Problem A: String LD

`Stringld` (left delete) is a function that gets a string and deletes its leftmost character (for instance `Stringld("acm")` returns `"cm"`).

You are given a list of distinct words, and at each step, we apply `stringld` on every word in the list. Write a program that determines the number of steps that can be applied until at least one of the conditions become true:

1) A word becomes empty string, or
2) a duplicate word is generated.

For example, having the list of words `aab`, `abac`, and `caac`, applying the function on the input for the first time results in `ab`, `bac`, and `aac`. For the second time, we get `b`, `ac`, and `ac`. Since in the second step, we have two `ac` strings, the condition 2 is true, and the output of your program should be 1. Note that we do not count the last step that has resulted in duplicate string. More examples are found in the sample input and output section.

Input (Standard Input)

There are multiple test cases in the input. The first line of each test case is n ($1 \le n \le 100$), the number of words.
Each of the next n lines contains a string of at most 100 lower case characters.
The input terminates with a line containing 0.

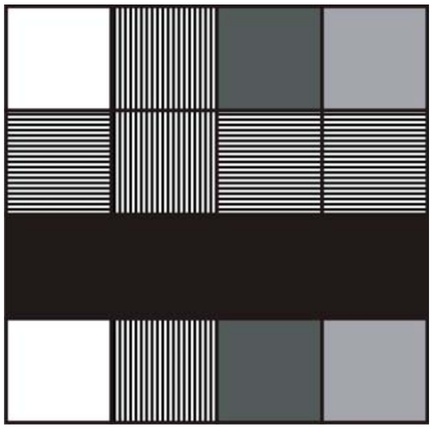
Output (Standard Output)

For each test case, write a single line containing the maximum number of `stringld` we can call.

Sample Input and Output

Standard Input	Standard Output
4 aaba aaca baabcd dcba 3 aaa bbbb ccccc 0	1 2

Problem B: Painting



Ethan wants to draw a painting on an $m \times n$ board. He can draw some strips on the board using a paintbrush of width one. In each step, he must choose a new color and paint a full column or a full row.
He has a great image to be drawn on the board, but he doesn't know which color to use first. You must help him in finding out the order of colors.

Input (Standard Input)

There are multiple test cases in the input. The first line of each test case contains two integers m and n, the size of the board ($0 < m, n < 100$). Following the first line, there are m lines with n integers denoting the color in each cell. All the colors are positive integer numbers less than 10000. The input is terminated with a single line containing two consecutive zeros.

Output (Standard Output)

For each test case, write a single line containing the order of colors used to paint the board. If there are several answers, output the one which is lexicographically smallest (considering each number as a symbol).

Sample Input and Output

Standard Input	Standard Output

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

Problem C: Another Brick in the Wall

After years as a brick-layer, you've been called upon to analyze the instability of brick walls. The instability of a wall can be approximated by the maximum damage to a wall in case of taking one brick out. A brick will fall if all bricks that are directly underneath it are removed. Note that if the space underneath a brick is partially empty, it does not fall. You are given the description of all bricks in a wall, and must determine the instability of the wall as described in the following sections.

Input (Standard Input)

There are multiple test cases in the input. Each test case consists of a single line, "M N" ($1 \le M, N \le 100$) where M and N indicate the height and width (in units), respectively, of the input wall.

Each of the next M lines is a string of N digits which specifies a row in the wall. Each brick in a row is represented by a substring of the row (like s) such that every digit in s is the same, which is equal to the length of s too. For example, 333 and 22 are two bricks of length 3 and 2 respectively, but 111 specifies three bricks of length one. A 0 in a row means there is no brick in that place of wall. Note that the height of each brick is one. The input terminates with a line containing 0 0. You may assume that the input is correct. This means:

1) There is no brick such that the length of the brick does not conform to the digits in the brick (like 222 in the row 12221).
2) No brick can fall initially.

Output (Standard Output)

For each test case, write a single line containing maximum sum of the bricks' lengths that will fall if we take one brick out (including that brick).

Sample Input and Output

Standard Input	Standard Output

4 5
33322
22333
33322
22333
4 6
122333
444422
111111
333333
3 3
022
220
111
0 0
	5
	8
	4

Problem D: Blast the Enemy!

A new computer game has just arrived and as an active and always-in-the-scene player, you should finish it before the next university term starts. At each stage of this game, you have to shoot an enemy robot on its weakness point. The weakness point of a robot is always the "center of mass" of its 2D shape in the screen. Fortunately, all robot shapes are simple polygons with uniform density and you can write programs to calculate exactly the center of mass for each polygon.

Let's have a more formal definition for center of mass (COM). The center of mass for a square, (also circle, and other symmetric shapes) is its center point. And, if a simple shape *C* is partitioned into two simple shapes *A* and *B* with areas S_A and S_B, then *COM(C)* (as a vector) can be calculated by

$$COM(C) = \frac{S_A \times COM(A) + S_B \times COM(B)}{S_A + S_B}.$$

As a more formal definition, for a simple shape *A* with area S_A:

$$COM(A) = \frac{\iint_A \vec{a}\,.\,ds}{S_A}.$$

Input (Standard Input)

The input contains a number of robot definitions. Each robot definition starts with a line containing *n*, the number of vertices in robot's polygon (*n* <= 100). The polygon vertices are specified in the next *n* lines (in either clockwise or counter-clock-wise order). Each of these lines contains two space-separated integers showing the coordinates of the corresponding vertex. The absolute value of the coordinates does not exceed 100. The case of *n*=0 shows the end of input and should not be processed.

Output (Standard Output)

The ith line of the output should be of the form "`Stage #i: x y`" (omit the quotes), where (*x*,*y*) is the center of mass for the ith robot in the input. The coordinates must be rounded to exactly 6 digits after the decimal point.

Sample Input and Output

Standard Input	Standard Output

4
0 0
0 1
1 1
1 0
3
0 1
1 0
2 2
8
1 1
2 1
2 7
3 7
3 0
0 0
0 7
1 7
0
	Stage #1: 0.500000 0.500000
	Stage #2: 1.000000 1.000000
	Stage #3: 1.500000 3.300000

Problem E: deltree

You have just run out of disk space and decided to delete some of your directories. Rationally, you will first have an exploration of what you have in your file system. And more rationally, you will do this exploration through a command line interface. The interface used in this problem is called “`MSDOS--`”, since it is something like `MSDOS` with fewer features. The commands of `MSDOS--` are as follows:

- `cd` <*directory*>

  Assuming <*directory*> to be the name of a relative descendant of current directory, this command changes the current directory to <*directory*>. For example, when the current directory is “`\A\B\`” and one of its descendants is “`C\D`”, the execution of “`cd C\D`” will change the current directory to “`\A\B\C\D\`”.

- `cd \`

  This command changes the current directory to “`\`” (the root of the file system). For example, when the current directory is “`\A\B\`”, the execution of “`cd \`” will change the current directory to “`\`”.

- `cd ..`

  Assuming the current directory to be anything except “`\`”, this command changes the current directory to its parent directory. For example, when the current directory is “`\A\B\`”, the execution of “`cd ..`” will change the current directory to “`\A\`”.

- `cd \`<*directory*>

  This command is equivalent to the execution of the following two commands:
  `cd \`
  `cd` <*directory*>

- `dir`

  This command lists the name of files and directories directly in the current directory, each on a separate line. These file/directory names are made up of (lowercase and uppercase) letters, digits, and dots (“`.`”). Directory names precede the file names in the list, and each one, comes alone in a single line. On the contrary, each file name is accompanied by its size separated by a space. A sample output of “`dir`” is as follows:
  ```
  HW1
  HW1.old
  Syllab.pdf 10000
  notes.txt 3241
  ```

- `deltree` <*directory*>

  Assuming <*directory*> to be the name of a relative descendant of current directory, this command tries to delete <*directory*> and all its descendant files and subdirectories (and thus, freeing that much of space). For example, when the current directory is “`\A\B\`” and one of its descendants is “`C\D`”, the execution of “`deltree C\D`” will try to delete directory “`\A\B\C\D\`” and all of its descendant files and directories.

- `deltree \`<*directory*>

  This command is equivalent to the execution of the following two commands:
  `cd \`
  `deltree` <*directory*>

- `exit`

  This command terminates the command line interface.

A “scenario” is an exploration (a consistent series of “`cd`” and “`dir`” commands and their results, starting from root) followed by exactly one “`deltree`” command. Given a scenario, you are to find the maximum space guaranteed to be freed by executing its “`deltree`” command.

Input (Standard Input)

Input contains multiple independent scenarios. There is an empty line after each scenario. The input ends with an “`exit`” command. There is a “**>**” sign before each command in the input (with no spaces in between). The length of each file name does not exceed 50. You may assume that the input is correct.

Output (Standard Output)

Write the result of the *i*th scenario as a single integer on the *i*th line of output.

Sample Input and Output

Standard Input

```
>cd A
>dir
B
C
d 12
e 62
>cd B
>cd ..
>cd ..
>deltree A

>dir
G
s 2
>cd G
>dir
>cd \
>deltree G

>dir
A
B
x 3
>cd A
>dir
AA
AB
ax 10
ay 12
>cd AA
>dir
d 32
a 28
>cd ..
>cd AB
>dir
F
x 100
>cd F
>dir
G
>cd \
>deltree A
>cd D1\D2
>dir
D3
a 32
>cd D3
>dir
b 31
>cd \D1\D3
>dir
d 7
>deltree \D1

>exit
```

Standard Output

```
74
0
182
70
```

Problem F: Solar Eclipse

A new Solar Eclipse is going to happen in Mars. Scientists from different parts of the world are travelling to Mars to watch and study this phenomenon. You just managed to calculate exactly the best point of Mars lands for your study of the eclipse, and want to land your flying saucer on that place. But, you notice that there are already other spacecrafts landed on near that area.

In the bird's eye view, all the spacecrafts (including yours) are circles with constant radius *R*. Logically, you hate to land your spacecraft on the others (no intersection of areas is allowed, but touching the other crafts is acceptable), though, the other saucers did not obey this rule on their own landings (i.e. their circles might have positive-area intersections with each other). In order to land your own craft on Mars, you want to find the place which minimizes the distance between the center of your flying saucer and your already calculated best point (and obeys the no-intersection rule). That's what you should do in this problem.

Input (Standard Input)

The input has multiple test cases. Each test case starts with a line containing an integer *n* (number of already landed spacecrafts), and a real number *R*. The land is small enough for us to be modeled by a two-dimensional plane, and (0,0) is conventionally the best point for us to land. Each of the next *n* lines specifies the location of a landed flying saucer by giving two real numbers *x* and y as the coordinates of its center.
The input ends with a case of $n = R = 0$ which must not be processed. Assume $n \leq 100$, $R > 0$, and their absolute value does not exceed 1000.

Output (Standard Output)

Write the result of the i^{th} test case on the i^{th} line of the output. You should just write the minimum possible distance between the center of your landed craft and the origin of the plane, rounded to exactly 6 digits after the decimal point.

Sample Input and Output

Standard Input	Standard Output

1 1.234
2.468 0
1 2
2 2
2 1
1 1
-1 -1
0 0
	0.000000
	1.171573
	1.414214

Problem G: Hurry Plotter

A *plotter* is a vector graphics printing device that connects to a computer to print graphical plots. There are two types of plotters: *pen plotters* and *electrostatic plotters*. Pen plotters print by moving a *pen* across the surface of a piece of paper. They can draw complex line art, including text, but do so very slowly because of the mechanical movement of the pens. In this problem, we are considering this matter of slowness for our special type of pen plotter. A *discrete horizontal pen plotter* can draw only horizontal line segments whose end points have discrete coordinates (integer x and y's). The drawing method is quite simple. The pen starts its journey from the upper left corner of the page (x=y=0) and moves only right while drawing the specified lines on that row. Then, it moves back completely to the left, moves one row down ($y \leftarrow y+1$), and repeats this task for the second row. The same is done for the next rows. In other words, the pen can move down only when it is far on the left side (i.e. when x=0), and can have at most one left-to-right pass and at most one right-to-left pass on each row.

It takes one unit of time to move the pen one unit of length to the left ($x \leftarrow x$-1), or to the right ($x \leftarrow x$+1). This time is doubled if the pen is on the paper and is drawing a line segment. It takes no time to move one row down (when x=0).

Since it might take a long time for the plotter to draw all the given line segments, we have decided to add a new feature to our plotter: drawing *time-limit*. By specifying the time-limit, the plotter should draw the maximum number of lines (using the same drawing method given above) that can be drawn within that time-limit. Given the time-limit and line segments, you should find this maximum number.

Input (Standard Input)

The input contains multiple test cases. Each test case starts with a line containing two integers n and t. The integer n is the number of line segments ($n \le 1000$) and t is the time-limit ($t \le 10^6$). Each of the next n lines specifies a line segment by giving three integers y, x_s, and x_t. Integer y indicates the row of that line segment ($0 \le y \le 2000$), and x_s and x_t are the x-coordinates of its end points ($0 \le x_s \le x_t \le 10^6$). The line segments are disjoint and do not have any intersections. A case of $n = t = 0$ shows the end of input and should not be processed.

Output (Standard Output)

Write the result of the i^{th} test case on the i^{th} line of output. Each line should have only one integer, indicating the maximum number of line segments that can be drawn in its corresponding test case.

Sample Input and Output

Standard Input	Standard Output

1 3
0 1 2
3 5
1 1 2
3 1 3
1 3 4
3 6
1 1 2
3 1 3
1 3 4
4 11
1 3 4
1 1 2
2 1 2
2 3 4
0 0
	1
	1
	2
	3

Problem H: ACM Coalition

In the recent parliament election, none of the parties have vast majority of seats, so there should be a coalition to select the members of the Management Board, which are one speaker, two deputy speakers and six secretaries. The board has a special voting system: the speaker has 25 votes, deputy speakers have 8 votes and each secretary has 1 vote.

ACM party decides to take a commanding role and shape the coalition, but they are *m* seats short to have a majority. They know the number of seats that every other party has taken. To participate in the coalition, each party demands its share from the Management Board in the form of a triplet (a, b, c) where a, b, and c are the number of speakers, deputy speakers, and secretaries that are expected to be selected from that party. For example, if the party BDN has a demand of (1, 1, 2), it expects that the speaker, one of the deputy speakers, and two of the secretaries are selected from BDN. A party may have multiple demands, meaning that the party accepts to participate in the coalition if one of its demands is satisfied.

Knowing the demands of all other parties, ACM wants to know how powerful it can be in Management Board. This means that ACM wants to maximize its number of votes while forming a coalition with other parties such that it overcomes its shortage of *m* seats.

Input (Standard Input)

The input contains multiple test cases. Each test case starts with a line containing two integers *n* and *m*. The integer *n* is the number of line parties ($n \leq 50$) and *m* is the number of seats ACM party needs. The next *n* lines contain other parties' information, each beginning with number of seats the party has, followed by a colon, a space, and a list of demands for that party. The list of demands is in the form of triplets (a, b, c) where $0 \leq a \leq 1$, $0 \leq b \leq 2$, and $0 \leq c \leq 6$. The triplets are separated by the string " `or` " and are terminated with a semicolon in the end (see the sample input). The input is terminated with a line containing 0 0.

Output (Standard Output)

For each test case, write a single line containing three integers, which represent speaker, deputy speaker and secretaries which ACM party can have in the coalition to have maximum votes in board's voting system.

Sample Input and Output

Standard Input	Standard Output

3 4
1: (0,0,0);
2: (1,2,0);
3: (1,0,5) or (1,2,0) or (0,2,6);
1 0
1: (1,1,1);
1 1
1: (1,1,1);
4 6
6: (1,0,0) or (1,2,6);
2: (0,2,0);
2: (0,0,3);
2: (0,0,3);
0 0
	1 0 0
	1 2 6
	0 1 5
	1 0 0

Problem I: Fruit Weights

Have you ever thought about comparing the weight of fruits? That's what you should do in this problem! Given a series of fruit weight comparisons, you should anticipate the result of another comparison. In this problem, all fruits of the same kind are identical and have the same weights. Each fruit weight comparison is something like "a X $\leq b$ Y" in which a and b are positive integers, and X and Y are fruit names. Such a comparison means that the weight of a fruits of type X is less than or equal to the weight of b fruits of type Y.

Input (Standard Input)

The input contains multiple test cases. Each test case starts with a line containing n, which is the number of given comparisons. Each of the next n lines gives a comparison in the form of "a X b Y" meaning that "a X $\leq b$ Y". The last line of each test case contains the comparison query in the same form of "a X b Y" inquiring the comparison of "a X" and "b Y".

A case of $n = 0$ shows the end of input and should not be processed. All integers in the input (except the last n which is 0) are positive and are not greater than 100. Fruit names are case-sensitive strings of (lowercase and uppercase) letters with length no more than 50.

Output (Standard Output)

For each test case, write one line with your result for that test case. Your result can be one of the followings (assume the comparison query was "a X b Y"):

- "<=": meaning you are sure that "a X $\leq b$ Y".
- ">=": meaning you are sure that "a X $\geq b$ Y".
- "==": meaning you are sure that "a X $= b$ Y" (i.e. you have reached both of the above results).
- "UNAVAILABLE": meaning that you can say nothing for sure in comparing "a X" and "b Y" (i.e. you have reached none of the above results).
- "INCONSISTENT": meaning that there is an inconsistency in the given comparisons (i.e. you are sure that all the given comparisons for that test case cannot hold at the same time).

Sample Input and Output

Standard Input	Standard Output
2 2 Orange 3 Apple 1 Apple 1 Peach 2 Orange 3 Peach 1 2 Orange 3 Apple 2 Orange 2 Apple 2 3 a 2 A 2 A 3 a 5 A 5 a 2 3 B 2 A 2 A 3 B 2 A 3 B 3 2 b 2 A 2 A 2 C 3 C 2 b 1 A 1 b 0	<= UNAVAILABLE >= == INCONSISTENT

Problem J: Royal Gems

In the game of Royal Gems, you are given an $n \times m$ board and arbitrarily large number of ruby, emerald, sapphire, and diamond gemstones. You must put one gemstone in each cell of the board according to the following rules:

1) Every ruby has an emerald, a sapphire and a diamond in his neighbors.
2) Every emerald has a sapphire and a diamond in her neighbors.
3) Every sapphire has a diamond in his neighbors.

A neighbor of a cell is one of the four cells that are directly above, below, left, or right of the cells. Write a program that finds the maximum number of ruby gemstones that could be put on the board satisfying the above rules.

Input (Standard Input)

There are multiple test cases in the input. Each test case consists of n ($1 < n < 8$) and m ($1 < m < 8$). The input terminates with a line containing 0 0.

Output (Standard Output)

For each test case, write a single line containing the maximum number of ruby gemstones on the board.

Sample Input and Output

Standard Input	Standard Output

2 2
2 3
3 3
0 0
	0
	1
	2