|  |
| --- |
| Some of UVA Problems |

04

list

1. **101 - The Blocks Problem**

Time limit: 3.000 seconds

|  |
| --- |
| **The Blocks Problem** |

## Background

Many areas of Computer Science use simple, abstract domains for both analytical and empirical studies. For example, an early AI study of planning and robotics (STRIPS) used a block world in which a robot arm performed tasks involving the manipulation of blocks.

In this problem you will model a simple block world under certain rules and constraints. Rather than determine how to achieve a specified state, you will ``program'' a robotic arm to respond to a limited set of commands.

## The Problem

The problem is to parse a series of commands that instruct a robot arm in how to manipulate blocks that lie on a flat table. Initially there are *n* blocks on the table (numbered from 0 to *n*-1) with block *bi* adjacent to block *bi*+1 for all$0 \leq i < n-1$ as shown in the diagram below:

|  |
| --- |
| \begin{figure} \centering \setlength{\unitlength}{0.0125in} % \begin{picture} (2... ...raisebox{0pt}[0pt][0pt]{$\bullet \bullet \bullet$ }}} \end{picture} \end{figure} |

**Figure:** Initial Blocks World

The valid commands for the robot arm that manipulates blocks are:

* move *a* onto *b*

where *a* and *b* are block numbers, puts block *a* onto block *b* after returning any blocks that are stacked on top of blocks *a* and *b* to their initial positions.

* move *a* over *b*

where *a* and *b* are block numbers, puts block *a* onto the top of the stack containing block *b*, after returning any blocks that are stacked on top of block *a*to their initial positions.

* pile *a* onto *b*

where *a* and *b* are block numbers, moves the pile of blocks consisting of block*a*, and any blocks that are stacked above block *a*, onto block *b*. All blocks on top of block *b* are moved to their initial positions prior to the pile taking place. The blocks stacked above block *a* retain their order when moved.

* pile *a* over *b*

where *a* and *b* are block numbers, puts the pile of blocks consisting of block *a*, and any blocks that are stacked above block *a*, onto the top of the stack containing block *b*. The blocks stacked above block *a* retain their original order when moved.

* quit

terminates manipulations in the block world.

Any command in which *a* = *b* or in which *a* and *b* are in the same stack of blocks is an illegal command. All illegal commands should be ignored and should have no affect on the configuration of blocks.

## The Input

The input begins with an integer *n* on a line by itself representing the number of blocks in the block world. You may assume that 0 < *n* < 25.

The number of blocks is followed by a sequence of block commands, one command per line. Your program should process all commands until the quit command is encountered.

You may assume that all commands will be of the form specified above. There will be no syntactically incorrect commands.

## The Output

The output should consist of the final state of the blocks world. Each original block position numbered *i* ( $0 \leq i < n$ where *n* is the number of blocks) should appear followed immediately by a colon. If there is at least a block on it, the colon must be followed by one space, followed by a list of blocks that appear stacked in that position with each block number separated from other block numbers by a space. Don't put any trailing spaces on a line.

There should be one line of output for each block position (i.e., *n* lines of output where*n* is the integer on the first line of input).

## Sample Input

10

move 9 onto 1

move 8 over 1

move 7 over 1

move 6 over 1

pile 8 over 6

pile 8 over 5

move 2 over 1

move 4 over 9

quit

## Sample Output

0: 0

1: 1 9 2 4

2:

3: 3

4:

5: 5 8 7 6

6:

7:

8:

9:

### 674 - Coin Change

### Time limit: 3.000 seconds

|  |
| --- |
| **Coin Change** |

Suppose there are 5 types of coins: 50-cent, 25-cent, 10-cent, 5-cent, and 1-cent. We want to make changes with these coins for a given amount of money.

For example, if we have 11 cents, then we can make changes with one 10-cent coin and one 1-cent coin, two 5-cent coins and one 1-cent coin, one 5-cent coin and six 1-cent coins, or eleven 1-cent coins. So there are four ways of making changes for 11 cents with the above coins. Note that we count that there is one way of making change for zero cent.

Write a program to find the total number of different ways of making changes for any amount of money in cents. Your program should be able to handle up to 7489 cents.

## Input

The input file contains any number of lines, each one consisting of a number for the amount of money in cents.

## Output

For each input line, output a line containing the number of different ways of making changes with the above 5 types of coins.

## Sample Input

11

26

## Sample Output

4

13

Miguel Revilla   
2000-08-14

### 572 - Oil Deposits

### Time limit: 3.000 seconds

|  |
| --- |
| **Oil Deposits** |

The GeoSurvComp geologic survey company is responsible for detecting underground oil deposits. GeoSurvComp works with one large rectangular region of land at a time, and creates a grid that divides the land into numerous square plots. It then analyzes each plot separately, using sensing equipment to determine whether or not the plot contains oil.

A plot containing oil is called a pocket. If two pockets are adjacent, then they are part of the same oil deposit. Oil deposits can be quite large and may contain numerous pockets. Your job is to determine how many different oil deposits are contained in a grid.

## Input

The input file contains one or more grids. Each grid begins with a line containing *m* and*n*, the number of rows and columns in the grid, separated by a single space. If *m* = 0 it signals the end of the input; otherwise $1 \le m \le 100$ and $1 \le n \le 100$. Following this are *m* lines of *n* characters each (not counting the end-of-line characters). Each character corresponds to one plot, and is either `\*', representing the absence of oil, or `@', representing an oil pocket.

## Output

For each grid, output the number of distinct oil deposits. Two different pockets are part of the same oil deposit if they are adjacent horizontally, vertically, or diagonally. An oil deposit will not contain more than 100 pockets.

## Sample Input

1 1

\*

3 5

\*@\*@\*

\*\*@\*\*

\*@\*@\*

1 8

@@\*\*\*\*@\*

5 5

\*\*\*\*@

\*@@\*@

\*@\*\*@

@@@\*@

@@\*\*@

0 0

## Sample Output

0

1

2

2

### 476 - Points in Figures: Rectangles

### Time limit: 3.000 seconds

|  |
| --- |
| **Points in Figures: Rectangles** |

Given a list of rectangles and a list of points in the *x*-*y* plane, determine for each point which figures (if any) contain the point.

## Input

There will be *n*( tex2html_wrap_inline220 ) rectangles descriptions, one per line. The first character will designate the type of figure (``r'' for rectangle). This character will be followed by four real values designating the *x*-*y* coordinates of the upper left and lower right corners.

The end of the list will be signalled by a line containing an asterisk in column one.

The remaining lines will contain the *x*-*y* coordinates, one per line, of the points to be tested. The end of this list will be indicated by a point with coordinates 9999.9 9999.9; these values should not be included in the output.

Points coinciding with a figure border are not considered inside.

## Output

For each point to be tested, write a message of the form:

Point i is contained in figure j

for each figure that contains that point. If the point is not contained in any figure, write a message of the form:

Point i is not contained in any figure

Points and figures should be numbered in the order in which they appear in the input.

## Sample Input

r 8.5 17.0 25.5 -8.5

r 0.0 10.3 5.5 0.0

r 2.5 12.5 12.5 2.5

\*

2.0 2.0

4.7 5.3

6.9 11.2

20.0 20.0

17.6 3.2

-5.2 -7.8

9999.9 9999.9

## Sample Output

Point 1 is contained in figure 2

Point 2 is contained in figure 2

Point 2 is contained in figure 3

Point 3 is contained in figure 3

Point 4 is not contained in any figure

Point 5 is contained in figure 1

Point 6 is not contained in any figure

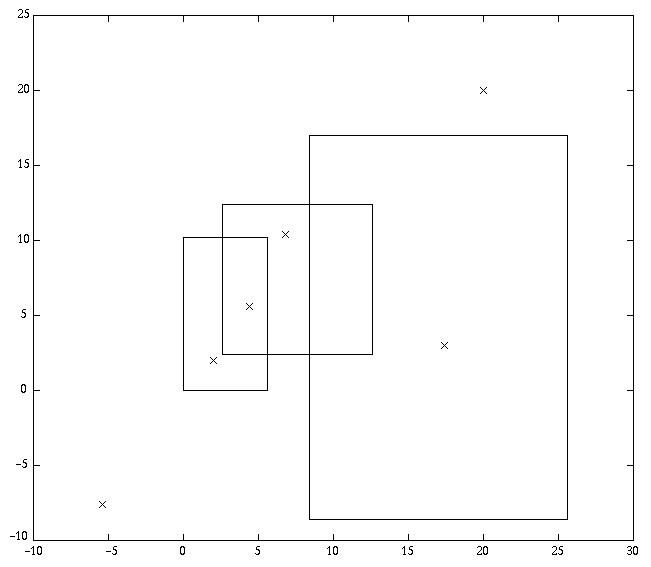


Diagrama of sample input figures and data points

### 138 - Street Numbers

### Time limit: 3.000 seconds

|  |
| --- |
| **Street Numbers** |

A computer programmer lives in a street with houses numbered consecutively (from 1) down one side of the street. Every evening she walks her dog by leaving her house and randomly turning left or right and walking to the end of the street and back. One night she adds up the street numbers of the houses she passes (excluding her own). The next time she walks the other way she repeats this and finds, to her astonishment, that the two sums are the same. Although this is determined in part by her house number and in part by the number of houses in the street, she nevertheless feels that this is a desirable property for her house to have and decides that all her subsequent houses should exhibit it.

Write a program to find pairs of numbers that satisfy this condition. To start your list the first two pairs are: (house number, last number):

6 8

35 49

## Input and Output

There is no input for this program. Output will consist of 10 lines each containing a pair of numbers, each printed right justified in a field of width 10 (as shown above).

### 439 - Knight Moves

### Time limit: 3.000 seconds

|  |
| --- |
| **Knight Moves** |

A friend of you is doing research on the Traveling Knight Problem (TKP) where you are to find the shortest closed tour of knight moves that visits each square of a given set of *n* squares on a chessboard exactly once. He thinks that the most difficult part of the problem is determining the smallest number of knight moves between two given squares and that, once you have accomplished this, finding the tour would be easy.

Of course you know that it is vice versa. So you offer him to write a program that solves the "difficult" part.

Your job is to write a program that takes two squares *a* and *b* as input and then determines the number of knight moves on a shortest route from *a* to *b*.

## Input Specification

The input file will contain one or more test cases. Each test case consists of one line containing two squares separated by one space. A square is a string consisting of a letter (a-h) representing the column and a digit (1-8) representing the row on the chessboard.

## Output Specification

For each test case, print one line saying "To get from *xx* to *yy* takes *n*knight moves.".

## Sample Input

e2 e4

a1 b2

b2 c3

a1 h8

a1 h7

h8 a1

b1 c3

f6 f6

## Sample Output

To get from e2 to e4 takes 2 knight moves.

To get from a1 to b2 takes 4 knight moves.

To get from b2 to c3 takes 2 knight moves.

To get from a1 to h8 takes 6 knight moves.

To get from a1 to h7 takes 5 knight moves.

To get from h8 to a1 takes 6 knight moves.

To get from b1 to c3 takes 1 knight moves.

To get from f6 to f6 takes 0 knight moves.

### 414 - Machined Surfaces

### Time limit: 3.000 seconds

|  |
| --- |
| **Machined Surfaces** |

An imaging device furnishes digital images of two machined surfaces that eventually will be assembled in contact with each other. The roughness of this final contact is to be estimated.

A digital image is composed of the two characters, "X" and " " (space). There are always 25 columns to an image, but the number of rows, *N*, is variable. Column one (1) will always have an "X" in it and will be part of the left surface. The left surface can extend to the right from column one (1) as contiguous X's.

Similarly, column 25 will always have an "X" in it and will be part of the right surface. The right surface can extend to the left from column 25 as contiguous X's.

Digital-Image View of Surfaces

Left Right

XXXX XXXXX tex2html_wrap_inline50

XXX XXXXXXX

XXXXX XXXX

XX XXXXXX

. .

. .

. .

XXXX XXXX

XXX XXXXXX tex2html_wrap_inline52

tex2html_wrap_inline54 tex2html_wrap_inline56

1 25

In each row of the image, there can be zero or more space characters separating the left surface from the right surface. There will never be more than a single blank *region*in any row.

For each image given, you are to determine the total ``void" that will exist after the left surface has been brought into contact with the right surface. The ``void" is the total count of the spaces that remains between the left and right surfaces after theyhave been brought into contact.

The two surfaces are brought into contact by displacing them strictly horizontally towards each other until a rightmost "X" of the left surface of some row is immediately to the left of the leftmost "X" of the right surface of that row. There is no rotation or twisting of these two surfaces as they are brought into contact; they remain rigid, and only move horizontally.

**Note:** The original image may show the two surfaces already in contact, in which case no displacement enters into the contact roughness estimation.

## Input

The input consists of a series of digital images. Each image data set has the following format:

**First line -**

A single unsigned integer, *N*, with value greater than zero (0) and less than 13. The first digit of *N* will be the first character on a line.

**Next *N* lines -**

Each line has exactly 25 characters; one or more X's, then zero or more spaces, then one or more X's.

The end of data is signaled by a null data set having a zero on the first line of an image data set and no further data.

## Output

For each image you receive as a data set, you are to reply with the total void (count of spaces remaining after the surfaces are brought into contact). Use the default output for a single integer on a line.

## Sample Input (character "B" for ease of reading. The actual input file will use the ASCII-space character, not "B").

4

XXXXBBBBBBBBBBBBBBBBXXXXX

XXXBBBBBBBBBBBBBBBXXXXXXX

XXXXXBBBBBBBBBBBBBBBBXXXX

XXBBBBBBBBBBBBBBBBBXXXXXX

2

XXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXXXXXXX

1

XXXXXXXXXBBBBBBBBBBBBBBXX

0

## Sample Output

4

0

0

### 294 - Divisors

### Time limit: 3.000 seconds

|  |
| --- |
| **Divisors** |

Mathematicians love all sorts of odd properties of numbers. For instance, they consider 945 to be an interesting number, since it is the first odd number for which the sum of its divisors is larger than the number itself.

To help them search for interesting numbers, you are to write a program that scans a range of numbers and determines the number that has the largest number of divisors in the range. Unfortunately, the size of the numbers, and the size of the range is such that a too simple-minded approach may take too much time to run. So make sure that your algorithm is clever enough to cope with the largest possible range in just a few seconds.

## Input Specification

The first line of input specifies the number *N* of ranges, and each of the *N* following lines contains a range, consisting of a lower bound *L* and an upper bound *U*, where *L*and *U* are included in the range. *L* and *U* are chosen such thattex2html_wrap_inline42 and tex2html_wrap_inline44 .

## Output Specification

For each range, find the number *P* which has the largest number of divisors (if several numbers tie for first place, select the lowest), and the number of positive divisors *D* of*P* (where *P* is included as a divisor). Print the text 'Between *L* and *H*, *P* has a maximum of *D* divisors.', where *L*, *H*, *P*, and *D* are the numbers as defined above.

## Example input

3

1 10

1000 1000

999999900 1000000000

## Example output

Between 1 and 10, 6 has a maximum of 4 divisors.

Between 1000 and 1000, 1000 has a maximum of 16 divisors.

Between 999999900 and 1000000000, 999999924 has a maximum of 192 divisors.

### 438 - The Circumference of the Circle

### Time limit: 3.000 seconds

|  |
| --- |
| **The Circumference of the Circle** |

To calculate the circumference of a circle seems to be an easy task - provided you know its diameter. But what if you don't?

You are given the cartesian coordinates of three non-collinear points in the plane.

Your job is to calculate the circumference of the unique circle that intersects all three points.

## Input Specification

The input file will contain one or more test cases. Each test case consists of one line containing six real numbers tex2html_wrap_inline26 , representing the coordinates of the three points. The diameter of the circle determined by the three points will never exceed a million. Input is terminated by end of file.

## Output Specification

For each test case, print one line containing one real number telling the circumference of the circle determined by the three points. The circumference is to be printedaccurately rounded to two decimals. The value of tex2html_wrap_inline28 is approximately 3.141592653589793.

## Sample Input

0.0 -0.5 0.5 0.0 0.0 0.5

0.0 0.0 0.0 1.0 1.0 1.0

5.0 5.0 5.0 7.0 4.0 6.0

0.0 0.0 -1.0 7.0 7.0 7.0

50.0 50.0 50.0 70.0 40.0 60.0

0.0 0.0 10.0 0.0 20.0 1.0

0.0 -500000.0 500000.0 0.0 0.0 500000.0

## Sample Output

3.14

4.44

6.28

31.42

62.83

632.24

3141592.65

### 568 - Just the Facts

### Time limit: 3.000 seconds

The expression *N*!, read as ``*N* factorial," denotes the product of the first *N* positive integers, where *N* is nonnegative. So, for example,

|  |  |
| --- | --- |
| *N* | *N*! |
| 0 | 1 |
| 1 | 1 |
| 2 | 2 |
| 3 | 6 |
| 4 | 24 |
| 5 | 120 |
| 10 | 3628800 |

For this problem, you are to write a program that can compute the last non-zero digit of any factorial for ( $0 \le N \le 10000$). For example, if your program is asked to compute the last nonzero digit of 5!, your program should produce ``2" because 5! = 120, and 2 is the last nonzero digit of 120.

## Input

Input to the program is a series of nonnegative integers not exceeding 10000, each on its own line with no other letters, digits or spaces. For each integer *N*, you should read the value and compute the last nonzero digit of *N*!.

## Output

For each integer input, the program should print exactly one line of output. Each line of output should contain the value *N*, right-justified in columns 1 through 5 with leading blanks, not leading zeroes. Columns 6 - 9 must contain `` -> " (space hyphen greater space). Column 10 must contain the single last non-zero digit of *N*!.

## Sample Input

1

2

26

125

3125

9999

## Sample Output

1 -> 1

2 -> 2

26 -> 4

125 -> 8

3125 -> 2

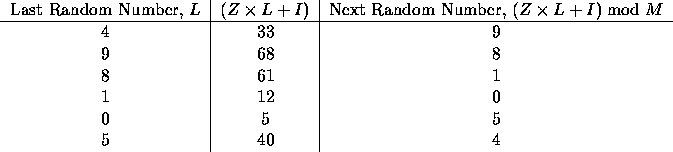
9999 -> 8

### 350 - Pseudo-Random Numbers

### Time limit: 3.000 seconds

Computers normally cannot generate really random numbers, but frequently are used to generate sequences of pseudo-random numbers. These are generated by some algorithm, but appear for all practical purposes to be really random. Random numbers are used in many applications, including simulation.

A common pseudo-random number generation technique is called the linear congruential method. If the last pseudo-random number generated was *L*, then the next number is generated by evaluating ( tex2html_wrap_inline32 , where *Z* is a constant multiplier, *I* is a constant increment, and *M* is a constant modulus. For example, suppose *Z* is 7, *I* is 5, and *M* is 12. If the first random number (usually called the *seed*) is 4, then we can determine the next few pseudo-random numbers are follows:



As you can see, the sequence of pseudo-random numbers generated by this technique repeats after six numbers. It should be clear that the longest sequence that can be generated using this technique is limited by the modulus, *M*.

In this problem you will be given sets of values for *Z*, *I*, *M*, and the seed, *L*. Each of these will have no more than four digits. For each such set of values you are to determine the length of the cycle of pseudo-random numbers that will be generated. But be careful: the cycle might not begin with the seed!

## Input

Each input line will contain four integer values, in order, for *Z*, *I*, *M*, and *L*. The last line will contain four zeroes, and marks the end of the input data. *L* will be less than *M*.

## Output

For each input line, display the case number (they are sequentially numbered, starting with 1) and the length of the sequence of pseudo-random numbers before the sequence is repeated.

## Sample Input

7 5 12 4

5173 3849 3279 1511

9111 5309 6000 1234

1079 2136 9999 1237

0 0 0 0

## Sample Output

Case 1: 6

Case 2: 546

Case 3: 500

Case 4: 220

### 357 - Let Me Count The Ways

### Time limit: 3.000 seconds

|  |
| --- |
| **Let Me Count The Ways** |

After making a purchase at a large department store, Mel's change was 17 cents. He received 1 dime, 1 nickel, and 2 pennies. Later that day, he was shopping at a convenience store. Again his change was 17 cents. This time he received 2 nickels and 7 pennies. He began to wonder ' "How many stores can I shop in and receive 17 cents change in a different configuration of coins? After a suitable mental struggle, he decided the answer was 6. He then challenged you to consider the general problem.

Write a program which will determine the number of different combinations of US coins (penny: 1c, nickel: 5c, dime: 10c, quarter: 25c, half-dollar: 50c) which may be used to produce a given amount of money.

## Input

The input will consist of a set of numbers between 0 and 30000 inclusive, one per line in the input file.

## Output

The output will consist of the appropriate statement from the selection below on a single line in the output file for each input value. The number *m* is the number your program computes, *n* is the input value.

There are *m* ways to produce *n* cents change.

There is only 1 way to produce *n* cents change.

## Sample input

17

11

4

## Sample output

There are 6 ways to produce 17 cents change.

There are 4 ways to produce 11 cents change.

There is only 1 way to produce 4 cents change.

### 371 - Ackermann Functions

### Time limit: 3.000 seconds

An Ackermann function has the characteristic that the length of the sequence of numbers generated by the function cannot be computed directly from the input value. One particular integer Ackermann function is the following:

displaymath32

This Ackermann has the characteristic that it eventually converges on 1. A few examples follow in which the starting value is shown in square brackets followed by the sequence of values that are generated, followed by the length of the sequence in curly braces:

[10] 5 16 8 4 2 1 {6}

[13] 40 20 10 5 16 8 4 2 1 {9}

[14] 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 {17}

[19] 58 29 88 44 22 ... 2 1 {20}

[32] 16 8 4 2 1 {5}

[1] 4 2 1 {3}

## Input and Output

Your program is to read in a series of pairs of values that represent the first and last numbers in a closed sequence. For each closed sequence pair determine which value generates the longest series of values before it converges to 1. The largest value in the sequence will not be larger than can be accomodated in a 32-bit Pascal LongInt or C long. The last pair of values will be 0, 0. The output from your program should be as follows:

Between *L* and *H*, *V* generates the longest sequence of *S* values.

Where:

*L* = the lower boundary value in the sequence

*H* = the upper boundary value in the sequence

*V* = the first value that generates the longest sequence, (if two or more values generate the longest sequence then only show the lower value) *S* = the length of the generated sequence.

In the event that two numbers in the interval should both produce equally long sequences, report the first.

## Sample Input

1 20

35 55

0 0

## Sample Output

Between 1 and 20, 18 generates the longest sequence of 20 values.

Between 35 and 55, 54 generates the longest sequence of 112 values.

### 147 - Dollars

### Time limit: 3.000 seconds

|  |
| --- |
| **Dollars** |

New Zealand currency consists of $100, $50, $20, $10, and $5 notes and $2, $1, 50c, 20c, 10c and 5c coins. Write a program that will determine, for any given amount, in how many ways that amount may be made up. Changing the order of listing does not increase the count. Thus 20c may be made up in 4 ways: 1 tex2html_wrap_inline25 20c, 2 tex2html_wrap_inline25 10c, 10c+2 tex2html_wrap_inline255c, and 4 tex2html_wrap_inline25 5c.

## Input

Input will consist of a series of real numbers no greater than $300.00 each on a separate line. Each amount will be valid, that is will be a multiple of 5c. The file will be terminated by a line containing zero (0.00).

## Output

Output will consist of a line for each of the amounts in the input, each line consisting of the amount of money (with two decimal places and right justified in a field of width 6), followed by the number of ways in which that amount may be made up, right justified in a field of width 17.

## Sample input

0.20

2.00

0.00

## Sample output

0.20 4

2.00 293

### 573 - The Snail

### Time limit: 3.000 seconds

A snail is at the bottom of a 6-foot well and wants to climb to the top. The snail can climb 3 feet while the sun is up, but slides down 1 foot at night while sleeping. The snail has a fatigue factor of 10%, which means that on each successive day the snail climbs 10% $\times$ 3 = 0.3 feet less than it did the previous day. (The distance lost to fatigue is always 10% of the first day's climbing distance.) On what day does the snail leave the well, i.e., what is the first day during which the snail's height exceeds 6 feet? (A day consists of a period of sunlight followed by a period of darkness.) As you can see from the following table, the snail leaves the well during the third day.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Day | Initial Height | Distance Climbed | Height After Climbing | Height After Sliding |
| 1 | 0' | 3' | 3' | 2' |
| 2 | 2' | 2.7' | 4.7' | 3.7' |
| 3 | 3.7' | 2.4' | 6.1' | - |

Your job is to solve this problem in general. Depending on the parameters of the problem, the snail will eventually either leave the well or slide back to the bottom of the well. (In other words, the snail's height will exceed the height of the well or become negative.) You must find out which happens first and on what day.

## Input

The input file contains one or more test cases, each on a line by itself. Each line contains four integers *H*, *U*, *D*, and *F*, separated by a single space. If *H* = 0 it signals the end of the input; otherwise, all four numbers will be between 1 and 100, inclusive.*H* is the height of the well in feet, *U* is the distance in feet that the snail can climb during the day, *D* is the distance in feet that the snail slides down during the night, and *F* is the fatigue factor expressed as a percentage. The snail never climbs a negative distance. If the fatigue factor drops the snail's climbing distance below zero, the snail does not climb at all that day. Regardless of how far the snail climbed, it always slides *D* feet at night.

## Output  For each test case, output a line indicating whether the snail succeeded (left the well) or failed (slid back to the bottom) and on what day. Format the output exactly as shown in the example.

## Sample Input

6 3 1 10

10 2 1 50

50 5 3 14

50 6 4 1

50 6 3 1

1 1 1 1

0 0 0 0

## Sample Output

success on day 3

failure on day 4

failure on day 7

failure on day 68

success on day 20

failure on day 2

### 107 - The Cat in the Hat

### Time limit: 3.000 seconds

|  |
| --- |
| **The Cat in the Hat** |

## Background

(An homage to Theodore Seuss Geisel)

The Cat in the Hat is a nasty creature,  
But the striped hat he is wearing has a rather nifty feature.

With one flick of his wrist he pops his top off.

Do you know what's inside that Cat's hat?  
A bunch of small cats, each with its own striped hat.

Each little cat does the same as line three,  
All except the littlest ones, who just say ``Why me?''

Because the littlest cats have to clean all the grime,  
And they're tired of doing it time after time!

## The Problem

A clever cat walks into a messy room which he needs to clean. Instead of doing the work alone, it decides to have its helper cats do the work. It keeps its (smaller) helper cats inside its hat. Each helper cat also has helper cats in its own hat, and so on. Eventually, the cats reach a smallest size. These smallest cats have no additional cats in their hats. These unfortunate smallest cats have to do the cleaning.

The number of cats inside each (non-smallest) cat's hat is a constant, *N*. The height of these cats-in-a-hat is tex2html_wrap_inline35 times the height of the cat whose hat they are in.

The smallest cats are of height one;   
these are the cats that get the work done.

All heights are positive integers.

Given the height of the initial cat and the number of worker cats (of height one), find the number of cats that are not doing any work (cats of height greater than one) and also determine the sum of all the cats' heights (the height of a stack of all cats standing one on top of another).

## The Input

The input consists of a sequence of cat-in-hat specifications. Each specification is a single line consisting of two positive integers, separated by white space. The first integer is the height of the initial cat, and the second integer is the number of worker cats.

A pair of 0's on a line indicates the end of input.

## The Output

For each input line (cat-in-hat specification), print the number of cats that are not working, followed by a space, followed by the height of the stack of cats. There should be one output line for each input line other than the ``0 0'' that terminates input.

## Sample Input

216 125

5764801 1679616

0 0

## Sample Output

31 671

335923 30275911

### 116 - Unidirectional TSP

### Time limit: 3.000 seconds

|  |
| --- |
| **Unidirectional TSP** |

## Background

Problems that require minimum paths through some domain appear in many different areas of computer science. For example, one of the constraints in VLSI routing problems is minimizing wire length. The Traveling Salesperson Problem (TSP) -- finding whether all the cities in a salesperson's route can be visited exactly once with a specified limit on travel time -- is one of the canonical examples of an NP-complete problem; solutions appear to require an inordinate amount of time to generate, but are simple to check.

This problem deals with finding a minimal path through a grid of points while traveling only from left to right.

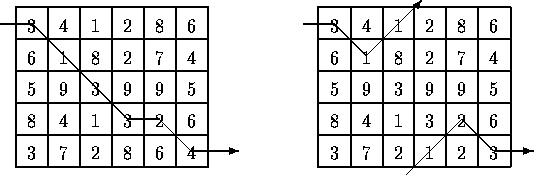
## The Problem

Given an tex2html_wrap_inline352 matrix of integers, you are to write a program that computes a path of minimal weight. A path starts anywhere in column 1 (the first column) and consists of a sequence of steps terminating in column *n* (the last column). A step consists of traveling from column *i* to column *i*+1 in an adjacent (horizontal or diagonal) row. The first and last rows (rows 1 and *m*) of a matrix are considered adjacent, i.e., the matrix ``wraps'' so that it represents a horizontal cylinder. Legal steps are illustrated below.



The weight of a path is the sum of the integers in each of the *n* cells of the matrix that are visited.

For example, two slightly different tex2html_wrap_inline366 matrices are shown below (the only difference is the numbers in the bottom row).



The minimal path is illustrated for each matrix. Note that the path for the matrix on the right takes advantage of the adjacency property of the first and last rows.

## The Input

The input consists of a sequence of matrix specifications. Each matrix specification consists of the row and column dimensions in that order on a line followed by tex2html_wrap_inline376integers where *m* is the row dimension and *n* is the column dimension. The integers appear in the input in row major order, i.e., the first *n* integers constitute the first row of the matrix, the second *n* integers constitute the second row and so on. The integers on a line will be separated from other integers by one or more spaces. Note: integers are not restricted to being positive. There will be one or more matrix specifications in an input file. Input is terminated by end-of-file.

For each specification the number of rows will be between 1 and 10 inclusive; the number of columns will be between 1 and 100 inclusive. No path's weight will exceed integer values representable using 30 bits.

## The Output

Two lines should be output for each matrix specification in the input file, the first line represents a minimal-weight path, and the second line is the cost of a minimal path. The path consists of a sequence of *n* integers (separated by one or more spaces) representing the rows that constitute the minimal path. If there is more than one path of minimal weight the path that is *lexicographically* smallest should be output.

## Sample Input

5 6

3 4 1 2 8 6

6 1 8 2 7 4

5 9 3 9 9 5

8 4 1 3 2 6

3 7 2 8 6 4

5 6

3 4 1 2 8 6

6 1 8 2 7 4

5 9 3 9 9 5

8 4 1 3 2 6

3 7 2 1 2 3

2 2

9 10 9 10

## Sample Output

1 2 3 4 4 5

16

1 2 1 5 4 5

11

1 1

19

### 352 - The Seasonal War

Time limit: 3.000 seconds

|  |
| --- |
| **The Seasonal War** |

The inhabitants of Tigerville and Elephantville are engaged in a seasonal war. Last month, Elephantville successfully launched and orbited a spy telescope called the Bumble Scope. The purpose of the Bumble Scope was to count the number of War Eagles in Tigerville. The Bumble Scope, however, developed two problems because of poor quality control during its construction. Its primary lens was contaminated with bugs which block part of each image, and its focusing mechanism malfunctioned so that images vary in size and sharpness.

The computer programmers, who must rectify the Bumble Scope's problems are being held hostage in a Programming Contest Hotel in Alaland by elephants dressed like tigers. The Bumble Scope's flawed images are stored by pixel in a file called Bumble.in. Each image is square and each pixel or cell contains either a 0 or a 1. The unique Bumble Scope Camera (BSC) records at each pixel location a 1 if part or all of a war eagle is present and a 0 if any other object, including a bug, is visible. The programmers must assume the following:

**a)**

A war eagle is represented by at least a single binary one.

**b)**

Cells with adjacent sides on common vertices, which contain binary ones, comprise one war eagle. A very large image of one war eagle might contain all ones.

**c)**

Distinct war eagles do not touch one another. This assumption is probably flawed, but the programmers are desperate.

**d)**

There is no wrap-around. Pixels on the bottom are not adjacent to the top and the left is not adjacent to the right (unless, of course, there are only 2 rows or 2 columns)

## Input and Output

Write a program that reads images of pixels from the input file (a text file), correctly counts the number of war eagles in the images and prints the image number and war eagle count for that image on a single line in the output file (also a text file).

Use the format in the sample output. Do this for each image in the input file. Each image will be preceded by a number indicating its square dimension. No dimension will exceed 25.

## Sample input

6

100100

001010

000000

110000

111000

010100

8

01100101

01000001

00011000

00000010

11000011

10100010

10000001

01100000

## Sample output

Image number 1 contains 3 war eagles.

Image number 2 contains 6 war eagles.

### 532 - Dungeon Master

Time limit: 3.000 seconds

|  |
| --- |
| **Dungeon Master** |

You are trapped in a 3D dungeon and need to find the quickest way out! The dungeon is composed of unit cubes which may or may not be filled with rock. It takes one minute to move one unit north, south, east, west, up or down. You cannot move diagonally and the maze is surrounded by solid rock on all sides.

Is an escape possible? If yes, how long will it take?

## Input Specification

The input file consists of a number of dungeons. Each dungeon description starts with a line containing three integers *L*, *R* and *C* (all limited to 30 in size).

*L* is the number of levels making up the dungeon.

*R* and *C* are the number of rows and columns making up the plan of each level.

Then there will follow *L* blocks of *R* lines each containing *C* characters. Each character describes one cell of the dungeon. A cell full of rock is indicated by a `#' and empty cells are represented by a `.'. Your starting position is indicated by `S' and the exit by the letter 'E'. There's a single blank line after each level. Input is terminated by three zeroes for *L*, *R* and *C*.

## Output Specification

Each maze generates one line of output. If it is possible to reach the exit, print a line of the form

Escaped in *x* minute(s).

where *x* is replaced by the shortest time it takes to escape.

If it is not possible to escape, print the line

Trapped!

## Sample Input

3 4 5

S....

.###.

.##..

###.#

#####

#####

##.##

##...

#####

#####

#.###

####E

1 3 3

S##

#E#

###

0 0 0

## Sample Output

Escaped in 11 minute(s).

Trapped!

### 429 - Word Transformation

Time limit: 3.000 seconds

A common word puzzle found in many newspapers and magazines is the word transformation. By taking a starting word and successively altering a single letter to make a new word, one can build a sequence of words which changes the original word to a given end word. For instance, the word ``spice'' can be transformed in four steps to the word ``stock'' according to the following sequence: spice, slice, slick, stick, stock. Each successive word differs from the previous word in only a single character position while the word length remains the same.

Given a dictionary of words from which to make transformations, plus a list of starting and ending words, your team is to write a program to determine the number of steps in the shortest possible transformation.

## Input and Output

The first line of the input is an integer N, indicating the number of test set that your correct program should test followed by a blank line. Each test set will have two sections. The first section will be the dictionary of available words with one word per line, terminated by a line containing an asterisk (\*) rather than a word. There can be up to 200 words in the dictionary; all words will be alphabetic and in lower case, and no word will be longer than ten characters. Words can appear in the dictionary in any order.

Following the dictionary are pairs of words, one pair per line, with the words in the pair separated by a single space. These pairs represent the starting and ending words in a transformation. All pairs are guaranteed to have a transformation using the dictionary given. The starting and ending words will appear in the dictionary.

Two consecutive input set will separated by a blank line.

The output should contain one line per word pair for each test set, and must include the starting word, the ending word, and the number of steps in the shortest possible transformation, separated by single spaces. Two consecutive output set will be separated by a blank line.

## Sample Input

1

dip

lip

mad

map

maple

may

pad

pip

pod

pop

sap

sip

slice

slick

spice

stick

stock

\*

spice stock

may pod

## Sample Output

spice stock 4

may pod 3

### 558 - Wormholes

Time limit: 3.000 seconds

In the year 2163, wormholes were discovered. A wormhole is a subspace tunnel through space and time connecting two star systems. Wormholes have a few peculiar properties:

* Wormholes are one-way only.
* The time it takes to travel through a wormhole is negligible.
* A wormhole has two end points, each situated in a star system.
* A star system may have more than one wormhole end point within its boundaries.
* For some unknown reason, starting from our solar system, it is always possible to end up in any star system by following a sequence of wormholes (maybe Earth is the centre of the universe).
* Between any pair of star systems, there is at most one wormhole in either direction.
* There are no wormholes with both end points in the same star system.

All wormholes have a constant time difference between their end points. For example, a specific wormhole may cause the person travelling through it to end up 15 years in the future. Another wormhole may cause the person to end up 42 years in the past.  
A brilliant physicist, living on earth, wants to use wormholes to study the Big Bang. Since warp drive has not been invented yet, it is not possible for her to travel from one star system to another one directly. This can be done using wormholes, of course.  
The scientist wants to reach a cycle of wormholes somewhere in the universe that causes her to end up in the past. By travelling along this cycle a lot of times, the scientist is able to go back as far in time as necessary to reach the beginning of the universe and see the Big Bang with her own eyes. Write a program to find out whether such a cycle exists.

## Input

The input file starts with a line containing the number of cases *c* to be analysed. Each case starts with a line with two numbers *n* and *m* . These indicate the number of star systems ( $1 \le n \le 1000$) and the number of wormholes ( $0 \le m \le 2000$) . The star systems are numbered from 0 (our solar system) through *n*-1 . For each wormhole a line containing three integer numbers *x*, *y* and *t* is given. These numbers indicate that this wormhole allows someone to travel from the star system numbered *x* to the star system numbered *y*, thereby ending up *t* ( $-1000 \le t \le 1000$) years in the future.

## Output

The output consists of *c* lines, one line for each case, containing the word possible if it is indeed possible to go back in time indefinitely, or not possible if this is not possible with the given set of star systems and wormholes.

## Sample Input

2

3 3

0 1 1000

1 2 15

2 1 -42

4 4

0 1 10

1 2 20

2 3 30

3 0 -60

## Sample Output

possible

not possible

### 167 - The Sultan's Successors

Time limit: 3.000 seconds

|  |
| --- |
| **The Sultan's Successors** |

The Sultan of Nubia has no children, so she has decided that the country will be split into up to *k* separate parts on her death and each part will be inherited by whoever performs best at some test. It is possible for any individual to inherit more than one or indeed all of the portions. To ensure that only highly intelligent people eventually become her successors, the Sultan has devised an ingenious test. In a large hall filled with the splash of fountains and the delicate scent of incense have been placed *k*chessboards. Each chessboard has numbers in the range 1 to 99 written on each square and is supplied with 8 jewelled chess queens. The task facing each potential successor is to place the 8 queens on the chess board in such a way that no queen threatens another one, and so that the numbers on the squares thus selected sum to a number at least as high as one already chosen by the Sultan. (For those unfamiliar with the rules of chess, this implies that each row and column of the board contains exactly one queen, and each diagonal contains no more than one.)

Write a program that will read in the number and details of the chessboards and determine the highest scores possible for each board under these conditions. (You know that the Sultan is both a good chess player and a good mathematician and you suspect that her score is the best attainable.)

## Input

Input will consist of *k* (the number of boards), on a line by itself, followed by *k* sets of 64 numbers, each set consisting of eight lines of eight numbers. Each number will be a positive integer less than 100. There will never be more than 20 boards.

## Output

Output will consist of *k* numbers consisting of your *k* scores, each score on a line by itself and right justified in a field 5 characters wide.

## Sample input

1

1 2 3 4 5 6 7 8

9 10 11 12 13 14 15 16

17 18 19 20 21 22 23 24

25 26 27 28 29 30 31 32

33 34 35 36 37 38 39 40

41 42 43 44 45 46 47 48

48 50 51 52 53 54 55 56

57 58 59 60 61 62 63 64

## Sample output

260

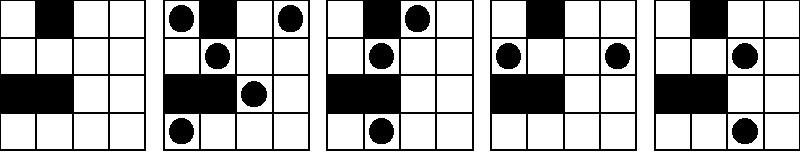
### 639 - Don't Get Rooked

Time limit: 3.000 seconds

|  |
| --- |
| **Don't Get Rooked** |

In chess, the rook is a piece that can move any number of squares vertically or horizontally. In this problem we will consider small chess boards (at most 4$\times$4) that can also contain walls through which rooks cannot move. The goal is to place as many rooks on a board as possible so that no two can capture each other. A configuration of rooks is ***legal*** provided that no two rooks are on the same horizontal row or vertical column unless there is at least one wall separating them.

The following image shows five pictures of the same board. The first picture is the empty board, the second and third pictures show legal configurations, and the fourth and fifth pictures show illegal configurations. For this board, the maximum number of rooks in a legal configuration is 5; the second picture shows one way to do it, but there are several other ways.



Your task is to write a program that, given a description of a board, calculates the maximum number of rooks that can be placed on the board in a legal configuration.

## Input

The input file contains one or more board descriptions, followed by a line containing the number 0 that signals the end of the file. Each board description begins with a line containing a positive integer ***n*** that is the size of the board; ***n*** will be at most 4. The next***n*** lines each describe one row of the board, with a `.' indicating an open space and an uppercase `X' indicating a wall. There are no spaces in the input file.

## Output

For each test case, output one line containing the maximum number of rooks that can be placed on the board in a legal configuration.

## Sample Input

4

.X..

....

XX..

....

2

XX

.X

3

.X.

X.X

.X.

3

...

.XX

.XX

4

....

....

....

....

0

## Sample Output

5

1

5

2

4

Miguel A. Revilla   
2000-01-17

### 725 - Division

Time limit: 3.000 seconds

|  |
| --- |
| **Division** |

Write a program that finds and displays all pairs of 5-digit numbers that between them use the digits 0 through 9 once each, such that the first number divided by the second is equal to an integer *N*, where $2
\le N \le 79$. That is,

abcde / fghij = *N*

where each letter represents a different digit. The first digit of one of the numerals is allowed to be zero.

## Input

Each line of the input file consists of a valid integer *N*. An input of zero is to terminate the program.

## Output

Your program have to display ALL qualifying pairs of numerals, sorted by increasing numerator (and, of course, denominator).

Your output should be in the following general form:

xxxxx / xxxxx = *N*

xxxxx / xxxxx = *N*

.

.

In case there are no pairs of numerals satisfying the condition, you must write ``There are no solutions for *N*.". Separate the output for two different values of *N* by a blank line.

## Sample Input

61

62

0

## Sample Output

There are no solutions for 61.

79546 / 01283 = 62

94736 / 01528 = 62

### 750 - 8 Queens Chess Problem

Time limit: 3.000 seconds

|  |
| --- |
| **8 Queens Chess Problem** |

In chess it is possible to place eight queens on the board so that no one queen can be taken by any other. Write a program that will determine all such possible arrangements for eight queens given the initial position of one of the queens.

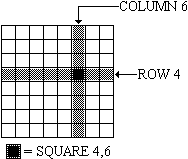
Do not attempt to write a program which evaluates every possible 8 configuration of 8 queens placed on the board. This would require 88 evaluations and would bring the system to its knees. There will be a reasonable run time constraint placed on your program.

## Input

The first line of the input contains the number of datasets, and it's followed by a blank line. Each dataset will be two numbers separated by a blank. The numbers represent the square on which one of the eight queens must be positioned. A valid square will be represented; it will not be necessary to validate the input.

To standardize our notation, assume that the upper left-most corner of the board is position (1,1). Rows run horizontally and the top row is row 1. Columns are vertical and column 1 is the left-most column. Any reference to a square is by row then column; thus square (4,6) means row 4, column 6.

Each dataset is separated by a blank line.



## Output

Output for each dataset will consist of a one-line-per-solution representation.

Each solution will be sequentially numbered $1 \dots N$. Each solution will consist of 8 numbers. Each of the 8 numbers will be the ROW coordinate for that solution. The column coordinate will be indicated by the order in which the 8 numbers are printed. That is, the first number represents the ROW in which the queen is positioned in column 1; the second number represents the ROW in which the queen is positioned in column 2, and so on.

The sample input below produces 4 solutions. The full 8$\times$8 representation of each solution is shown below.

**DO NOT SUBMIT THE BOARD MATRICES AS PART OF YOUR SOLUTION!**

SOLUTION 1 SOLUTION 2 SOLUTION 3 SOLUTION 4

1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0

0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0

0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0

0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0

0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0

0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0

Submit only the one-line, 8 digit representation of each solution as described earlier. Solution #1 below indicates that there is a queen at Row 1, Column 1; Row 5, Column 2; Row 8, Column 3; Row 6, Column 4; Row 3,Column 5; ... Row 4, Column 8.

Include the two lines of column headings as shown below in the sample output and print the solutions in lexicographical order.

Print a blank line between datasets.

## Sample Input

1

1 1

## Sample Output

SOLN COLUMN

# 1 2 3 4 5 6 7 8

1 1 5 8 6 3 7 2 4

2 1 6 8 3 7 4 2 5

3 1 7 4 6 8 2 5 3

4 1 7 5 8 2 4 6 3

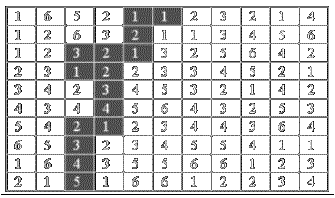
### 868 - Numerical Maze

Time limit: 3.000 seconds

**Numerical maze**

**Background**

   No one knows who is the creator of the first maze (or labyrinth), but all sorts of mazes can be found almost everywhere and some of them have been built long time ago. The most famous maze is maybe the one in Crete, drawn and built by Daedalus for the king Minos, a place where it is believed the Minotaur (a monster with the head of a bull and the body of a man) was kept, feeding on human flesh, until destroyed by Theseus.  
   The following figure illustrates a kind of maze.

  
**Figure 3.1** - The maze

**Problem**

   Are you able to find a path that takes you from the start to exit of the maze? You are only allowed to move horizontally or vertically diagonal movements are not permitted. The path consists of subsequences obeying to the following rule: **1; 1,2; 1,2,3; 1,2,3,4;**and so on. Subsequences may include changes of direction.  
   The problem you have to solve is to determine an entry point and a path that takes you to the exit point, for a given maze. The start point is always a cell in the top row of the maze (with the value 1!) and the exit point is always a cell in the last line of the maze.

**Input**

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

   The first input line contains two positive integers **N** and **M** for the number of rows and columns of the maze, respectively. Each of the N subsequent lines contains the M cell values, separated by single spaces. Cell values are greater or equal to 1.

**Output**

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

   Two lines, the first with the coordinates, row and column, of the starting cell, and the second with the coordinates of the exit cell. If there are several solutions, print the one with lexicographically smallest starting point. If still a tie, print the one with lexicographically smallest ending point.

**Sample Input**  
1  
  
10 11  
1 6 5 2 1 1 2 3 2 1 4  
1 2 6 3 2 1 1 3 4 5 6  
1 2 3 2 1 3 2 5 6 4 2  
2 3 1 2 2 3 3 4 5 2 1  
3 4 2 3 4 5 3 2 1 4 2  
4 3 4 4 5 6 4 3 2 5 3  
5 4 2 1 2 3 4 4 3 6 4  
6 5 3 2 3 4 5 5 4 1 1  
1 6 4 3 5 5 6 6 1 2 3  
2 1 5 1 6 6 1 2 2 3 4

**Sample Output**  
1 6  
10 3

### 735 - Dart-a-Mania

Time limit: 3.000 seconds

## Description of the Game

The game of darts has many variations. One such variation is the game of **301**. In the game of 301 each player starts with a score of 301 (hence the name). Each player, in turn, throws three darts to score points which are subtracted from the player's current score. For instance, if a player has a current score of 272 and scores 55 points with the three darts, the new score would be 217. Each dart that is tossed may strike regions on the dartboard that are numbered between 1 and 20. (A value of zero indicates that the player either missed the dartboard altogether or elected to not throw the dart.) A dart that strikes one of these regions will either score the number printed on the dartboard, double the number printed, or triple the number printed. For example, a player may score 17, 34, or 51 points with a toss of one dart that hits one of the regions marked with a 17. A third way to score points with one dart is to hit the BULLS EYE which is worth 50 points. (There is no provision for doubling or tripling the bull's eye score.)

The first player to reduce his score to exactly zero wins the game. If a player scores more points than his/her current score, the player is said to have ``busted" and the new score is returned to the last current score.

## Problem Statement

Given a player's current dart score, write a program to calculate all the possible combinations and permutations of scores on throwing three darts that would reduce the player's score to exactly zero (meaning the player won the game). The output of the program should contain the number of combinations and permutations found.

For example, if the player's current score is 2, then there would be two combinations and six permutations. The combinations would be: 1) obtain a score of 2 on any one dart and zero on the other two, and 2) obtain a score of one on two different darts and zero on the third dart. The order in which this is accomplished is not important.

With permutations the order is significant; therefore the six permutations would be as follows:

**Dart 1:**

2 0 0 1 1 0

**Dart 2:**

0 2 0 1 0 1

**Dart 3:**

0 0 2 0 1 1

**Note:** The program doesn't print out the actual permutations and combinations, just the total number of each.

## Input

The input file contains a list of integers (each $\le 999$), one per line, that represent several players' current scores. A value of zero or less will signify the end of the input file.

## Output

For each positive integer in the input file, 2 or 3 line(s) will be written to the output file.

If the score can be reduced to zero, your program should write the lines:

NUMBER OF COMBINATIONS THAT SCORES *x* IS *c*.

NUMBER OF PERMUTATIONS THAT SCORES *x* IS *p*.

where *x* is the value of the player's score while *c* and *p* are the total number of combinations and permutations possible, respectively.  
If it is impossible to reduce the player's score to zero, write the line:

THE SCORE OF *x* CANNOT BE MADE WITH THREE DARTS.

After the line(s) above are printed, your program should write a line of 70 asterisks to separate output for different scores. The message ``END OF OUTPUT" should appear at the end of the output file.

## Sample Input

162

175

2

68

211

114

-100

## Sample Output

NUMBER OF COMBINATIONS THAT SCORES 162 IS 7.

NUMBER OF PERMUTATIONS THAT SCORES 162 IS 28.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

THE SCORE OF 175 CANNOT BE MADE WITH THREE DARTS.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NUMBER OF COMBINATIONS THAT SCORES 2 IS 2.

NUMBER OF PERMUTATIONS THAT SCORES 2 IS 6.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NUMBER OF COMBINATIONS THAT SCORES 68 IS 187.

NUMBER OF PERMUTATIONS THAT SCORES 68 IS 1056.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

THE SCORE OF 211 CANNOT BE MADE WITH THREE DARTS.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NUMBER OF COMBINATIONS THAT SCORES 114 IS 82.

NUMBER OF PERMUTATIONS THAT SCORES 114 IS 445.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

END OF OUTPUT

### 341 - Non-Stop Travel

Time limit: 3.000 seconds

|  |
| --- |
| **Non-Stop Travel** |

David hates to wait at stop signs, yield signs and traffic signals while driving. To minimize this aggravation, he has prepared maps of the various regions in which he frequently drives, and measured the average delay (in seconds) at each of the various intersections in these regions. He wants to find the routes between specified points in these regions which minimize his delay at intersections (regardless of the total distance he has to drive to avoid delays), and has enlisted your assistance in this effort.

## Input

For each region, David provides you with a map. The map data first identifies some number of intersections, *NI*. The regions never include more than 10 intersections. The intersections in each region are numbered sequentially, starting with the number one (1). For each intersection, in turn, the input then specifies the number of streets leading away from the intersection, and for each such street, the number of the intersection to which the street leads, and the average delay, in seconds, that David encounters at that intersection. Following the data for the last intersection in a region there appear the numbers associated with the intersections where David wants to start and end his drive. The entire input consists of a sequence of maps, followed by the single integer zero (0).

## Output

For each region, in order, print a single line of output which contains the region number (they, too, are sequentially numbered, starting with 1), a list of the intersection numbers David will encounter in the route with minimum average delay, and the average number of seconds he will be delayed while travelling this route. A suitable format is shown in the example below.

## Notes

1. There will always be a unique route with the minimum average delay in each region.
2. A street from intersection *I* to intersection *J* is one-way. To represent a two-way street from *I* to *J*, the map must also include a route from intersection *J* to intersection *I*.
3. There will never be more than one route directly from intersection *I* to intersection *J*.

## Example

Suppose David wants to travel from intersection 2 to intersection 4 in the region shown in the following map:

+---------------+ From To Delay

| V 1 3 3

1<------2------>3------>4<------5 1 4 6

| | ^ ^ 2 1 2

| +---------------|-------+ 2 3 7

| | 2 5 6

+-----------------------+ 3 4 5

5 4 7

The input and output for this example is shown as the first case in the Sample Input and Sample Output shown on the next page.

## Sample Input

5

2 3 3 4 6

3 1 2 3 7 5 6

1 4 5

0

1 4 7

2 4

2

1 2 5

1 1 6

1 2

7

4 2 5 3 13 4 8 5 18

2 3 7 6 14

1 6 6

2 3 5 5 9

3 6 2 7 9 4 6

1 7 2

0

1 7

0

## Sample Output

Case 1: Path = 2 1 4; 8 second delay

Case 2: Path = 1 2; 5 second delay

Case 3: Path = 1 2 3 6 7; 20 second delay