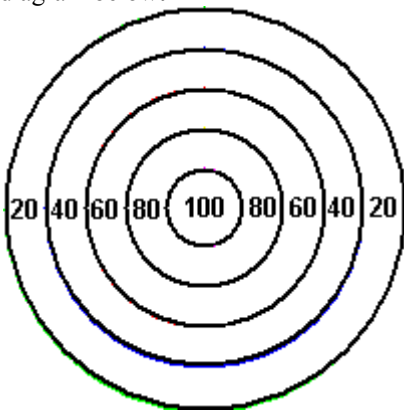
A• Bullseye

Problem

A simple dartboard consists of a flat, circular piece of cork with concentric rings drawn on it. Darts are thrown at the board by players in an attempt to hit the center of the dartboard (the *Bullseye*). The region between each pair of rings (or the center and the first ring) represents a certain point value. The closer the region is to the center of the dartboard, the more points the region is worth, as shown in the diagram below:



Ring radii are at 3", 6", 9", 12" and 15" (the *Bullseye* has a diameter of 6"). A game of *Simple Darts* between two players is played as follows. The first player throws 3 darts at the board. A score is computed by adding up the point values of each region that a dart lands in. The darts are removed. The second player throws 3 darts at the board; the score for player two is computed the same way as it is for player one. The player with the higher score wins.

For this problem, you are to write a program that computes the scores for two players, and determine who, if anyone, wins the game. If a dart lands exactly on a ring (region boundary), the higher point value is awarded. Any dart outside the outer ring receives no points. For the purposes of this problem, you can assume that a dart has an infinitely fine point and can not land *paritially* on a ring; it is either *on* the ring or it is *not on* the ring. Standard double precision floating point operations will be should be used.

Input

Input consists of 1 or more datasets. A dataset is a line with 12 double-precision values separated by spaces. Each pair of values represents the ***X*** and ***Y*** distances respectively of a dart from the center of the board in inches. (the center is located at ***X=0, Y=0***. The range of values are: **-20.0 <= *X,Y* <= 20.0.** Player one's darts are represented by the first 3 pairs of values, and player two's by the last 3 pairs of values. Input is terminated by the first value of a dataset being **–100.**

Output

For each dataset, print a line of the form:

```
SCORE: N to M, PLAYER P WINS.
```

Or:

```
SCORE: N to M, TIE.
```

N is player one's score, and ***M*** is player two's score. ***P*** is either ***1*** or ***2*** depending on which player wins. All values are non-negative integers.

Formula

Recall: $\mathbf{r}^2 = x^2 + y^2$ where ***r*** is the radius, and (***x***,***y***) are the coordinates of a point on the circle.

Example

Input	Output
-9 0 0 -4.5 -2 2 9 0 0 4.5 2 -2 -19.0 19.0 0 0 0 0 3 3 6 6 12 12 -100 0 0 0 0 0 0 0 0 0 0 0	SCORE: 240 to 240, TIE. SCORE: 200 to 140, PLAYER 1 WINS.

B • An Excel-lent Problem

Problem

A certain spreadsheet program labels the columns of a spreadsheet using letters. Column 1 is labeled as "A", column 2 as "B", …, column 26 as "Z". When the number of columns is greater than 26, another letter is used. For example, column 27 is "AA", column 28 is "AB" and column 52 is "AZ". It follows that column 53 would be "BA" and so on. Similarly, when column "ZZ" is reached, the next column would be "AAA", then "AAB" and so on.

The rows in the spreadsheet are labeled using the row number. Rows start at 1.

The designation for a particular cell within the spreadsheet is created by combining the column label with the row label. For example, the upper-left most cell would be "A1". The cell at column 55 row 23 would be "BC23".

You will write a program that converts numeric row and column values into the spreadsheet designation.

Input

Input consists of lines of the form: **R*n*C*m***. ***n*** represents the row number [1,300000000] and ***m*** represents the column number, **1 <= *m* <= 300000000**. The values ***n*** and ***m*** define a single cell on the spreadsheet. Input terminates with the line: **R0C0** (that is, ***n*** and ***m*** are 0). There will be no leading zeroes or extra spaces in the input.

Output

For each line of input (except the terminating line), you will print out the spreadsheet designation for the specified cell as described above.

Example

Input	Output
R1C1 R3C1 R1C3 R299999999C26 R52C52 R53C17576 R53C17602 R0C0	A1 A3 C1 Z299999999 AZ52 YYZ53 YZZ53

C• Lenny's Lucky Lotto Lists

Problem

Lotto is a lottery, typically with an accumulating jackpot, in which participants play numbers of their choice in a random drawing. Lenny likes to play the lotto in Lincoln county Louisiana. In the game, he picks a list of ***n*** numbers in the range from **1** to ***m***. If his list matches the drawn list, he wins the big prize, a lifetime supply of large lemons.

Lenny has a scheme that he thinks is likely to be lucky. He likes to choose his list so that each number in it is at least twice as large as the one before it. So, for example, if ***n*** = **4** and ***m*** = **10**, then the possible lucky lists Lenny could like are:

```
1 2 4 8
1 2 4 9
1 2 4 10
1 2 5 10
```

Thus Lenny has 4 lists to choose from.

Your job, given ***n*** and ***m***, is to count how many lucky lists Lenny has to choose from.

Input

The first line of input is a single non-negative integer, which is the number of data sets to follow. All data sets should be handled identically.

The next lines, one per data set, contain two integers, ***n*** and ***m***. You are guaranteed that **1** <= ***n*** <= **10** and **1** <= ***m*** <= **2000** and ***n*** <= ***m***.

Output

For each data set, print a line like the following:

Data set *i*: *n m number*

where ***i*** is the data set number (beginning with 1), and ***number*** is the maximum number of lucky lists corresponding to the provided values of ***n*** and ***m***.

Example

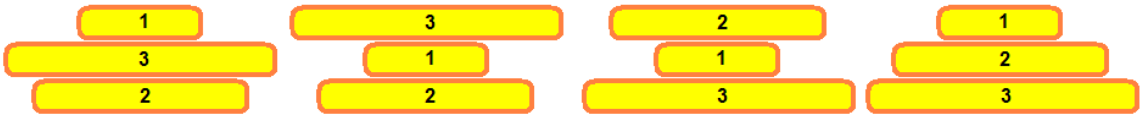
Input	Output
1 4 10	Data set 1: 4 10 4

D • Flipping Pancakes

Problem

We start with a stack *n* of pancakes of distinct sizes. The problem is to convert the stack to one in which the pancakes are in size order with the smallest on the top and the largest on the bottom. To do this, we are allowed to flip the top *k* pancakes over as a unit (so the *k-th* pancake is now on top and the pancake previously on top is now in the *k-th* position).

For example:



1 3 2 (flip 2) 3 1 2 (flip 3) 2 1 3 (flip 2) 1 2 3

This problem is to write a program, which finds a sequence of at most *(2n – 3)* flips, which converts a given stack of pancakes to a sorted stack.

Input

Each line of the input gives a separate data set as a sequence of numbers separated by spaces. The first number on each line gives the number, *N*, of pancakes in the data set. The input ends when *N* is 0 (zero) with no other data on the line. The remainder of the data set are the numbers 1 through *N* in some order giving the initial pancake stack. The numbers indicate the relative sizes of the pancakes. *N* will be, at most, 30.

Output

For each data set, the output is a single-space separated sequence of numbers on a line. The first number on each line, *K*, gives the number of flips required to sort the pancakes. This number is followed by a sequence of *K* numbers, each of which gives the number of pancakes to flip on the corresponding sorting step. There may be several correct solutions for some datasets. For instance `3 3 2 3` is also a solution to the first problem below.

Example

Input	Output

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

E• Card Trick

Problem

The following card trick is performed by a Magician and the Assistant. The Assistant asks a member of the audience to choose 5 cards from a standard deck of 52 playing cards (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K of C[lubs], D[iamonds], H[earts] and S[pades]). After examining the cards, the Assistant gives one of the cards back to the audience member and then hands the remaining cards to the Magician, one at a time in a specific order. After suitable mumbo-jumbo, the Magician identifies the (fifth) card held by the audience member.

The Magician determines the card as follows:

The order of the cards in the deck is determined first by the value and for cards of the same value by the suit (both in the order given above). So the total order of cards is: AC, AD, AH, AS, 2C,2D, … , KH, KS

1. Remember the suit and value of the first card.
2. Among the remaining three cards find the position of the smallest card (in the above order). Add this position (1, 2, or 3) to the value of the first card.
3. If the larger two of the last three cards are not in order, add 3 to the result of step 2.
4. The missing card has the same suit as the first card and value that computed in step 3 wrapping around if necessary.

For example:
QH, 10D, 10C, 4D
Smallest of the last 3 cards is 4D in place 3. 10D and 10C are out of order so add 3 + 3 to Q. Wrapping around the missing card is 5H.

This problem is to write a program to perform the function of the Assistant.

Input

The first line of the input consists of a positive integer ***n***, which is the number of datasets that follow. Each of the ***n*** following lines contains one data set. The dataset is a sequence of 5 cards separated by a space. Each card is given by a one or two character value and a one character suit as described in the first paragraph.

Output

For each dataset, output on a single line the card the Assistant gives back to the audience member, followed by the four remaining cards in the order they should be presented to the Magician so that she can determine the card that was given back to the audience member.

Example

Input	Output
2 QH 5H 10C 4D 10D 5C KS 8D 7H 6C	5H QH 10D 10C 4D 6C 5C 7H 8D KS

F• Model Rocket Height

Problem

One method for determining the height achieved by a model rocket is to have three observers **A**, **B** and **C** equally spaced **D** feet apart along a line at one edge of the flat test field. Each observer has a theodolite or some other device for measuring angle above the horizontal of a distant object. Each measuring device is on a stand **H** feet above the field. When a rocket is fired, near the top of its flight, it deploys a parachute and emits a puff of dust. Each observer measures the angle above the horizontal of the puff of dust from their location. From these angles α for **A**, β for **B** and γ for **C**, the height of the rocket above the field can be determined.

This problem is to write a program which, given the parameters **D** (the distance between observers along the line in feet), **H** (the distance of the measuring device above the field in feet), α (the angle of the rocket above the horizontal in degrees measured by the leftmost observer **A**), β (the angle above the horizontal in degrees observed by the center observer **B**) and γ (the angle above the horizontal in degrees measured by the rightmost observer **C**), computes the height of the rocket above the field in feet to the *nearest foot*.

Input

The first line of input contains the parameters **D** and **H** in that order as decimal numbers (not necessarily integers). These values would be measured once at the beginning of the day and remain fixed through all rocket shots. Each succeeding line of input will contain the angles α, β and γ in that order (measured in degrees). The last line of input will contain at least one value *less than* or *equal to* zero. Other than the last line indicating the end of data all angles will be strictly between 0 and 90 degrees.

Output

For each set of three angles (other than the end indicator), the output contains a line with the height above the field in feet (to the nearest foot) with no leading spaces. (Note: *x*.5 rounds up to *x+1*.)

Example

Input	Output
50 4 43.88 46.85 40.70 34.52 39.50 35.43 27.05 29.22 26.14 0 0 0	90 70 60

G• Region FIlling

Problem

This problem is to write a program to fill a region in a rectangular array of pixels given the outline of the region.

Input

The input is a sequence of *problem instances*. Each *problem instance* begins with a line containing the *row-count* of the array, the *column-count* of the array and the *number-of-regions* to fill as decimal integers. Rows are counted from top to bottom beginning with 1. Columns are counted from left to right beginning with 1. The input ends with a *row-count* of 0. *Row-count* will be at most 47 and column-count will be at most 63.

The first line of each *problem instance* is followed by region descriptions for *number-of-regions* regions. Each region description begins with a line containing: a single character to be used to fill the region, the *row number* of the start pixel, the *column number* of the start pixel and the *number of pixels* in the boundary which will always be at least two. The region fill character will be distinct for each region within a problem instance. This line is followed by lines of direction codes (*up* = A, *up right* = B, etc.):

```
H  A  B
G     C
F  E  D
```

describing the outline traversed clockwise. The start pixel may be any point on the outline.

Output

For each *problem instance*, the output consists of *row-count* lines of *column-count* characters each of which is either a period (`.`) indicating than no region includes that pixel or the fill character of the region which includes that pixel. The *row-count* lines are followed by a single blank line. This array may be preceded by one or more lines of the following form (where A or B are the fill characters specified in the region header):

REGION A GOES OUTSIDE THE ARRAY

If the boundary path goes outside the boundaries of the array specified for the problem instance.

REGION A BOUNDARY IS NOT CLOSED

If the boundary as specified does not return to the start point.

REGION B BOUNDARY INTERSECTS REGION A

If the boundary of region B contains points of a previously specified region. If a previous region fails any of these tests it is not considered for intersection with following regions.

For each region, the first condition to be violated is to be displayed. Any region for which an error line is given will not be filled in the rectangular array.

Example

Input

```
20 40 4
B 3 21 22
CCDDDCBBBCCFFFFFGHHHHH
C 5 8 36
CCDCDDDEDEEFEFFFGFGGHGHHHAHAABABBBCB
D 10 24 38
CCCCCCEEEGGFEDCCEEEGGGGGGAAACCBAHGGAAA
A 2 2 3
CEH
10 20 4
A 4 6 10
GGAAACCEEE
B 6 16 30
CCCCCCCCCCCCEEEGGGGGGGGGGGGAAA
C 5 6 10
CCCCDDFFFG
D 6 2 10
AAACCEEEGG
0 0 0
```

Output

```
........................................
.AA.....................................
..A..................BBB......BBB........
....................BBB....BBB.........
.......CCC............BBB..BBB..........
.....CCCCCCC...........BBBBBB...........
....CCCCCCCCC...........BBBB............
...CCCCCCCCCCC...........BB.............
..CCCCCCCCCCCCC.........................
..CCCCCCCCCCCCC........DDDDDDD..........
.CCCCCCCCCCCCCCC.......DDDDDDD..........
.CCCCCCCCCCCCCCC.......DDDDDDD..........
.CCCCCCCCCCCCCCC.......DDDDDDD..........
..CCCCCCCCCCCCC...........D..............
..CCCCCCCCCCCCC...........D..............
...CCCCCCCCCCC.........DDDDDDD..........
....CCCCCCCCC.........DDDDDDD..........
.....CCCCCCC...........DDDDDDD..........
.......CCC............DDDDDDD..........
........................................

REGION B GOES OUTSIDE THE ARRAY
REGION C BOUNDARY IS NOT CLOSED
REGION D BOUNDARY INTERSECTS REGION A
...AAA..............
...AAA..............
...AAA..............
...AAA..............
...................
...................
...................
...................
...................
...................
```

H• Histology Assistant

Problem

An application to assist in the analysis of tissue samples is to work as follows. A digital microphotograph of a stained tissue sample is scanned to identify stained pixels. For each region of stained pixels, an outline of the region is obtained. The outline is then analyzed for shape indicators of disease and the outlines (color-coded for possible disease) are overlaid on the microphotograph as it is displayed to the pathologist.

This problem is to write a program, which processes a bitmap of stained and unstained pixels, finds regions of stained pixels and, for each region, outputs the outline of the region. Regions with fewer stained pixels than a *minimum size* are ignored. Only the outer boundary is computed (interior holes are ignored).

A pixel is *adjacent* to another pixel if the second pixel is directly above, directly below, directly left or directly right of the first pixel. Two stained pixels are *connected* if there is a sequence of stained pixels starting with one of the pixels and ending with the other for which each pixel in the sequence is adjacent to the next. A *region* of stained pixels is a set of stained pixels, all of which are connected to a single stained pixel. A stained pixel is a *boundary pixel* of its region if at least one of the pixels adjacent to it is not stained. (All pixels immediately outside the bitmap are considered unstained so that pixels on the edge of the bitmap are boundary pixels.) In the example below, there are 4 regions (stained pixels are 'X', unstained are '.').

```
.....................................
.XX...................................
..X................XXX......XXX........
...................XXX....XXX.........
.......XXX...........XXX..XXX.........
.....XXXXXXX..........XXXXXX..........
....XXXXXXXXX..........XXXX...........
...XXXX...XXXX..........XX............
..XXX.......XXX.......................
..XXX.......XXX........XXXXXXX........
.XXX.........XXX.......XXXXXXX........
.XXX.........XXX.......XXXXXXX........
.XXX.........XXX.......XXXXXXX........
..XXX.......XXX...........X...........
..XXX.......XXX...........X...........
...XXXX...XXXX.........XXXXXXX........
....XXXXXXXXX.........XXXXXXX........
.....XXXXXXX..........XXXXXXX........
.......XXX............XXXXXXX........
.....................................
```

Outlines are to be specified as the left most point of the top most line of the region, a count of boundary pixels and a sequence of moves from one boundary pixel to the next clockwise using the codes (*up* = A, *up right* = B, etc.):

```
H  A  B
G     C
F  E  D
```

Rows are numbered from top to bottom beginning with 1. Columns are numbered from left to right beginning with 1. For example the outline of the 'v' shaped region above would be:

3 21 22
CCDDDCBBBCCFFFFFGHHHHH

Input

Input is a sequence of *problem instances*. Each *problem instance* begins with a line containing 3 decimal numbers: *row-count*, *column-count* and *minimum-number-of-pixels*. This line is followed by *row-count* lines of *column-count* characters. Each character is either a period (`.`) for an unstained pixel or an upper-case X for a stained pixel. The input ends when the *row-count* is 0. *Row-count* will be at most 47, *column-count* will be at most 63 and *minimum-number-of-pixels* will be at least 2.

Output

For each *problem instance*, the output begins with a line starting with a decimal integer giving the *number of components* of at least *minimum-number-of-pixels* stained pixels. This is followed by the description of the boundary of each component. The boundaries are to be listed in the order that a first pixel of the component appears while scanning across lines from left to right with line scanned from top to bottom. For each component, the output begins with a line giving the *row number* of the start pixel, the *column number* of the start pixel and the number of pixels in the boundary as decimal integers separated by a single space. This line is followed by lines of direction codes 'A' through 'H'. Each line shall have 40 characters except the last line.

Example

Input

```
20 40 4
........................................
.XX.....................................
..X.................XXX......XXX........
...................XXX....XXX..........
.......XXX............XXX..XXX..........
.....XXXXXXX............XXXXXX..........
....XXXXXXXXX............XXXX...........
...XXXX...XXXX............XX............
..XXX.......XXX.........................
..XXX.......XXX........XXXXXX...........
.XXX.........XXX.......XXXXXX...........
.XXX.........XXX.......XXXXXX...........
.XXX.........XXX.......XXXXXX...........
..XXX.......XXX............X.............
..XXX.......XXX...........X..............
...XXXX...XXXX.........XXXXXX...........
....XXXXXXXXX..........XXXXXX...........
.....XXXXXXX............XXXXXX...........
.......XXX..............XXXXXX...........
........................................
12 40 4
.X.X.X.X.X.X......XX...XXXXXXXXXXXXXXXX.
.XXX.XXX.XXX......XX..XXXXXXXXXXXXXXXXXX
.X...X...X........XX..XX.............XXX
.X.X.X.X.X.X......XX..XX...XXXXXXXX...XX
.XXX.XXX.XXX......XX..XX..XXXXXXXXXX..XX
.X...X...X........XX..XX..XX......XX..XX
.X.X.X.X.X.X......XX..XXX........XXX..XX
.XXX.XXX.XXX......XX..XXXXXXXXXXXXXX..XX
.X...X...X........XX...XXXXXXXXXXXX...XX
.X.X.X.X.X.X......XXX................XXX
.XXXXXXXXXX......XXXXXXXXXXXXXXXXXXXXXX
..................XXXXXXXXXXXXXXXXXXXX.
0 0 0
```

Output

```
3
3 21 22
CCDDDCBBBCCFFFFFGHHHHH
5 8 36
CCDCDDDEDEEFEFFFGFGGHGHHHAHAABABBBCB
10 24 38
CCCCCCEEEGGFEDCCEEEGGGGGGAAACCBAHGGAAA
2
1 2 103
DBEGFEDBEGFEDBEGFEDBDBAAAAAAAAADBEGFEDBE
GFEDBEGFEDBDBAAAAAAAAADBEGFEDBEGFEDBEGFE
DBEGGGGGGGGGGAAAAAAAAAA
1 19 159
CEEEEEEEEDDCCCCCCCCCCCCCCBBAAAAAHHGGGGG
GGGGGGGFEEEDDCCCCCCCBBHGGGGGFGABCCCCCCCD
EEEFGGGGGGGGGGGHAAAAAABCCCCCCCCCCCCCCDE
EEEEEEEEFGGGGGGGGGGGGGGGGGGHAAAAAAAAA
```

I• Mr. Young's Picture Permutations

Problem

Mr. Young wishes to take a picture of his class. The students will stand in rows with each row no longer than the row behind it and the left ends of the rows aligned. For instance, 12 students could be arranged in rows (from back to front) of 5, 3, 3 and 1 students.

```
X X X X X
X X X
X X X
X
```

In addition, Mr. Young wants the students in each row arranged so that heights decrease from left to right. Also, student heights should decrease from the back to the front. Thinking about it, Mr. Young sees that for the 12-student example, there are at least two ways to arrange the students (with 1 as the tallest etc.):

```
 1  2  3  4  5      1  5  8 11 12
 6  7  8           2  6  9
 9 10 11           3  7 10
12                 4
```

Mr. Young wonders how many different arrangements of the students there might be for a given arrangement of rows. He tries counting by hand starting with rows of 3, 2 and 1 and counts 16 arrangements:

```
123 123 124 124 125 125 126 126 134 134 135 135 136 136 145 146
45  46  35  36  34  36  34  35  25  26  24  26  24  25  26  25
6   5   6   5   6   4   5   4   6   5   6   4   5   4   3   3
```

Mr. Young sees that counting by hand is not going to be very effective for any reasonable number of students so he asks you to help out by writing a computer program to determine the number of different arrangements of students for a given set of rows.

Input

The input for each problem instance will consist of two lines. The first line gives the number of rows, ***k,*** as a decimal integer. The second line contains the lengths of the rows from back to front (***n1, n2, …nk***) as decimal integers separated by a single space. The problem set ends with a line with a row count of **0**. There will never be more than **5** rows and the total number of students, ***N***, (sum of the row lengths) will be at most **30**.

Output

The output for each problem instance shall be the number of arrangements of the ***N*** students into the given rows so that the heights decrease along each row from left to right and along each column from back to front as a decimal integer. (Assume all heights are distinct.) The result of each problem instance should be on a separate line. The input data will be chosen so that the result will always fit in an unsigned 32 bit integer.

Example

Input	Output
1 30 5 1 1 1 1 1 3 3 2 1 4 5 3 3 1 5 6 5 4 3 2 2 15 15 0	1 1 16 4158 141892608 9694845