Problem A: Tough Water Level

`water.c | water.C | water.java | water.p`

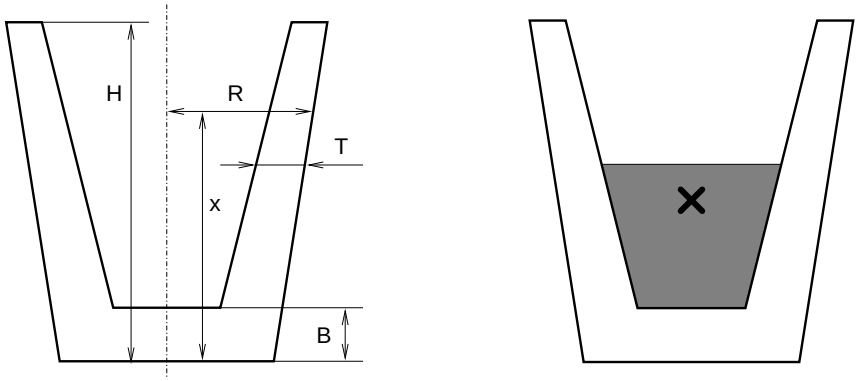
Czech Technical University has the word "technical" in its name. Beside others, this means that lectures in physics are important here. Do you still remember some of the basic physical principles?

For example, imagine a simple glass of water. Or, we will rather call it a *cup* to avoid ambiguity of this word. So, imagine a simple cup (made of glass) that is partially filled with water. You might have noticed that the stability of such a cup depends on the amount of water inside. If you brush against a full cup, it is relatively easy to knock it down and spill its contents. If the cup is empty, there is fortunately nothing to be spilled, but other than that, the situation does not improve much — it is still easy to knock the cup down with only a little force. The best stability is usually achieved with a "half-full" cup.

In this problem, your task is to determine the water level that makes a cup as much stable as possible. For the purpose of this problem, we will make a simple assumption that the "stability" of a cup is higher, if its center of mass (sometimes also called the center of gravity) is lower (closer to the bottom).

The *center of mass* can be informally defined as follows: Imagine that glass and water consist of a very large number of very small particles. Then the center of mass is an average of the position of all these particles. The average is weighted by particle masses. Since the density of glass is approximately 2 500 $kg.m^{-3}$ and the density of water only 1 000 $kg.m^{-3}$, we will suppose that the mass of a glass particle is 2.5 times higher than the mass of a water particle of the same size.

All cups considered in this problem will have an exact rotary shape. But their radius may vary with the height — some cups are wider at the top, others are wider at the bottom. Also, the thickness of the glass may not be constant.



The left figure shows a typical cup considered in this problem. It can be fully described by its height (H), thickness of the bottom (B), and two functions R and T. Both of these functions take a current height as their argument and they give the outer radius (R) and glass thickness (T) in the appropriate height. Please note that the thickness is always measured strictly horizontally and may therefore not reflect the "real thickness" of the glass in its usual meaning.

Input Specification

The input contains several cup descriptions. Each description consists of three lines. The first line contains two numbers: H (cup height) and B (bottom thickness), $0 < B < H \leq 100$. The second line contains an expression $R(x)$ (radius), the third line an expression $T(x)$ (glass thickness). All data are given in centimeters. The last description is followed by a line with two zeros.

The expressions will contain only digits ("`0`" through "`9`"), decimal points ("`.`"), four basic operators ("`+`", "`-`", "`*`", and "`/`"), parentheses ("`(`" and "`)`"), and the lowercase letters "`x`" denoting the input variable (height measured from the cup bottom).

$$\forall x, 0 \leq x \leq H, \text{ the radius will satisfy: } 0.1 < R(x) \leq 100$$

$$\forall x, B \leq x \leq H, \text{ the thickness will satisfy: } 0.1 \leq T(x) < R(x)$$

Arithmetical operators have their usual meaning and priorities, i.e., multiplication and division have a higher priority than addition and subtraction.

Output Specification

For each cup, output the sentence "`Pour` L `litres /` W `cm of water.`", where L is the amount of water that must be poured into the cup to reach the maximal stability (in litres). W is the water level (in centimeters) measured from the bottom of the cup. Both numbers must be given with exactly three decimal digits.

Sample Input

```
9 1
3+x/6
17/8-x/8
10 1
(x+10)
10/(x+10)
0 0
```

Output for Sample Input

```
Pour 0.030 litres / 3.365 cm of water.
Pour 0.878 litres / 2.193 cm of water.
```

Problem B: Robotic Sort

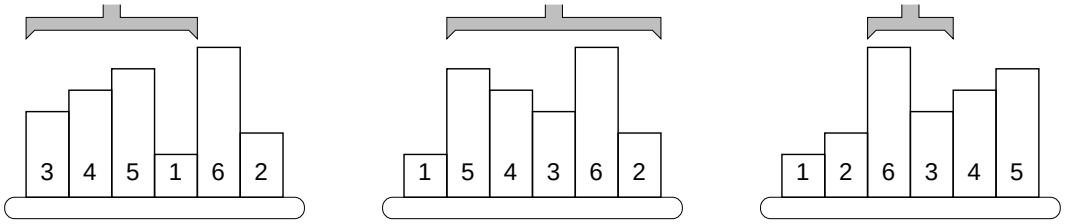
`sort.c | sort.C | sort.java | sort.p`

Somewhere deep in the Czech Technical University buildings, there are laboratories for examining mechanical and electrical properties of various materials. In one of yesterday's presentations, you have seen how was one of the laboratories changed into a new multimedia lab. But there are still others, serving to their original purposes.

In this task, you are to write software for a robot that handles samples in such a laboratory. Imagine there are material samples lined up on a running belt. The samples have different heights, which may cause troubles to the next processing unit. To eliminate such troubles, we need to sort the samples by their height into the ascending order.

Reordering is done by a mechanical robot arm, which is able to pick up any number of consecutive samples and turn them round, such that their mutual order is reversed. In other words, one robot operation can reverse the order of samples on positions between A and B.

A possible way to sort the samples is to find the position of the smallest one (P_1) and reverse the order between positions 1 and P_1, which causes the smallest sample to become first. Then we find the second one on position P_2 and reverse the order between 2 and P_2. Then the third sample is located etc.



The picture shows a simple example of 6 samples. The smallest one is on the 4th position, therefore, the robot arm reverses the first 4 samples. The second smallest sample is the last one, so the next robot operation will reverse the order of five samples on positions 2–6. The third step will be to reverse the samples 3–4, etc.

Your task is to find the correct sequence of reversal operations that will sort the samples using the above algorithm. If there are more samples with the same height, their mutual order must be preserved: the one that was given first in the initial order must be placed before the others in the final order too.

Input Specification

The input consists of several scenarios. Each scenario is described by two lines. The first line contains one integer number N, the number of samples, $1 \leq N \leq 100\,000$. The second line lists exactly N space-separated positive integers, they specify the heights of individual samples and their initial order.

The last scenario is followed by a line containing zero.

Output Specification

For each scenario, output one line with exactly N integers $P_1, P_2, \ldots P_N$, separated by a space. Each P_i must be an integer ($1 \leq P_i \leq N$) giving the position of the i-th sample just before the i-th reversal operation.

Note that if a sample is already on its correct position P_i, you should output the number P_i anyway, indicating that the "interval between P_i and P_i" (a single sample) should be reversed.

Sample Input

```
6
3 4 5 1 6 2
4
3 3 2 1
0
```

Output for Sample Input

```
4 6 4 5 6 6
4 2 4 4
```

Problem C: Reaux! Sham! Beaux!

`roshambo.c | roshambo.C | roshambo.java | roshambo.p`

Roshambo — this simple game is known all around the world. In German, it is called "Schnick, Schnack, Schnuck", in Japanese "Janken", in Spanish "Cachipún", in Polish "Papier, kamień, nożyce". The Czechs call it "Kámen, nůžky, papír".

Whatever is the name of the game, its principles remain the same. Two players simultaneously form their hand into one of three possible shapes (symbols): Rock (closed fist), Paper (open hand), or Scissors (two fingers extended). If both of them show the same symbol, it is a tie and no points are given. Otherwise, one of the symbols wins: Rock blunts Scissors, Scissors cut Paper, and Paper covers Rock.

Czech Technical University students also know the game very well and use it to resolve small disputes. Imagine, for example, two students living together in one room. Yesterday evening, there was a small celebration, and in the morning, no one wants to go to the lectures. They agreed that one person would be enough to take notices for both, but who will be the poor one? Roshambo is a very effective way to decide.

Did you know there are even the World Series of Roshambo? Our organizing team would like to host the World Championships in 2009.[1] Your task is to help us in developing a Roshambo scoring system and write a program that evaluates one game between two players.

Since the participants will come from different countries, the system must accept input in various languages. The following table shows names of three Roshambo symbols. Note that in some languages, there may be two different words for the same symbol.

Language	Code	Rock	Scissors	Paper
Czech	cs	Kamen	Nuzky	Papir
English	en	Rock	Scissors	Paper
French	fr	Pierre	Ciseaux	Feuille
German	de	Stein	Schere	Papier
Hungarian	hu	Ko | Koe	Ollo | Olloo	Papir
Italian	it	Sasso | Roccia	Forbice	Carta | Rete
Japanese	jp	Guu	Choki	Paa
Polish	pl	Kamien	Nozyce	Papier
Spanish	es	Piedra	Tijera	Papel

Input Specification

The input contains several games. Each game starts with two lines describing players. Each of these two lines contains two lowercase letters specifying the language used by the player (see the language code in the table above), one space, and a player name. The name will consist from at most twenty upper- or lower-case letters.

[1] As you know, the Regional Contest will move to Poland for the next year and we want some other international competition at the Czech Technical University.

After the players description, there are at most 100 lines containing individual rounds. Each round is described by two words separated with one space. The words name the symbol shown by the first and second player, respectively. All symbols are named in the mother tongue of the concerned player. All allowed words are shown in the table above, the first letter will be always in uppercase, all other letters in lowercase.

The last round is followed by a line containing one single dash character ("`-`") and then the next game begins. The only exception is the last game in the input, which is terminated by a dot ("`.`") instead of the dash.

Output Specification

For each game, print five lines of output. The first line should contain the string "`Game #`G`:`", where G is the number of the game, starting with one.

The second line will contain the first player name followed by a colon ("`:`"), one space and the number of rounds won by that player. The number should be followed by one space and the word "`points`". Use the singular form "`point`" if (and only if) the number of points of the player equals one.

The third line has the same format and shows the second player's name and points.

The fourth line displays the outcome of the game. It must contain the word "`WINNER`" followed by a colon, space and the name of the player who gained more points. If both players have the same number of points, the fourth line will contain words "`TIED GAME`" instead.

The fifth line is left empty to visually separate individual games.

Sample Input

```
cs Pepik
en Johnny
Nuzky Scissors
Papir Rock
Papir Scissors
-
de Gertruda
cs Lenka
Stein Papir
Schere Kamen
.
```

Output for Sample Input

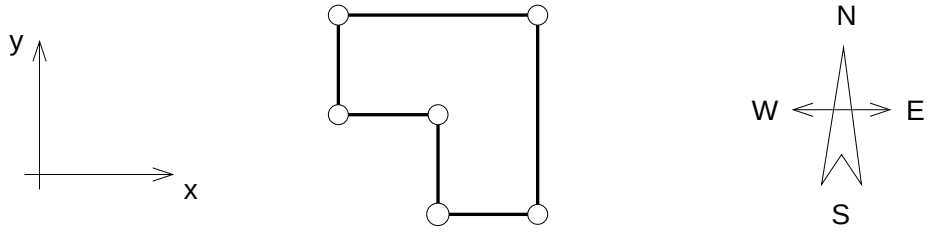
```
Game #1:
Pepik: 1 point
Johnny: 1 point
TIED GAME

Game #2:
Gertruda: 0 points
Lenka: 2 points
WINNER: Lenka
```

Acknowledgements to Wikipedia, the free encyclopedia, for providing background information and symbol names in various languages. If your own language is missing, it may be because it has no article on Roshambo in Wikipedia.

Problem D: Rectangular Polygons

`polygon.c | polygon.C | polygon.java | polygon.p`

In this problem, we will help the Faculty of Civil Engineering. They need a software to analyze ground plans of buildings. Specifically, your task is to detect outlines of a building when all of its corners are given.



You may assume that each building is a rectangular polygon with each of its sides being parallel either with X or Y axis. Therefore, each of its vertex angles is exactly either 90 or 270 degrees.

Input Specification

The input contains several buildings. The description of each building starts with a single positive integer N, the number of corners (polygon vertices), $1 \leq N \leq 1000$. Then there are N pairs of integer numbers X_i, Y_i giving coordinates of individual corners, $|X_i|, |Y_i| \leq 10\,000$.

You may assume that all corners are listed and no two of them have the same coordinates. The polygon does always exist, it is closed, its sides do not intersect or touch (except neighboring sides, of course), and it contains no "holes" inside. In other words, the outline is formed by one closed line. The order of corners in the input file may be arbitrary.

There is an empty line after each building, then the next one is described. After the last building, there is a single zero that signals the end of input.

Output Specification

For each building, output one line containing N characters without any whitespace between them. The characters should be uppercase letters that specify directions of individual walls (sides) when the building outline is followed. "`N`" stands for North (the positive direction of the Y axis), "`E`" for East (the positive direction of the X axis), "`W`" for West, and "`S`" for South. The "walk" should start in the vertex that has been given first in the input and always proceed in the *clockwise* direction.

Sample Input

```
4
0 0
2 2
0 2
2 0

6
1 1
2 2
0 1
1 0
0 2
2 0

0
```

Output for Sample Input

```
NESW
WNESWN
```

The second sample input corresponds to the picture.

Problem E: Weird Numbers

`numbers.c | numbers.C | numbers.java | numbers.p`

Binary numbers form the principal basis of computer science. Most of you have heard of other systems, such as ternary, octal, or hexadecimal. You probably know how to use these systems and how to convert numbers between them. But did you know that the system base (radix) could also be negative? One assistant professor at the Czech Technical University has recently met *negabinary* numbers and other systems with a negative base. Will you help him to convert numbers to and from these systems?

A number N written in the system with a positive base R will always appear as a string of digits between 0 and $R-1$, inclusive. A digit at the position P (positions are counted from right to left and starting with zero) represents a value of R^P. This means the value of the digit is multiplied by R^P and values of all positions are summed together. For example, if we use the octal system (radix $R=8$), a number written as 17024 has the following value:

$$1.8^4 + 7.8^3 + 0.8^2 + 2.8^1 + 4.8^0 = 1.4096 + 7.512 + 2.8 + 4.1 = 7700$$

With a negative radix $-R$, the principle remains the same: each digit will have a value of $(-R)^P$. For example, a negaoctal (radix $R=-8$) number 17024 counts as:

$$1.(-8)^4 + 7.(-8)^3 + 0.(-8)^2 + 2.(-8)^1 + 4.(-8)^0 = 1.4096 - 7.512 - 2.8 + 4.1 = 500$$

One big advantage of systems with a negative base is that we do not need a minus sign to express negative numbers. A couple of examples for the negabinary system ($R=-2$):

decimal	negabinary	decimal	negabinary	decimal	negabinary
-10	1010	-3	1101	4	100
-9	1011	-2	10	5	101
-8	1000	-1	11	6	11010
-7	1001	0	0	7	11011
-6	1110	1	1	8	11000
-5	1111	2	110	9	11001
-4	1100	3	111	10	11110

You may notice that the negabinary representation of any integer number is unique, if no "leading zeros" are allowed. The only number that can start with the digit "0", is the zero itself.

Input Specification

The input will contain several conversions, each of them specified on one line. A conversion from the decimal system to some negative-base system will start with a lowercase word "`to`" followed by a minus sign (with no space before it), the requested base (radix) R, one space, and a decimal number N.

A conversion to the decimal system will start with a lowercase word "`from`", followed by a minus sign, radix R, one space, and a number written in the system with a base of $-R$.

The input will be terminated by a line containing a lowercase word "`end`". All numbers will satisfy the following conditions: $2 \leq R \leq 10$, $-1\,000\,000 \leq N \leq 1\,000\,000$ (decimal).

Output Specification

For each conversion, print one number on a separate line. If the input used a decimal format, output the same number written in the system with a base $-R$. If the input contained such a number, output its decimal value.

Both input and output numbers must not contain any leading zeros. The minus sign "`-`" may only be present with negative numbers written in the decimal system. Any non-negative number or a number written in a negative-base system must not start with it.

Sample Input

```
to-2 10
from-2 1010
to-10 10
to-10 -10
from-10 10
end
```

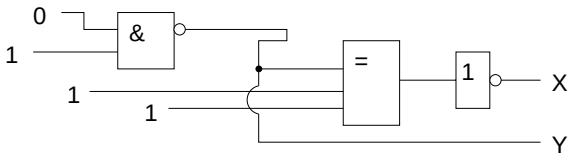
Output for Sample Input

```
11110
-10
190
10
-10
```

Problem F: Gates of Logic

`logic.c | logic.C | logic.java | logic.p`

The Department of Computer Science and Engineering runs courses dealing not only with algorithms but also with computer hardware. One such introductory course explains basic principles of integrated circuits ("chips"), binary logic, boolean algebra, etc. As you may know, the very basic units of logical circuits are called *gates*. A gate is an element performing one simple logical operation. It can be connected to other gates using *lines*.



Logical circuits may be drawn as pictures with the gates represented as squares with inputs on the left and outputs on the right. In each square, there is a symbol that determines the gate type: Number **1** denotes an OR gate (its outputs are 0 if and only if there is no input with the value of 1), **&** is an AND gate (outputs are 1 if and only if there is no 0 input), and **=** is a XOR gate (outputs are 1 if and only if there is an odd number inputs that have the value of 1).

Your task is to scan such a "picture" and compute values of all named circuit outputs. The lines may split and join again but you may assume that each "value consumer" (input port of a gate or a named output) will be connected to exactly one "value source" (output port of a gate or an input value). There will be no feedback loops, i.e., there exists no cycle that would lead through the same gate twice.

Input Specification

The input contains several pictures. Each picture consists of at least one and at most 200 rows composed of the following characters:

- **Space** (" "). Empty space in the picture. Spaces are used to indent other characters to appropriate locations, because the exact position of characters is often important. Trailing spaces at the end of input rows may be present but may also be left out.
- **Dash** ("`-`"). Horizontal line. It connects characters on its left and right together, those characters will always exist and be able to "accept" the connection.
- **Pipe** ("`|`"). Vertical line, connects characters that are directly above and below. Like with the horizontal line, those characters will always accept the connection.
- **Plus sign** ("`+`"). Line connection or a bend. Connects characters on all four sides. All characters that are able to accept the connection are considered connected (there will always be at least two). However, there may be sides that contain a non-empty character that is not connected. For example, if a dash is present on a position directly below the plus sign, they are not considered connected.
- **Lowercase letter x** ("`x`"). Crossing of two lines without a connection. All four neighboring characters will accept the connection. The character above is connected to the one below and the character to the left with the one on the right, but there is no mutual connection between these two pairs.

- **Equal sign** ("`=`"). Represents an input or output port. It always connects characters on its left and right, at least one of these characters is the port. If there is a port on the left, it may only be a value source. If there is a port on the right, it may only be a value consumer.
- **Lowercase letter o** ("`o`"). Negation. There will always be a gate on the left and a port on the right of this character. It makes the particular gate output negated.
- **Hash mark** ("`#`"). Gate, which has always a rectangular shape with two vertical and two horizontal sides. The left vertical side may be connected to input ports, the right side to output ports (possibly negated). No two gates will touch each other's side, which means that any two vertically or horizontally neighboring hash marks are always parts of the same gate.

  The rectangle size will always be at least 3 characters in both directions, which means there is at least one character inside. All inner characters are empty (spaces), with exactly one exception. That single non-empty character denotes the gate type (note that it may have different meaning than outside the gate area) and will be a digit "one" ("`1`"), ampersand ("`&`"), or an equal sign ("`=`").
- **Binary digit** ("`0`" and "`1`"). Input value of the circuit. It is connected to the character on its right, which is always an equal sign.
- **Uppercase letter** ("`A`" through "`Z`"). Named output of the circuit. It accepts connection from its left, which is always an equal sign. Each letter will appear at most once, which means the number of circuit outputs is between 0 and 26, inclusive.

Each picture will be terminated by a row consisting solely of asterisk ("`*`") characters (at least one). The last picture will be followed by two such rows. No row in the input will be longer than 200 characters.

Output Specification

For each picture, print the values of all named outputs, sorted alphabetically. Each output row should contain three characters: output name (one uppercase letter), equals sign, and a binary value (zero or one). Print one empty line after each test case.

Sample Input

```
     0=+
       |
       | #######
       +=#     #
         # &   #o=--+
1=------=#     #    |
         #     #    |
         #######    +--=### ###
                    |   #=#=#1#o==X
       1=-----------x--=# # ###
                1=---x--=###
                     +------------=Y
**********************************
1=A
***
*
```

Output for Sample Input

```
X=0
Y=1

A=1
```

Problem G: Key Task

`keys.c | keys.C | keys.java | keys.p`

The Czech Technical University is rather old — you already know that it celebrates 300 years of its existence in 2007. Some of the university buildings are old as well. And the navigation in old buildings can sometimes be a little bit tricky, because of strange long corridors that fork and join at absolutely unexpected places.

The result is that some first-graders have often difficulties finding the right way to their classes. Therefore, the Student Union has developed a computer game to help the students to practice their orientation skills. The goal of the game is to find the way out of a labyrinth. Your task is to write a verification software that solves this game.

The labyrinth is a 2-dimensional grid of squares, each square is either free or filled with a wall. Some of the free squares may contain doors or keys. There are four different types of keys and doors: blue, yellow, red, and green. Each key can open only doors of the same color.

You can move between adjacent free squares vertically or horizontally, diagonal movement is not allowed. You may not go across walls and you cannot leave the labyrinth area. If a square contains a door, you may go there only if you have stepped on a square with an appropriate key before.

Input Specification

The input consists of several maps. Each map begins with a line containing two integer numbers R and C ($1 \leq R, C \leq 100$) specifying the map size. Then there are R lines each containing C characters. Each character is one of the following:

Character		Meaning
Hash mark	`#`	Wall
Dot	`.`	Free square
Asterisk	`*`	Your position
Uppercase letter	`B Y R G`	Blue, yellow, red, or green door
Lowercase letter	`b y r g`	Blue, yellow, red, or green key
Uppercase X	`X`	Exit

Note that it is allowed to have

- more than one exit,
- no exit at all,
- more doors and/or keys of the same color, and
- keys without corresponding doors and vice versa.

You may assume that the marker of your position ("`*`") will appear exactly once in every map.

There is one blank line after each map. The input is terminated by two zeros in place of the map size.

Output Specification

For each map, print one line containing the sentence "`Escape possible in` S `steps.`", where S is the smallest possible number of step to reach any of the exits. If no exit can be reached, output the string "`The poor student is trapped!`" instead.

One step is defined as a movement between two adjacent cells. Grabbing a key or unlocking a door does not count as a step.

Sample Input

```
1 10
*........X

1 3
*#X

3 20
####################
#XY.gBr.*.Rb.G.GG.y#
####################

0 0
```

Output for Sample Input

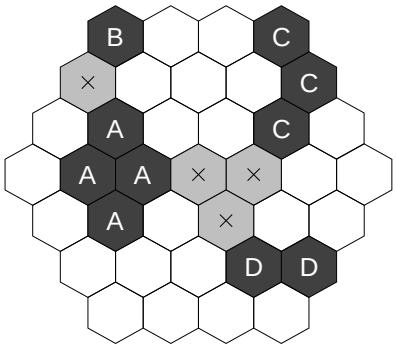
```
Escape possible in 9 steps.
The poor student is trapped!
Escape possible in 45 steps.
```

Problem H: Hexagonal Parcels

`hexagon.c` | `hexagon.C` | `hexagon.java` | `hexagon.p`

A civil engineer that has recently graduated from the Czech Technical University encountered an interesting problem and asked us for a help. The problem is more of economical than engineering nature. The engineer needs to connect several buildings with an infrastructure. Unfortunately, the investor is not the owner of all the land between these places. Therefore, some properties have to be bought first.

The land is divided into a regular "grid" of hexagonal parcels, each of them forms an independent unit and has the same value. Some of the parcels belong to the investor. These parcels form four connected areas, each containing one building to be connected with the others. Your task is to find the minimal number of parcels that must be acquired to connect the four given areas.



The whole land also has a hexagonal shape with six sides, each consisting of exactly H parcels. The above picture shows a land with $H = 4$, parcels with letters represent the four areas to be connected. In this case, it is necessary to buy four additional parcels. One of the possible solutions is marked by crosses.

Input Specification

The input contains several scenarios. Each scenario begins with an integer number H, which specifies the size of the land, $2 \leq H \leq 20$. Then there are $2.H - 1$ lines representing individual "rows" of the land (always oriented as in the picture). The lines contain one non-space character for each parcel. It means the first line will contain H characters, the second line $H + 1$, and so on. The longest line will be the middle one, with $2.H - 1$ characters. Then the "length" descends and the last line contains H parcels, again.

The character representing a parcel will be either a dot ("`.`") for the land that is not owned by the investor, or one of the uppercase letters "`A`", "`B`", "`C`", or "`D`". The areas of parcels occupied by the same letter will always be connected. It means that between any two parcels in the same area, there exists a path leading only through that area.

Beside the characters representing parcels, the lines may contain any number of spaces at any positions to improve "human readability" of the input. There is always at least one space between two letters (or the dots). After the land description, there will be one empty line and then the next scenario begins. The last scenario is followed by a line containing zero.

Output Specification

For each scenario, output one line with the sentence "`You have to buy` P `parcels.`", where P is the minimal number of parcels that must be acquired to make all four areas connected together.

Areas are considered *connected*, if it is possible to find a path between them that leads only through parcels that have been bought.

Sample Input

```
4
    B . . C
   . . . . C
  . A . . C .
 . A A . . . .
  . A . . . .
   . . . D D
    . . . .

0
```

Output for Sample Input

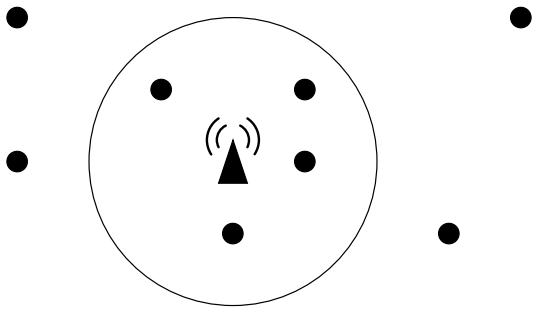
```
You have to buy 4 parcels.
```

Problem I: Phone Cell

`cell.c | cell.C | cell.java | cell.p`

Nowadays, everyone has a cellphone, or even two or three. You probably know where their name comes from. Do you? Cellphones can be moved (they are "mobile") and they use wireless connection to static stations called BTS (Base Transceiver Station). Each BTS covers an area around it and that area is called a *cell*.

The Czech Technical University runs an experimental private GSM network with a BTS right on top of the building you are in just now. Since the placement of base stations is very important for the network coverage, your task is to create a program that will find the optimal position for a BTS. The program will be given coordinates of "points of interest". The goal is to find a position that will cover the maximal number of these points. It is supposed that a BTS can cover all points that are no further than some given distance R. Therefore, the cell has a circular shape.



The picture above shows eight points of interest (little circles) and one of the possible optimal BTS positions (small triangle). For the given distance R, it is not possible to cover more than four points. Notice that the BTS does not need to be placed in an existing point of interest.

Input Specification

The input consists of several scenarios. Each scenario begins with a line containing two integer numbers N and R. N is the number of points of interest, $1 \leq N \leq 2\,000$. R is the maximal distance the BTS is able to cover, $0 \leq R < 10\,000$. Then there are N lines, each containing two integer numbers X_i, Y_i giving coordinates of the i-th point, $|X_i|, |Y_i| < 10\,000$. All points are distinct, i.e., no two of them will have the same coordinates.

The scenario is followed by one empty line and then the next scenario begins. The last one is followed by a line containing two zeros.

A point lying at the circle boundary (exactly in the distance R) is considered covered. To avoid floating-point inaccuracies, the input points will be selected in such a way that for any possible subset of points S that can be covered by a circle with the radius $R + 0.001$, there will always exist a circle with the radius R that also covers them.

Output Specification

For each scenario, print one line containing the sentence "`It is possible to cover` M `points.`", where M is the maximal number of points of interest that may be covered by a single BTS.

Sample Input

```
8 2
1 2
5 3
5 4
1 4
8 2
4 5
7 5
3 3

2 100
0 100
0 -100

0 0
```

Output for Sample Input

```
It is possible to cover 4 points.
It is possible to cover 2 points.
```

The first sample input scenario corresponds to the picture, providing that the X axis aims right and Y axis down.

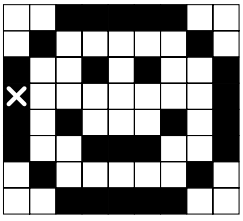
Problem J: Strange Billboard

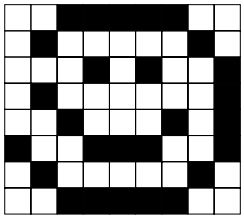
`billboard.c | billboard.C | billboard.java | billboard.p`

The marketing and public-relations department of the Czech Technical University has designed a new reconfigurable mechanical Flip-Flop Bill-Board (FFBB). The billboard is a regular two-dimensional grid of $R \times C$ square tiles made of plastic. Each plastic tile is white on one side and black on the other. The idea of the billboard is that you can create various pictures by flipping individual tiles over. Such billboards will hang above all entrances to the university and will be used to display simple pictures and advertise upcoming academic events.

To change pictures, each billboard is equipped with a "reconfiguration device". The device is just an ordinary long wooden stick that is used to tap the tiles. If you tap a tile, it flips over to the other side, i.e., it changes from white to black or vice versa. Do you agree this idea is very clever?

Unfortunately, the billboard makers did not realize one thing. The tiles are very close to each other and their sides touch. Whenever a tile is tapped, it takes all neighboring tiles with it and all of them flip over together. Therefore, if you want to change the color of a tile, all neighboring tiles change their color too. Neighboring tiles are those that touch each other with the whole side. All inner tiles have 4 neighbors, which means 5 tiles are flipped over when tapped. Border tiles have less neighbors, of course.





For example, if you have the billboard configuration shown in the left picture above and tap the tile marked with the cross, you will get the picture on the right. As you can see, the billboard reconfiguration is not so easy under these conditions. Your task is to find the fastest way to "clear" the billboard, i.e., to flip all tiles to their white side.

Input Specification

The input consists of several billboard descriptions. Each description begins with a line containing two integer numbers R and C ($1 \leq R, C \leq 16$) specifying the billboard size. Then there are R lines, each containing C characters. The characters can be either an uppercase letter "`X`" (black) or a dot "`.`" (white). There is one empty line after each map. The input is terminated by two zeros in place of the board size.

Output Specification

For each billboard, print one line containing the sentence "`You have to tap` T `tiles.`", where T is the minimal possible number of taps needed to make all squares white. If the situation cannot be solved, output the string "`Damaged billboard.`" instead.

Sample Input

```
5 5
XX.XX
X.X.X
.XXX.
X.X.X
XX.XX

5 5
.XX.X
.....
..XXX
..X.X
..X..

1 5
...XX

5 5
...X.
...XX
.XX..
..X..
.....

8 9
..XXXXX..
.X.....X.
X..X.X..X
X.......X
X.X...X.X
X..XXX..X
.X.....X.
..XXXXX..

0 0
```

Output for Sample Input

```
You have to tap 5 tiles.
Damaged billboard.
You have to tap 1 tiles.
You have to tap 2 tiles.
You have to tap 25 tiles.
```