ASU Annual Programming Competition 2006
Problem Set

Directions

Please read these directions carefully!

The following pages contain the problem set for the 2006 Arizona State University programming competition. There are ten (10) problems. You have four hours to solve these problems. Some problems are easier than others. You will receive the same credit for solving the easy problems as you will for solving the difficult ones, so it is in your best interest to read through all the problems before you begin solving.

All your solutions will need to read input from a file, and all output should be written to a file. The file names you should use are specified under the problem name. You may assume the input files exist in the directory in which your code is running; you should create output files in that same directory. Pay careful attention to the output-formatting instructions in the problem descriptions.

Good luck!

Problem A: Rename

In MS-DOS there exists a 'rename' command that allows you to change the name of a file. There is an equivalent command in Unix called 'mv'. Both commands take arguments in the same way:

```
rename oldname newname
mv oldname newname
```

However, the two commands treat the wild-card character '*' quite differently. In MS-DOS, you can say:

```
rename old* new*
```

and you will find that any filenames you have that previously began with the three characters 'old' have those characters replaces by 'new'. Try the equivalent under Unix and you will probably get an error message :-('mv' will only take the simple two argument, no wild-card form.

To rectify this discrepancy, your program must convert a 'rename' command in to a series of 'mv's'.

Input

Each dataset starts with a list of filenames. These appear one per line. The list is terminated by a line containing the character '#'. Following the list of filenames is the sequence of 'rename' commands. Each command appears on one line in the form:

```
rename wildfrom wildto
```

from and to will both contain one wild-card character, '*'. After the last 'rename' command will be a line containing only the character '#'. There won't be two files with the same name.

Output

For each rename command in the input, you program should first echo the rename command itself, in the same form as the input:

```
rename wildfrom wildto
```

Following that, your program should output the set of 'mv' commands needed to perform the equivalent renaming. Each 'mv' should appear on its own line in the form:

```
mv from to
```

The order of the mv-commands in the output should be the order of the filenames in the input. Print a blank line after each dataset.

Notes

The real MS-DOS '*' has some odd properties which do not concern us here. For example, an MS-DOS '*' will match at most eight characters, none of which is a period '.'. No such restrictions apply to our idealized '*' which will match any number of any printable character. MS-DOS treats upper and lower case letters the same. Unix treats the two cases as distinct, as should your program. MS-DOS limits filenames to 12 characters, including a '.' fixed at the 9th position. Some versions of Unix limit filenames to 14 characters. This is not a limit your program should assume. Each 'rename' command should be performed on the results of the previous command.

Sample Input

```
abFile001.c
abFile001.cxx
abprog001.c
abfile.c
abFile.c
abFileprog.c
#
rename abFile*.c bprog*.cxx
#
acm.c
#
rename ac*.c ib*.cpp
#
```

Sample Output

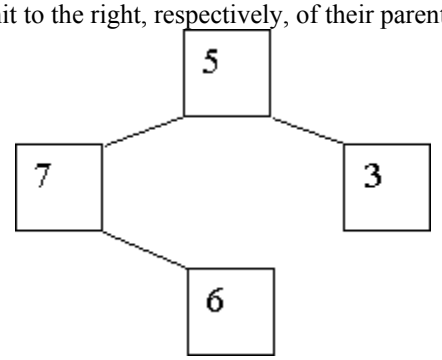
```
rename abFile*.c bprog*.cxx
mv abFile001.c bprog001.cxx
mv abFile.c bprog.cxx
mv abFileprog.c bprogprog.cxx

rename ac*.c ib*.cpp
mv acm.c ibm.cpp
```

Problem B: Falling Leaves

Each year, fall in the North Central region is accompanied by the brilliant colors of the leaves on the trees, followed quickly by the falling leaves accumulating under the trees. If the same thing happened to binary trees, how large would the piles of leaves become?

We assume each node in a binary tree "drops" a number of leaves equal to the integer value stored in that node. We also assume that these leaves drop vertically to the ground (thankfully, there's no wind to blow them around). Finally, we assume that the nodes are positioned horizontally in such a manner that the left and right children of a node are exactly one unit to the left and one unit to the right, respectively, of their parent. Consider the following tree:



The nodes containing 5 and 6 have the same horizontal position (with different vertical positions, of course). The node containing 7 is one unit to the left of those containing 5 and 6, and the node containing 3 is one unit to their right. When the "leaves" drop from these nodes, three piles are created: the leftmost one contains 7 leaves (from the leftmost node), the next contains 11 (from the nodes containing 5 and 6), and the rightmost pile contains 3. (While it is true that only leaf nodes in a tree would logically have leaves, we ignore that in this problem.)

Input

The input contains multiple test cases, each describing a single tree. Each tree appears on a single line. A tree is specified by giving the value in the root node, followed by the description of the left subtree, and then the description of the right subtree. If a subtree is empty, the value -1 is supplied. Thus the tree shown above is specified as 5 7 -1 6 -1 -1 3 -1 -1. Each actual tree node contains a positive, non-zero value. The last test case is followed by a single -1 (which would otherwise represent an empty tree).

Output

For each test case, display the case number (they are numbered sequentially, starting with 1) on a line by itself. On the next line display the number of "leaves" in each pile, from left to right, with a single space separating each value. This display must start in column 1. Follow the output for each case by a blank line. This format is illustrated in the examples below.

Sample Input

```
5 7 -1 6 -1 -1 3 -1 -1
8 2 9 -1 -1 6 5 -1 -1 12 -1 -1 3 7 -1 -1 -1
-1
```

Sample Output

```
Case 1:
7 11 3

Case 2:
9 7 21 15
```

Problem C: Deciphering Messages

The plaintext messages are written in the standard 26-letter English alphabet, A-Z. The encryption is done with the help of a cipher key. This cipher key is composed of an ordered set *L* of English alphabet letters, which is called the alphabetical key, and an integer *N* in the interval [1..25], which is called the numerical key. Each word of *k* characters is encrypted into a word of length between *k* and *k*3* characters. The encryption rules are as follows:

Rule 1:

Each word is encrypted separately; the space between words is preserved. Each word is encrypted from the leftmost to the rightmost letters.

Rule 2:

A plaintext message letter *p* that does not belong to *L* is enciphered as a ciphertext letter *c = ciph(p)* where *ciph* is a function defined below.

Rule 3:

A plaintext message letter *p* that belongs to *L* is enciphered as a string of three letters: the *m*-th letter of *L*, followed by a ciphertext letter *c = ciph(p)*, followed by the (*m* + 1)-th letter of *L*. The value of *m* is incremented by one each time this rule is applied. Arithmetic is performed as if *L* were circular.

For each plaintext message the initial value for *m* is one.

The function ciph translates each letter to the letter *N* letters after it in the alphabet and arithmetic is performed as if the alphabet were circular (for example, if *N* is 2, then *ciph(A) = C*, *ciph(B) = D*,..., *ciph(X) = Z*, *ciph(Y) = A*, *ciph(Z) = B)*.

NOTE: The keys used in the processes of ciphering and deciphering are such that the ciphered text can always be uniquely deciphered (that is, the keys do not contain images of function *ciph* for none of its elements).

Your task is to write a program to decode messages encrypted by using the method just described.

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.

Input data file contains the alphabetical key in the first line, the numerical key in the second line, the number of messages that are to be deciphered in the third line, and a ciphered text message in each of the remaining lines. These text messages only contain uppercase letters in the standard 26-letter English alphabet and, if it is the case, also a space between words. These text messages supposedly result from the encryption of several original plaintext messages using the method described above with the cipher key given in the first and second lines of the input file.

Output

For each test case, the output must follow the description below.

For each of the input ciphered text messages, output of the original plaintext message is required or an error message (‘error in encryption’) in case the input text is not a correct encryption of any plaintext message. The order of the messages must be preserved.

Sample Input

```
1
RSAEIO
2
5
RTSSKAEAGE
GRSCAV
RGSSCAV
RUSIQO
RUSSGAACEV JEGIITOOGR
```

Sample Output

```
RICE
error in encryption
EAT
error in encryption
SEAT HERE
```

Problem D: Ecological Premium

German farmers are given a premium depending on the conditions at their farmyard. Imagine the following simplified regulation: you know the size of each farmer's farmyard in square meters and the number of animals living at it. We won't make a difference between different animals, although this is far from reality. Moreover you have information about the degree the farmer uses environment-friendly equipment and practices, expressed in a single integer greater than zero. The amount of money a farmer receives can be calculated from these parameters as follows. First you need the space a single animal occupies at an average. This value (in square meters) is then multiplied by the parameter that stands for the farmer's environment-friendliness, resulting in the premium a farmer is paid per animal he owns. To compute the final premium of a farmer just multiply this premium per animal with the number of animals the farmer owns.

Input

The first line of input contains a single positive integer *n* (<=20), the number of test cases. Each test case starts with a line containing a single integer *f* (0<f<20), the number of farmers in the test case. This line is followed by one line per farmer containing three positive integers each: the size of the farmyard in square meters, the number of animals he owns and the integer value that expresses the farmer’s environment-friendliness. Input is terminated by end of file. No integer in the input is greater than 100000 or less than 0.

Output

For each test case output one line containing a single integer that holds the summed burden for Germany's budget, which will always be a whole number. Do not output any blank lines.

Sample Input

```
3
5
1 1 1
2 2 2
3 3 3
2 3 4
8 9 2
3
9 1 8
6 12 1
8 1 1
3
10 30 40
9 8 5
100 1000 70
```

Sample Output

```
38
86
7445
```

Problem E: Rat Attack!

Baaaam! Another deadly gas bomb explodes in Manhattan's underworld. Rats have taken over the sewerage and the city council is doing everything to get the rat population under control.

As you know, Manhattan is organized in a regular fashion with streets and avenues arranged like a rectangular grid. Waste water drains run beneath the streets in the same arrangement and the rats have always set up their nests below street intersections. The only viable method to extinguish them is to use gas bombs like the one which has just exploded. However, gas bombs are not only dangerous for rats. The skyscrapers above the explosion point have to be evacuated in advance and so the point of rat attack must be chosen very carefully.

The gas bombs used are built by a company called Wild Critters Slaughtered (WCS) and they are sold under the heading of "smart rat gas". They are smart because — when fired — the gas spreads in a rectangular fashion through the under street canals. The strength of a gas bomb is given by a number d which specifies the rectangular "radius" of the gas diffusion area. For example, Figure 2 shows what happens when a bomb with d = 1 explodes.

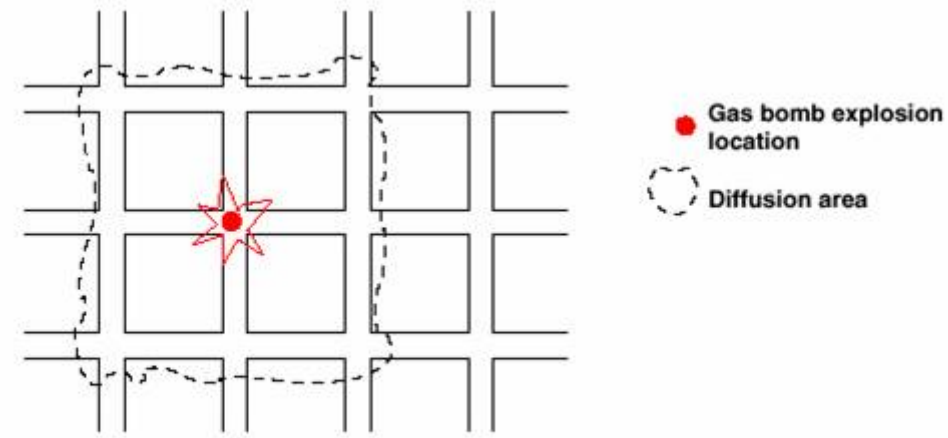


Fig: Rectangular diffusion area of gas bomb

The area of interest consists of a discrete grid of 1025 × 1025 fields. Rat exterminator scouts have given a detailed report on where rat populations of different sizes have built their nests. You are given a gas bomb with strength d and your task is to find an explosion location for this gas bomb which extinguishes the largest number of rats.

The best position is determined by the following criteria:

- The sum of all rat population sizes within the diffusion area of the gas bomb (given by *d*) is maximal.
- If there is more than one of these best positions then the location with the "minimal" position will be chosen. Positions are ordered first by their x coordinate and second by their y coordinate.

Formally, given a location (x1, y1) on the grid, a point (x2, y2) is within the diffusion area of a gas bomb with strength d if the following equation holds:

```
max(abs(x2 - x1), abs (y2 - y1)) <= d
```

Input

The first line contains the number of scenarios in the input.

For each scenario the first line contains the strength d of the gas bomb in the scenario (1 <= d <= 50). The second line contains the number n (1 <= n <= 20000) of rat populations. Then for every rat population follows a line containing three integers separated by spaces for the position (x, y) and "size" i of the population (1 <= i <= 255). It is guaranteed that position coordinates are valid (i.e., in the range between 0 and 1024) and no position is given more than once.

Output

For every problem print a line containing the x and y coordinate of the chosen location for the gas bomb, followed by the sum of the rat population sizes which will be extinguished. The three numbers must be separated by a space.

Sample Input

```
1
1
2
4 4 10
6 6 20
```

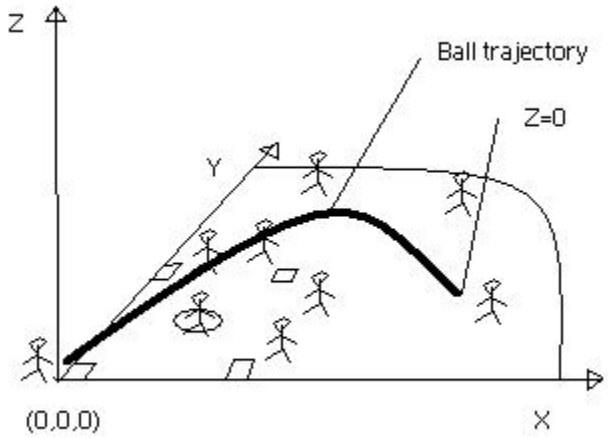
Sample Output

```
5 5 30
```

Problem F: Play Ball!

While on a trip to the US, you decide to stop by Atlanta and get to the baseball stadium where the Braves are playing a decisive baseball game against the NY Mets.

According to the players' position on the field, they may catch the balls that are hit by the batters, if they can get there on time before the ball touches the ground. The ball's position follows the following simple mathematical equations:



For the ball:
$z = at2+bt + c$
- z is the height of the ball above the ground.

$x = dt+e$
- x is the position along the x-axis.

$y = ft+g$
- y is the position along the y-axis.
- a,b,..,g are constants given for each ball hit.
- t is the time elapsed since the ball was hit.

Each player has an individual starting position (x_o, y_o) he always returns to before a new ball is hit. The players, each starting to run as soon as the ball is hit towards the correct position where the ball will eventually touch the ground, all theoretically have a chance to catch the ball. So you must verify all of them since the distance they can run is dependant on their individual speed. Well, after all, we assume that these players always run as fast as possible given the good paychecks they are given, so the variation of their position is always the same throughout the match, e.g. they run at the same speed.

Important notes:

- A ball that ends up touching the ground n the negative side of the ground axis (x or y) is said to be “foul” and can not be caught. These cases are trivial but must be taken care of.
- The time the ball takes to travel (t) is the smallest integer that satisfies the equation of the height of the ball, i.e. t is the first positive integer for which z becomes zero or negative. This integer is the time when the ball is considered to hit the ground.
- A ball "not caught" before it touches the ground is said to be “safe”.
- The maximum number of players is less than 20 players, but could be any number below that limit.

Input

The format of the input is the following (where all parameters are integer values):

#	// number of players/lines to read
x y v	// x,y - startup position of the player; v - the speed of the player
...	// for each player
#	// Number of balls that are hit
a b c d e f g	// Parameters according to the ball equations
...	// for each ball

Output

The output should consist of one line per ball following the format:
"Ball i was ??? at (xkor,ykor)", where i is the ball number, ??? is one of caught, safe, or foul, and (xkor,ykor) is the coordinates where the ball hit the ground. Look at the example below!

Sample Input

```
9
0 0 3
16 16 2
30 0 4
36 24 4
0 30 3
24 36 5
24 75 7
75 24 6
60 60 7
12
-1 3 1 21 3 14 5
-1 1 1 11 3 2 5
-3 4 0 8 5 11 2
-1 2 1 -4 1 6 7
-1 1 0 9 3 14 5
-2 4 1 32 3 12 5
-1 1 1 11 1 19 5
-4 5 0 18 5 20 5
-1 2 1 24 5 6 7
-1 1 0 9 3 4 1
-2 2 1 2 5 22 5
-1 6 1 10 5 13 5
```

Sample Output

```
Ball 1 was caught at (87,61)
Ball 2 was safe at (25,9)
Ball 3 was safe at (21,24)
Ball 4 was foul at (-11,25)
Ball 5 was safe at (12,19)
Ball 6 was safe at (99,41)
Ball 7 was caught at (23,43)
Ball 8 was safe at (41,45)
Ball 9 was caught at (77,25)
Ball 10 was safe at (12,5)
Ball 11 was safe at (9,49)
Ball 12 was caught at (75,96)
```

Problem G: Send a Table

When participating in programming contests, you sometimes face the following problem: You know how to calculate the output for the given input values, but your algorithm is way too slow to ever pass the time limit. However hard you try, you just can't discover the proper break-off conditions that would bring down the number of iterations to within acceptable limits.

Now if the range of input values is not too big, there is a way out of this. Let your PC rattle for half an our and produce a table of answers for all possible input values, encode this table into a program, submit it to the judge, et voila: Accepted in 0.000 seconds! (Some would argue that this is cheating, but remember: In love and programming contests everything is permitted).

Faced with this problem during one programming contest, Jimmy decided to apply such a 'technique'. But however hard he tried, he wasn't able to squeeze all his pre-calculated values into a program small enough to pass the judge. The situation looked hopeless, until he discovered the following property regarding the answers: the answers where calculated from two integers, but whenever the two input values had a common factor, the answer could be easily derived from the answer for which the input values were divided by that factor. To put it in other words:

Say Jimmy had to calculate a function Answer(x, y) where x and y are both integers in the range [1, N]. When he knows Answer(x, y), he can easily derive Answer(k*x, k*y), where k is any integer from it by applying some simple calculations involving Answer(x, y) and k. For example if N=4, he only needs to know the answers for 11 out of the 16 possible input value combinations: Answer(1, 1), Answer(1, 2), Answer(2, 1), Answer(1, 3), Answer(2, 3), Answer(3, 2), Answer(3, 1), Answer(1, 4), Answer(3, 4), Answer(4, 3) and Answer(4, 1). The other 5 can be derived from them (Answer(2, 2), Answer(3, 3) and Answer(4, 4) from Answer(1, 1), Answer(2, 4) from Answer(1, 2), and Answer(4, 2) from Answer(2, 1)). Note that the function Answer is not symmetric, so Answer(3, 2) can not be derived from Answer(2, 3).

Now what we want you to do is: for any values of N from 1 up to and including 50000, give the number of function Jimmy has to pre-calculate.

Input

The input file contains at most 600 lines of inputs. Each line contains an integer less than 50001 which indicates the value of N. Input is terminated by a line which contains a zero. This line should not be processed.

Output

For each line of input produce one line of output. This line contains an integer which indicates how many values Jimmy has to pre-calculate for a certain value of N.

Sample Input

```
2
5
0
```

Sample Output

```
3
19
```

Problem H: Move the Bishop

Consider you have a chess board with *N*x*N* squares, $1 \leq N \leq 100.000.000$. There is only a piece on the board: the bishop. The position of the bishop is given by a pair of numbers $1 \leq r,c \leq N$; *r* is the row and *c* is the column. Position (1,1) refers to the bottom-left square of the board, while position (*N*,*N*) refers to the up-right square.
The problem consists of calculating the minimum number of movements that the bishop has to do to reach a given square of the board, given the position of the bishop and the position of that square. If this movement is not possible, the output must be: "no move". Don't worry if you don't know how to play chess. The only information you need is: the bishop moves diagonally any number of squares, forwards or backwards as long as its path is not blocked by other pieces.

Input

The input begins with a single integer, *C*, indicating the number of test cases following, each of them as described below.
For each test case, the first line contains an integer $1 \leq T \leq 100$, indicating the number of tests for that case. The second line contains an integer $1 \leq N \leq 100.000.000$, for a chess board with *N*x*N* squares. Then the test lines follow, and for each one, there are four numbers separated by a single space. The first two numbers are the row and column where the bishop is, and the last two numbers are the row and column of the position of the square that the bishop has to reach.

Output

For each test line you should produce an output line. This line just contains a number that indicates the minimum number of movements for the bishop according to positions described in the input line, or the message "no move" if the position is not reachable.

Sample Input

```
2
3
8
3 6 6 3
4 2 2 3
7 2 1 4
2
6
1 2 6 5
2 3 5 1
```

Sample Output

```
1
no move
2
2
no move
```

Problem I: Spreadsheet

In 1979, Dan Bricklin and Bob Frankston wrote VisiCalc, the first spreadsheet application. It became a huge success and, at that time, was the killer application for the Apple II computers. Today, spreadsheets are found on most desktop computers.

The idea behind spreadsheets is very simple, though powerful. A spreadsheet consists of a table where each cell contains either a number or a formula. A formula can compute an expression that depends on the values of other cells. Text and graphics can be added for presentation purposes.

You are to write a very simple spreadsheet application. Your program should accept several spreadsheets. Each cell of the spreadsheet contains either a numeric value (integers only) or a formula, which only support sums. After having computed the values of all formulas, your program should output the resulting spreadsheet where all formulas have been replaced by their value.

A1	B1	C1	D1	E1	F1	...
A2	B2	C2	D2	E2	F2	...
A3	B3	C3	D3	E3	F3	...
A4	B4	C4	D4	E4	F4	...
A5	B5	C5	D5	E5	F5	...
A6	B6	C6	D6	E6	F6	...
...	...	...	...	...	...	...

Figure: Naming of the top left cells

Input

The first line of the input file contains the number of spreadsheets to follow. A spreadsheet starts with a line consisting of two integer numbers, separated by a space, giving the number of columns and rows. The following lines of the spreadsheet each contain a row. A row consists of the cells of that row, separated by a single space.

A cell consists either of a numeric integer value or of a formula. A formula starts with an equal sign (=). After that, one or more cell names follow, separated by plus signs (+). The value of such a formula is the sum of all values found in the referenced cells. These cells may again contain a formula. There are no spaces within a formula.

You may safely assume that there are no cyclic dependencies between cells. So each spreadsheet can be fully computed.

The name of a cell consists of one to three letters for the column followed by a number between 1 and 999 (including) for the row. The letters for the column form the following series: A, B, C, ..., Z, AA, AB, AC, ..., AZ, BA, ..., BZ, CA, ..., ZZ. These letters correspond to the number from 1 to 676. The top left cell has the name A1. See figure 1.

Output

The output of your program should have the same format as the input, except that the number of spreadsheets and the number of columns and rows are not repeated. Furthermore, all formulas should be replaced by their value.

Sample Input

```
1
4 3
10 34 37 =A1+B1+C1
40 17 34 =A2+B2+C2
=A1+A2 =B1+B2 =C1+C2 =D1+D2
```

Sample Output

```
10 34 37 81
40 17 34 91
50 51 71 172
```

Problem J: What is the Median?

Medians play an important role in the world of statistics. By definition, it is a value which divides an array into two equal parts. In this problem you are to determine the current median of some long integers.

Suppose, we have five numbers {1,3,6,2,7}. In this case, 3 is the median as it has exactly two numbers on its each side. {1,2} and {6,7}.

If there are even number of values like {1,3,6,2,7,8}, only one value cannot split this array into equal two parts, so we consider the average of the middle values {3,6}. Thus, the median will be (3+6)/2 = 4.5. In this problem, you have to print only the integer part, not the fractional. As a result, according to this problem, the median will be 4!

Input

The input file consists of series of integers X (0 <= X < 2^31) and total number of integers N is less than 10000. The numbers may have leading or trailing spaces.

Output

For each input print the current value of the median.

Sample Input

```
1
3
4
60
70
50
2
```

Sample Output

```
1
2
3
3
4
27
4
```