction which, in case of collapse, will cut off workers in one section of the mine from other workers (illudium phosdex, as you know, is highly unstable). To counter this, he wants to install special escape shafts from the junctions to the surface. He could install one escape shaft at each junction, but Digger doesn't care about his workers *that* much. Instead, he wants to install the minimum number of escape shafts so that if any of the junctions collapses, all the workers who survive the junction collapse will have a path to the surface.

Write a program to calculate the minimum number of escape shafts and the total number of ways in which this minimum number of escape shafts can be installed.

Input

The input consists of several test cases. The first line of each case contains a positive integer N ($N \leq 5 \cdot 10^4$) indicating the number of mine tunnels. Following this are N lines each containing two distinct integers s and t, where s and t are junction numbers. Junctions are numbered consecutively starting at 1. Each pair of junctions is joined by at most a single tunnel. Each set of mine tunnels forms one connected unit (that is, you can get from any one junction to any other).

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 minimum number of escape shafts needed for the system of mine tunnels and the total number of ways these escape shafts can be installed. You may assume that the result fits in a signed 64-bit integer.

Follow the format of the sample output.

Sample input	Output for the Sample Input
9 1 3 4 1 3 5 1 2 2 6 1 5 6 3 1 6 3 2 6 1 2 1 3 2 4 2 5 3 6 3 7 0	Case 1: 2 4 Case 2: 4 1

Problem I
Mummy Madness
Problem ID: mummy

During an excursion to the desert at the 2011 ACM-ICPC World Finals, you come across an old Egyptian tomb. Unfortunately, opening the tomb turns out to be a bad idea: all of a sudden, what was just a few moments ago an empty desert has now become a desert crawling with grumpy mummies (you would be grumpy too if you were suddenly awakened after a few thousand years of peaceful sleep).[2]

Faced with this murderous mass of mad mummies, your only chance is to run for it and try to escape before they catch you. The question is: how long will it take before a mummy catches you, assuming neither you nor the mummies ever get tired?

We model the desert as a grid of squares. You and the mummies take turns making moves on the grid. You make the first move. In your turns, you can move to any of the eight squares adjacent to your current location, or you can choose to stand still. In the mummies' turns, each mummy simply moves to the adjacent square that brings it closest to you (measured by Euclidean distance, assuming that you and all the mummies stand in the centers of their respective squares). It is possible for two mummies to occupy the same square.

A time step consists of your move followed by the mummies' moves. A mummy catches you if it moves to the square where you are located, or if you move to the square occupied by the mummy. Of course, you try to avoid being caught for as long as possible. After how many time steps will you be caught?

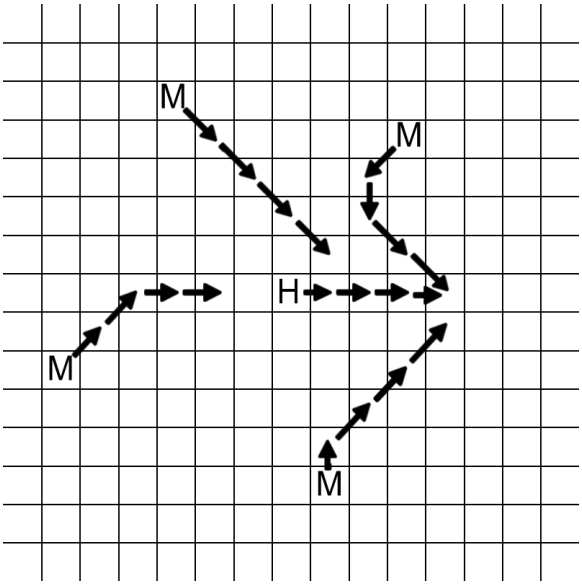


Figure I.1: A mummy chase

The figure illustrates what might happen if you are being chased by four mummies. The square labeled H is your initial position, and the squares labeled M are the initial positions of mummies. After four time steps, you are caught by the mummy whose initial position was $(3, 4)$ with respect to your initial position.

[2]Fortunately, after solving this problem, you woke up safe and sound in a hotel room in Florida. The enraged mummies had just been a dream. Or had they?

Input

The input consists of several test cases. Each test case begins with an integer n ($0 \leq n \leq 10^5$) giving the number of mummies in the desert. The following n lines each contain two integers x and y, indicating that there is initially a mummy at coordinates (x, y) of the desert, where x and y are both bounded by 10^6 in absolute value. Your starting position is $(0, 0)$, and no mummy starts at this position.

The last test case is followed by a line containing the number -1.

Output

For each test case, display its test case number followed by the maximum number of time steps until you are caught (measured as the total number of turns that you get), or the word "`never`" if you can avoid capture indefinitely.

Follow the format of the sample output.

Sample input	Output for the Sample Input
4 -3 5 3 4 -6 -2 1 -5 1 0 -1 -1	Case 1: 4 Case 2: never

Problem J
Pyramids
Problem ID: pyramids

It is not too hard to build a pyramid if you have a lot of identical cubes. On a flat foundation you lay, say, 10×10 cubes in a square. Centered on top of that square you lay a 9×9 square of cubes. Continuing this way you end up with a single cube, which is the top of the pyramid. The height of such a pyramid equals the length of its base, which in this case is 10. We call this a *high* pyramid.

If you think that a high pyramid is too steep, you can proceed as follows. On the 10×10 base square, lay an 8×8 square, then a 6×6 square, and so on, ending with a 2×2 top square (if you start with a base of odd length, you end up with a single cube on top, of course). The height of this pyramid is about half the length of its base. We call this a *low* pyramid.

Once upon a time (quite a long time ago, actually) there was a pharaoh who inherited a large number of stone cubes from his father. He ordered his architect to use all of these cubes to build a pyramid, not leaving a single one unused. The architect kindly explained that not every number of cubes can form a pyramid. With 10 cubes you can build a low pyramid with base 3. With 5 cubes you can build a high pyramid of base 2. But no pyramid can be built using exactly 7 cubes.

The pharaoh was not amused, but after some thinking he came up with new restrictions.

1. All cubes must be used.
2. You may build more than one pyramid, but you must build as few pyramids as possible.
3. All pyramids must be different.
4. Each pyramid must have a height of at least 2.
5. Satisfying the above, the largest of the pyramids must be as large as possible (i.e., containing the most cubes).
6. Satisfying the above, the next-to-largest pyramid must be as large as possible.
7. And so on...

Drawing figures and pictures in the sand, it took the architect quite some time to come up with the best solution.

Write a program that determines how to meet the restrictions of the pharaoh, given the number of cubes.

Input

The input consists of several test cases, each one on a single line. A test case is an integer c, where $1 \leq c \leq 10^6$, giving the number of cubes available.

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 pyramids to be built. The pyramids should be ordered with the largest first. Pyramids are specified by the length of their base followed by an `L` for low pyramids or an `H` for high pyramids. If two differenct pyramids have the same number of cubes, list the high pyramid first. Print "`impossible`" if it is not possible to meet the requirements of the pharaoh.

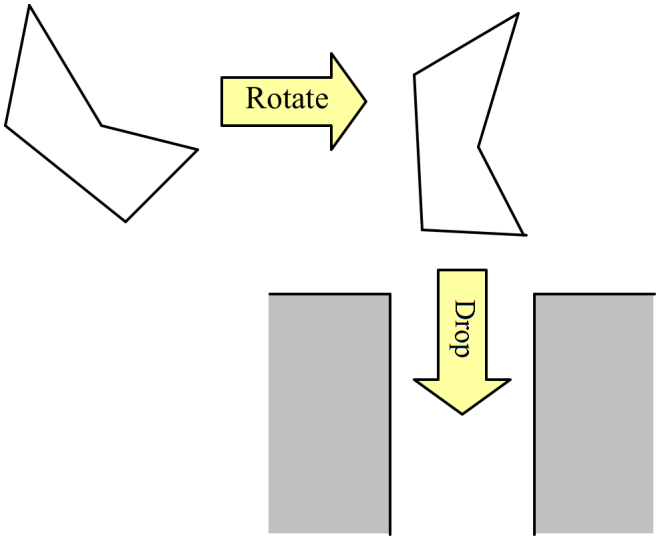
Follow the format of the sample output.

Sample input	Output for the Sample Input
29 28 0	Case 1: 3H 3L 2H Case 2: impossible

Problem K
Trash Removal
Problem ID: trash

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	Output for the Sample Input
3 0 0 3 0 0 4 4 0 10 10 0 20 10 10 20 0	Case 1: 2.40 Case 2: 14.15