PROBLEM A **JEAN AND JOE'S CLOTHES** **3 POINTS**

Jean and Joe are a rather untidy couple with a similar bogun taste in clothes. Their 2 children, Jane and James are also clothed in the style.

Jean's mother came to stay and was horrified at the untidy state of the house. She formed a pile of clothes for each of the 4 people in the house, but could only do so by looking at the size of each item. Unfortunately sometimes the size had been cut off or wasn't readable, so these clothes she put in a separate pile.

If the size is 'M' or 'L', then it is Joe's.

If the size is 'S' then it is James'.

If it is 12 or greater then it is Jean's.

If it is smaller than 12 it is Jane's.

Jean's mother makes frequent visits and has to go through this process every visit. Your task in this problem is to write a program that will assist her.

Input consists of data for a number of visits by Jean's mother. The first line of each visit will consist of a whole number, N (0 < N <= 50), which is the total number of clothes found strewn round the house. Then follows N lines each representing the size of an item of clothing, or an 'X' if it is missing or unreadable. The sizes will either be a 2 digit number or a letter 'S', 'M' or 'L'.

Input is finished when the number of clothes strewn, N, is 0. Do not process this line.

Output consists of one line for each visit. The line will contain five numbers, each separated by a space. The numbers represent the number of clothes belonging to Joe, Jean, Jane and James respectively in the current visit. The final number is the number of clothes unable to be assigned to anyone. If there are no clothes in a pile, the number 0 must be shown.

Sample Input

```
8
M
12
X
14
10
L
S
S
0
```

Output for Sample Input

```
2 2 1 2 1
```

Explanation

Joe has 2 (M and L)

Jean has 2 (12 and 14)

Jane has 1 (10)

James has 2 (S and S)

There is one unreadable (X)

PROBLEM B **SEQUENCES** **3 POINTS**

An arithmetic sequence is one in which there is some first number, and then a series of numbers which are all a fixed number different.

For example 3, 5, 7, 9 is an arithmetic sequence that has a first number 3. Then each term after that in the sequence is formed by adding 2 to the previous term. (The terms are different by 2). The 3 is also called the first term (term 1) and 9 is the 4th term.

Given a starting number, a difference and a value, your program is to work out if the number could be part of the sequence. If so, output which term that number would be, and if not, output a letter X.

Input will consist of a number of lines, where each line has 3 numbers separated by spaces.

The first number is an integer that is the first term in the sequence. The second is the difference - this will be a non-zero integer. The third is the value that you will need to test to determine whether it can be part of the sequence or not.

Input is terminated by a zero value for each of the 3 numbers.

Output will consist of one line for each input line. It will consist of either a number indicating which term it is, or X if the number isn't part of the sequence

Sample Input

```
3 2 11

-1 -3 -8

0 0 0
```

Output for Sample Input

```
5

X
```

Explanation

11 is the 5th term

The sequence is –1, -4, -7, -10

(-1+ -3 = -4, -4 + –3 = -7, -7+ -3 = -10)

-8 isn't in the sequenc

PROBLEM C **SARAH'S TOYS** **3 POINTS**

Sarah, aged 5, has lots and lots of stuffed toys. They are supposed to be kept in her bedroom, but do not always make it back each night.

When she gets into bed at night, she puts the stuffed toys that are in her room on the spare bed. She likes to give each stuffed toy a friend for the night and puts them into pairs. Sometimes there isn't an even number, so she might have to make one group of 3.

Your task is to write a program that will report on the arrangement of Sarah's toys on a particular night.

Input will be a number of lines, with each line representing a night in Sarah's house. Each line will have 2 whole numbers, separated by a space. The first number is how many stuffed toys she owns at the time. The second number is the number of toys left round the house that don't make it back to Sarah's bedroom that night.

The last line will be 0 0. Do not process this line.

Output will consist of 2 numbers separated by a space for each night. The first represents how many pairs there are and the second will be either a 0 or a 1. A 0 means there were no groups of 3 and a 1 represents that there was one group of 3.

Sample Input

```
7 3
6 3
5 4
0 0
```

Output for Sample Input

```
2 0
0 1
0 0
```

Explanation

Only 4 toys (7 less 3) are paired, forming 2 pairs

3 toys form 1 group of 3 (and no pairs)

There is only 1 toy

PROBLEM D **DVDS** **3 POINTS**

A local company, “DVDs R Us”, need your help with a stock management system. They sell DVDs on-line from a local warehouse and need to know at any moment how many DVDs of each title they have in stock.

DVDs of a particular title have a stock code and are allocated a certain amount of storeage space in the warehouse. The more popular a DVD title is, the more space is allocated. DVDs are continually being sold and replaced, hence the need for a system to keep track of how many are in stock.

Input consists of data about a number of DVD titles. The data for a DVD title begins with a stock code, a 7 character code consisting of upper case letters and digits only. Input is terminated by a code consisting of a single # - do not process this title.

The next line for each title consists of two integers, M and S, separated by a space. M is the maximum number of DVDs of that title that can be held in stock at any one time (20 <= M <= 500). S is the number of that title currently in stock (0 <= S <= M).

The third line is a single integer, T, the number of transactions to follow (0 < T <= 100). There then follow T lines, each containing details of 1 transaction. A transaction line consists of a single upper case letter (S or R), followed by a space, followed by a positive integer less than 1000.

- If the letter is S it represents a sale, in which case the number shows how many of the DVD have been sold. If the sale is for more than the current stock, only those DVDs currently in stock may be sold.

- If the letter is R it is a restock, so the number shows how many DVDs are added to the current stock. If this would take the number of DVDs in stock to more than the maximum, then the extra items have to be sent back.

Output consists of the DVD's stock code, followed by a space, followed by the number in stock at the end of all the transactions.

Sample Input

```
HG67966

100 64

3

S 10

S 25

R 30

#
```

Output for Sample Input

```
HG67966 59
```

PROBLEM E **NOMOPHOBIA** **10 POINTS**

For Gen Y, digital communication is all encompassing. Extreme stress can be induced in many people by removing a teenager's cell phone.

Ms Hatchett is a teacher who has decided that students will accumulate de-merit points for every cell phone misdemeanor. Confiscation of the miscreant's cell phone occurs after 100 demerit points have been accumulated.

She communicates the following table to her classes

Misdemeanor	Code	De-merit points
Texting while teacher talks	TT	75
Texting during exercises	TX	50
Phone rings	PR	80
Receiving texts	RT	30
Argues about phone use	AP	25
Takes pictures	PX	60

She then records the student's first name and their points over the course of a week. At the end of the week, she announces the scorers of at least 100 points and confiscates their phone over the weekend. Everyone starts the week with a clean slate – that is, points aren't accumulated over more than a week.

Your job is to help her find the owners of phones needing to be confiscated.

Input consists of a number of scenarios, each representing one week. The first line of a scenario consists of 2 whole numbers, W and N. ($0 < W, N <= 50$). W is the week number and N is the number of pupil demerit entries in that week. Input is terminated by a zero value for the week number. Do not process that line.

The first line will be followed by N lines, with each line containing a name (at most 20 characters) and a code as described in the table above. The name and code will be separated by a space.

There may be more than one week's data.

Output will consist of one line for each week. The line starts with the word "Week" separated by a space from the week number followed by the names of the people whose phone's are confiscated. Names must appear in the order that they appear in the input list. These should be commas between the names and a space after the week number. If no phones are confiscated in that week, output a message "No phones confiscated".

Sample Input

```
1 6
Sophie TX
Sophie AP
Matt PR
Sophie PX
Mandy RT
Matt PR
2 3
Matt TX
Jess TT
Lance TX
0 0
```

Output for Sample Input

```
Week 1 Sophie,Matt
Week 2 No phones confiscated
```

PROBLEM F **POSTMAN JOE** **10 POINTS**

Postman Joe is a fitness fanatic. He has a single row of 20 houses on his delivery route, and sets himself a task every day that will require him to walk up and down the row several times. A typical task will look something like this:

3 U3 D2 U1 U6 D4 D5 U7 U9 D5 D3 U5 U4 D2 D5 U8

This means that Joe delivers firstly to house 3, moves 3 houses up the street and delivers to house 6, then moves 2 houses down the street and delivers to house 4, and so on. The houses are numbered sequentially from 1 to 20, house 1 being at the bottom of the street.

Not every house will necessarily receive a delivery each day, but Joe's instructions must never take him to the same house twice, nor take him beyond either end of the row of houses.

To solve this problem, you must write a program to help Joe. It must check that Joe has set himself a legal task and, if so, must report those houses which do not receive a delivery.

Input will consist of a number of tasks that Joe has set, each on a single line. The last line will contain just a # character – that line should not be processed.

Each task represents the deliveries for a single day. Each line starts with a single integer S (1 <=S <= 20) being the first house to which Joe makes a delivery. This is followed by a sequence of letter-number pairs, each separated by a single space. The letter will be U or D, U meaning go up the street (increasing house number), D go down the street. The number will be a single digit integer, stating the number of houses to move.

Output will consist of 1 line for each line of input. If the instructions for the task are legal (ie no house is visited twice, and Joe is not taken beyond either end of the street), output will be a list of houses that do not receive a delivery that day. The list will be in numerical order with a space between each house number. If all houses receive a delivery, output should be the word **none**.

If the instructions for a task are not legal, output should be the word **illegal**.

Sample Input

```
3 U3 D2 U1 U6 D4 D5 U7 U9 D5 D3 U5 U4 D2 D5 U8

8 D7 U5 U6 D9 U2 D4 U9 U3 D2 U7 U2

#
```

Output For Sample Input	**Explanation**
`1 8 14 16`	Joe makes 16 deliveries, missing these 4 houses.
`illegal`	The 7th delivery (D4) is to house 1 which received the 2nd delivery.

PROBLEM G **TARGET** **10 POINTS**

An international newspaper, available in New Zealand, has a letter puzzle called Target. The puzzle presents a grid of 9 letters, from which participants are asked to make words of at least 4 letters. At least one of the words must contain all nine letters. The central letter in the grid must be contained in each word.

M	N	I
O	P	A
C	R	A

In the example shown, all words must contain the letter P. The solution the paper gave for this puzzle is:

apian apron camp campion capon carp corpa corp cramp crampon crimp

crop napa orpin pain pair panic **panoramic** para piano pica pion pram prana

prim prima primo prom ramp rampion romp

In this problem, you will be given a solution and asked to recreate the puzzle. In your answer, the letters of the puzzle must be arranged in alphabetical order except for the central required letter (P in this puzzle) which must be placed 5th in the list (ie in the middle). The puzzle above would be shown as

A A C I **P** M N O R

As stated, the required letter P is placed in the middle.

Input will consist of a number of puzzles. Each puzzle starts with a single integer, N (2 < N <= 50) representing the number of words in the solution to the puzzle. The last line of input is the case where N = 0. This puzzle should not be processed.

Input for each puzzle will contain N words, each on a separate line. Words will be entirely in lower case and will contain between 4 and 9 letters. At least one word in each puzzle will contain 9 letters. All other words in the puzzle will contain only letters found in the 9 letter word, with no letter occurring more times in a single word than it occurs in the 9 letter word. There will be only 1 letter of the alphabet that occurs in every word.

Output will be one line for each puzzle. That line will contain the 9 letters of the puzzle, in **upper** case and separated by spaces, arranged in alphabetical order except that the required letter will be placed in 5th place whatever its alphabetical position.

Turn over for sample input and output.

Sample Input

```
31
apian
apron
camp
campion
capon
carp
corpa
corp
cramp
crampon
crimp
crop
napa
orpin
pain
pair
panic
panoramic
para
piano
pica
pion
pram
prana
prim
prima
primo
prom
ramp
rampion
romp
0
```

Output For Sample Input

```
A A C I P M N O R
```

PROBLEM H **SECRET SANTA** **10 POINTS**

Secret Santa works like this. The people in an organisation (eg an office) each buy one Christmas present of a given value (eg $10). All the names of the people participating are put into a hat, and each person draws out one name. A person is not allowed to draw their own name. Each person then writes on their gift the name of the person drawn from the hat, and the presents are put in a safe place until they are given out. The idea is that nobody knows who bought the present that they receive.

Somebody with too much spare time thought about loops in the giving chain! If you start with one person, move to the person to whom they gave a present, then to the person to whom that person gave a present, you would eventually end up back with the first person – that would be a giving loop! A giving loop could be just 2 people, if A gave to B and B gave to A. On the other hand, there could be a single giving loop containing everybody.

In this problem, you have to write a program to calculate how many giving loops there are in a number of Secret Santa scenarios.

Input consists of a number of scenarios. Each begins with a single integer, N, the number of people taking part (3 <= N <=20). Input ends with a scenario where the number of people taking part is 0 – this should not be processed.

In each scenario, the initial integer is followed by N lines of data. A line consists of the name of a person followed by a space followed by the name of another person. In the N lines of data there will be N distinct names; each name will appear once as the first name on a line and once as the second name. No line will contain the same name twice.

Output will consist of the scenario number followed by a space, followed by the number of giving loops in that scenario.

Sample Input

```
5
Andrew Sally
Chen Andrew
Ahmed Tess
Sally Chen
Tess Ahmed
0
```

Output For Sample Input

```
1 2
```

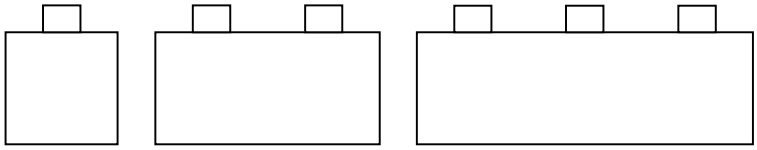
Explanation

The two giving loop are:

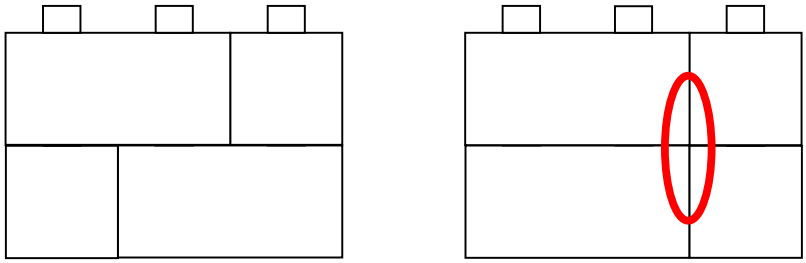
1. Andrew to Sally, Sally to Chen, Chen back to Andrew
2. Ahmed to Tess, Tess back to Ahmed.

Problem IM Lego Bricks 30 Points

You have just been employed by Lego Bricks Corporation and assigned to the team that writes instruction booklets. The senior members of your team design exciting new Lego models, but don't have time to work out all the details. Your job is to check their partial models to see if it is possible to completely build them from the bricks in a given set. A common situation is making flat rectangular walls. This happens so often that there is value in writing a program to do the checking. Your program is check that it is possible to build rectangular walls from a given set of blocks of three kinds: R1, R2, and R3 respectively 1, 2 and 3 units wide.



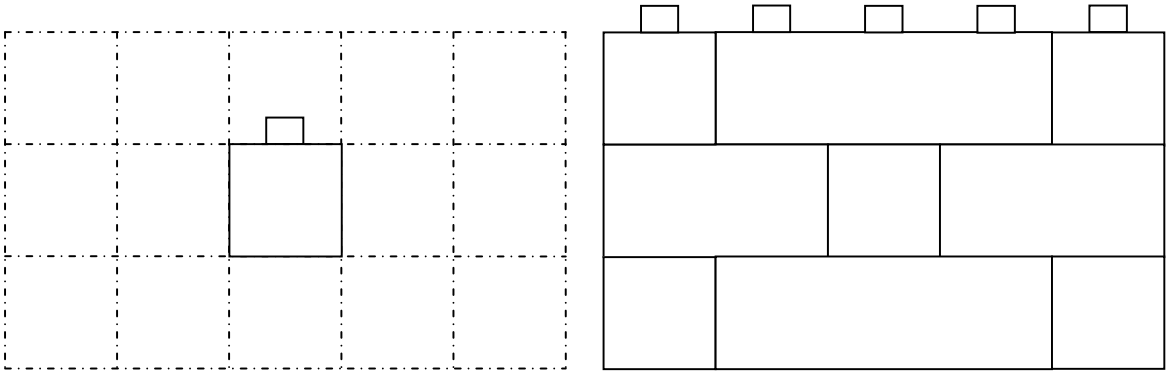
Walls must obey the 'brick' rule – that the vertical joints between bricks must not be aligned. In the diagram below are two 2 by 3 (rows by columns) walls each made from 2 R1 and 2 R2 blocks. The left one obeys the brick rule; the right one does not (the oval marks the illegal vertical alignment).



For each wall you are asked to build you will be given the dimensions (number of rows and columns in Lego units), and also some known block locations that the designer has already determined (these occur as a result of connection of the wall to other parts of a model). When the wall is wide (> 10 columns), the designer is careful to determine most of the bricks along the top row. You will also be given a set of blocks from which the wall must be built.

Notes: You do not need to use all blocks in the set.
The pre-determined blocks are not taken from the set.

Example: Build a 3 row by 5 column wall where there is known to be an R1 block at row 2, column 3. You have a set of 5 R1's, 5 R2's and 5 R3's available. The problem and a solution are as follows. The solution uses 4 R1's, 2 R2's and 2 R3's.



Input Format

Input consists of a number of problems. Each problem starts with a line holding the number of rows and the number of columns. Both lie in the range 1 to 30 (inclusive). 0, 0 signals the end of input. The second line for each problem holds three numbers – being the number of R1's, the number of R2's and the number of R3's. The third line holds the number (N) of predefined blocks. N may be zero. Finally, there are N further lines of input. Each line specifies a predetermined block with a triple of numbers S, R, C. S is the size of the block (1, 2 or three). R and C give row and column numbers of its leftmost column. The top left corner of the wall has R = 1 and C = 1.

Output Format

One line of output per problem. The line should have the string “Yes” if a wall can be build; “No” otherwise.

Sample Input

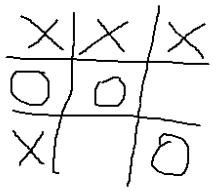
```
2 3
10 10 10
0
3 5
10 10 10
1
1 2 3
2 3
10 0 0
0
0 0
```

Sample Output

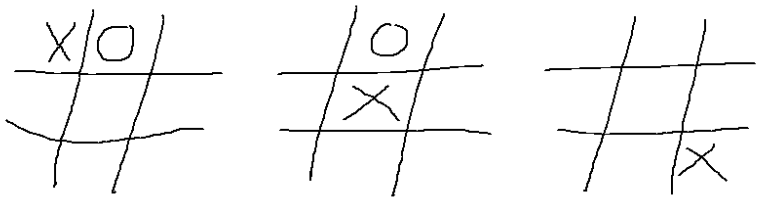
```
Yes
Yes
No
```

Problem J Extreme TTT 30 Points

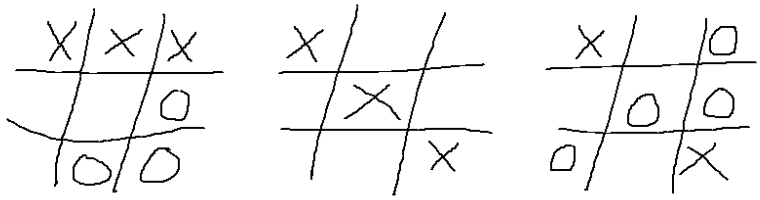
Tic Tac Toe, (Naughts and Crosses) is a simple pencil and paper game played by children. The idea is that players take turns at drawing their symbol (O or X) in squares on a 3 by 3 grid. The first to get three of their symbols in a row (horizontal, vertical or diagonal) wins the game. If the grid is filled without either player making a line, the game is a draw. Here is a game that the X player has just won.



Unfortunately the game is rather limited. An interesting extension is to make it three dimensional – using 3 layers of 3 by 3. Real 3D requires some construction – plastic sets are available commercially. However it is possible to play a 3D game on paper, drawing three 2D grids and imagining them piled above each other. X wins again:



What about playing in 4, 5 or 6 dimensions? This is the basic idea of the game of Extreme Tic Tac Toe (ETTT), a game played by AI computers. ETTT scoring differs from normal Tic Tac Toe. Rather than stop as soon as one player achieves a line of their symbols, ETTT players play until the grid is full, or until they agree to stop. The winner is the one with the greatest number of lines of symbols. By ETTT rules, X would score 4 and O would score 1 in the following 3D game:



Your task is to write a program to read N-Dimensional ETTT game grids and work out the scores.

Input Specification

The input consists of a series of input ETTT board configurations. Each board configuration starts with a line holding N, the dimension of this game (1 <= N <= 10). End of input is signaled by an N value of zero. That is followed by lines of X's, O'x and ~'s for the cells of the game (~ represents an empty cell). Each line holds at least one, and no more than 40 symbols. To understand the order in which data is input, imagine the board being held in an N dimensional array. With N = 5, for example, the board could accessed as cell[a,b,c,d,e] or (cell[a][b][c][d][e] if your language doesn't support the nicer syntax). The following pseudo-code would read the data in the correct order (ignoring line breaks).

```
for a = 1 to 3
   for b = 1 to 3
      for c = 1 to 3
         for d = 1 to 3
            for e = 1 to 3
               read cell[a,b,c,d,e]
```

Output Specification

For each board configuration, output the scores for X and for O as shown in the sample data.

Sample Input

```
2
XXX
OO~
X~O
3
XXX
~~O
~OO
X~~~X~~~X
X~O
~OO
O~X
0
```

Sample Output

```
X scores 1 and O scores 0
X scores 4 and O scores 1
```

Problem K Microspikes 30 Points

In a study of domestic power consumption, researchers built a simulator for New Zealand homes. For each home the software simulates the power consumption of appliances. The goal of the project was to identify microspikes in power usage – short periods of time during which total power consumption rose above a specified limit.

Before a simulation starts all appliances are turned off (using no power). At various times during the simulation period appliances will increase or decrease their power usage. Every time this happens the simulator outputs a record with the appliance number, the time (seconds) since that appliance's last change (or since the start of the simulation), and the change in power level (watts). For a given appliance, records are in order, but unfortunately the simulation was written in such a way that results for different appliances are randomly interleaved. In particular it cannot be assumed that a record for appliance A, written before a record for appliance B, reports an event on appliance A that occurred before B's event.

Your task is to write a program to read files of appliance records, and count the number of microspikes that occur. For any given simulation you will be given a power threshold M and a time threshold S. A microspike occurs if the power level P satisfies P > M for a period of time T: 1 <= T <= S.

Input Specification

Your input is data from a number of simulations. The data for each simulation begins with a line holding three integers (T, M and S), separated by single spaces. T is the total simulation time in seconds (0 <= T <= 100,000); M is the power threshold (0 <= M <= 10^9); and S is the time threshold (1 <= S <= 1000). A line with three zeroes ends the input. Following are N power change lines (0 <= N <= 1,000,000). Each holds three integers (a, t and p), again separated by single spaces: 'a' is the appliance number (1 <= a <= 100,000); 't' is the time in seconds since that appliance's last change (0 <= t <= T); and 'p' is the change in power level (-10,000 <= p <= 10,000). A line with three zeros indicates the end of data for that simulation.

You can assume that the power level of an appliance never goes negative, and that total power consumption never exceeds 1,000,000,000. A microspike is only counted if both increase from below to above the threshold and reduction to below or at the threshold are reported (At the end of simulation appliances may not have been turned off).

Output Specification

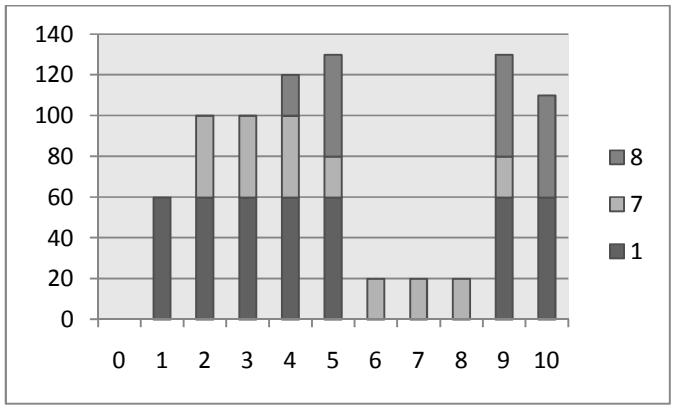
For each simulation one line of output is required, with the number of microspikes observed.

Sample Input (see graph)

```
10 100 2
8 4 20
8 1 30
1 1 60
7 2 40
7 3 -20
1 5 -60
1 3 60
7 5 -20
8 1 -50
8 3 50
0 0 0
0 0 0
```

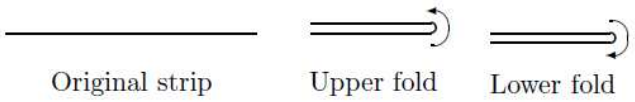
Sample Output

```
1
```



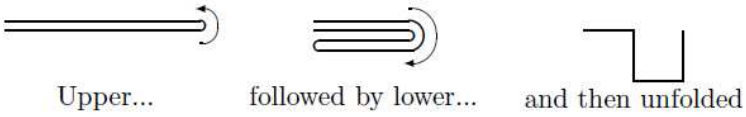
Total power consumption for sample input. Labels on the horizontal axis are the times for the starts of the seconds represented by the bar. All power changes occur instantaneously at the start of a second. Note that the column at time 10 is not part of the simulation. It is included in the chart to show the power level that continues indefinitely after the simulation end.

Suppose you have a strip of paper and are given instructions to fold the paper in one of two ways: an upper fold, where the right end of the paper is brought over to the top of the left end; and a lower fold, where the right end of the paper is brought below the left end. The diagram below illustrates both types of folds.



Original strip Upper fold Lower fold

Now, after meticulously folding the strip several times, you are asked to unfold it by making a 90 degree angle at each crease. The example below shows the result of an upper fold, followed by a lower fold and then an unfolding.



Upper... followed by lower... and then unfolded

If the left end of the folded strip is placed at the origin (0,0) and the first right angle is at (1,0), it is natural to ask the questions: Where will the second right angle be located? The third right angle? Where will the other end of the strip be located? Well, that's for us to know and you to figure out.

Input

The input file will contain multiple test cases. The first line of the file will contain a single integer indicating the number of test cases. Each case will consist of a string of letters U and L indicating a series of upper and lower folds followed by an integer m. The length of the string will be between 1 and 30, inclusive. The value of m identifies a position on the paper. A value of $m = 0$ indicates the left end (at location $(0, 0)$). If there are n folds, then a value of $m = 2^n$ indicates the right end of the strip. Any value for m between these two extremes represents one of the right angles; $m = 1$ indicates the first right angle, and so on.

Output

For each test case, output a single line of the form `(x,y)` indicating the location of the right angle (or end point) specified by the problem. You should assume that if there are n folds in the test case, the length of the string is 2^n so that the distance between creases is 1 unit long.

Sample Input

```
3
UL 4
UL 3
LLUL 13
```

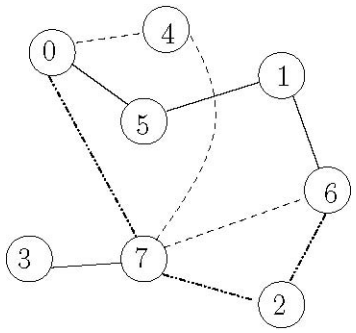
Sample Output

```
(2,0)
(2,-1)
(1,-2)
```

PROBLEM M **CONGESTED NETWORKS** **100 POINTS**

Given a connected computer network (bidirectional communication) we want to find two different nodes u and v such that we can maximize the congestion between u and v with a continuously sent virus being sent between the pair. We define the congestion level as the maximum number of edge-disjoint paths between nodes u and v.

For example, the network shown in the following figure has three different paths between nodes 0 and 6 such that each edge used is only part of one connecting path. Note that two paths are allowed to go through the same node, such as node 7. No other pair of nodes provides a higher congestion level.



Input Specification

We will be given a sequence of connected computer networks each with n nodes, n ≤ 40, labeled {0, 1,...,n − 1}. The last input case will be followed by a network of n =0 nodes, which should not be processed.

The specification for a computer network will be as follows: the first line contains a single non-negative value n, denoting the number of nodes. This is then followed by n lines of integers, separated by spaces, denoting the neighbors (zero indexed) of each node. Expect up to 2000 test cases.

Output Specification

For each input case output one integer on a line by itself denoting the maximum congestion level possible for some pair of its network nodes.

[Turn over for sample data]

Sample Input

```
8
4 5 7
5 6
6 7
7
0 7
0 1
1 2 7
0 2 3 4 6
4
1 2
0 2
3 0 1
2
0
```

Output for Sample Input

```
3
2
```

Problem N Functional Programming Counts 100 Points

The formal methods team at your place of work has decided to experiment with very small functional language. They have, of course, specified it. The bad news is that you have to implement it. The good news is that you can do so in a programming language of your choice and a formal proof of the correctness of your implementation is not required. In the grand tradition of Lisp your implementation will take the form of an interpreter which accepts expressions, assignments and function definitions from the console. Function definitions are remembered and the functions may be used in subsequent expressions. Expressions are immediately evaluated, and the result is displayed at the console.

Input Specification

The input consists of a series of input lines, each holding an input action. No line has more than 100 characters. Actions can be of the following kinds:

1. <expression>
   The expression should be evaluated (to give an integer result) and the result should be displayed.
2. set <identifier> = <expression>
   An assignment statement: The identifier represents a variable. If it has not been seen before, create a new variable and give it the value of the expression, otherwise just update an existing variable.
3. def <identifier> (<parameter>) = <expression>
   A function definition line, that wholly or partially defines a function with name <identifier> and one integer parameter. <parameter> can be a <number> or an <identifier>. If it is a <number> then it defines the function when the parameter takes that value. The order of definition lines is important. All definitions for a function are kept, and function calls check them in order of input for the first parameter match. (This makes it impossible to change the definition of a function.)
4. profile
   The software should report the number of function calls made on each definition line for each defined function, since the last profile request (or the start of the program). See sample output.
5. exit
   The end of input.

Input should be processed, one line at a time, performing the requested action immediately, until the line with 'exit' is reached. The only data type processed by the language is integer. Values lie in the range -1,000,000 to 1,000,000 at all times. (The range is inclusive). The syntax for expressions is as follows. Operator precedence applies in the normal manner – evaluate factors, then terms and finally expressions.

<expression> is a sum(+) or difference(-) of <term>s to be evaluated from left to right (eg: a + b – 3 + 4 – c)

<term> is a product '*', quotient '/' or remainder '%' of <factor>s, to be evaluated left to right. Remainder should never be used with negative numbers. Division is integer division. (eg: 3 * 4 / 5 % x).

<factor> can be a <number>; an <identifier> being a variable, or the parameter of the current function; a function call of the form <identifier>(<expression>); or an expression in parenthesis (<expression>).

<number> is a string of digits, being a decimal value in the range 0 to 1,000,000 (inclusive)

<identifier> is a string of upper and lower case letters – case is significant, eg: 'A' and 'a' are different variables. The following are reserved words and should not be used as identifiers: 'def', 'set', 'profile', 'exit'.

White space should be ignored on all input lines, except in so far as it separates words. You can assume that all input is syntactically correct, that all function calls return (terminate), and that all expressions produce results in the permitted range. The total number of calls profiled on any one function line will never exceed 1,000,000.

Output Specification

Expression results should be reported on a line of their own, prefixed by '>> ' (two greater than signs and a single space).

Profile results should be reported with one line per function of the form '<identifier> calls: n_1 n_2 n_3 … => n_t' where n_i is the number of calls for definition line i in order input, and n_t is the total number of calls to the function (single spaced).

Sample Input

```
2
2 + 5 / 2 * 2
2 - 5 % 2
def fib(1) = 1
def fib(2) = 1
def fib(p) = fib(p-1) + fib(p-2)
fib(1)
profile
fib(2)
profile
fib(3)
profile
fib(5)
profile
set x = 6
x * 4 + fib(x)
def test(1) = 1
def test(2) = 1
def test(p) = (test(p-1) + test(p-2) + 23) % 1000
def test(2) = 10
test(3)
profile
exit
```

Sample Output

```
>> 2
>> 6
>> 1
>> 1
fib calls: 1 0 0 => 1
>> 1
fib calls: 0 1 0 => 1
>> 2
fib calls: 1 1 1 => 3
>> 5
fib calls: 2 3 4 => 9
>> 32
>> 25
fib calls: 3 5 7 => 15
test calls: 1 1 1 0 => 3
```

Problem O Packages Par Avion 100 Points

You have been asked to help manage an airmail package delivery system for a company called Packages Par Avion (PPA). PPA works at a number of airports. At each airport they maintain a reception desk where customers can bring packages. They run a fleet of aircraft flying between the airports. PPA works with a ground transport company called PPT that takes responsibility for delivery of packages to their final destinations. Your task is to write software to determine how packages should be loaded onto planes.

Operations at each airport are as independent of those at other airports as possible. The day to day operation of a PPA branch is very structured. From 9 am to 5 pm each day parcels are accepted at reception. When a customer brings a parcel to reception, it is first weighed. The weight is rounded up to the nearest multiple of 1kg. The customer provides the delivery address and the value of the parcel in dollars. Staff determine the closest airport to the delivery address and record that, along with a time stamp, on the parcel. The time stamp is a real number – recorded in days and fractions of a day. The low precision bits of the time stamp are coded to ensure globally unique times across the company's operation – no two parcels get the same time stamp. The reception area has limited capacity for storing parcels. If accepting a parcel would cause that capacity (sum of rounded parcel weights) to be exceeded, the parcel will be rejected.

At 5.05 pm each day, parcels are moved from reception and added to any already at the loading bay (those waiting from previous days, or parcels at an intermediate point in a multi-stage journey). Next, each branch sends an email message to every other branch stating the total weight (sum of rounded weights) of parcels they have in their loading bays. Planes are then loaded with parcels and fly to their destinations, always arriving in good time to complete unloading before the next day's work begins. After unloading: if parcels are at their final destination, they are passed on to PPT, and can be assumed to be delivered. Otherwise they are held in the loading bay, ready for the next stage of their journey. Note: All airports operate in the same time zone.

Your software must assign parcels to planes. PPA policy is to do this as follows. Firstly a next hop is chosen for each parcel. The hop chosen is the first stop on the route which is a shortest sequence of plane trips – ie: a shortest number of hops. Where there is more than one route of shortest length, choose the one whose first step is to the airport with the lowest weight of parcels currently at their loading bay. If this is still ambiguous chose the first step airport with the lowest airport number. If there is no route to the parcel's destination, it is left in the loading bay. Secondly, load as many parcels as possible on to the planes chosen for the first steps in their routes. Parcels are chosen firstly to maximize the value of cargo on board the plane, and secondly to give priority to the oldest parcels. Of two sets of parcels with equal total value and timestamps {1.4, 1.6, 1.7} and {1.4, 1.5, 1.9} the latter would be chosen. If parcels don't fit on their assigned plane, they are left in the loading bay for the next day.

Input Specification

The input consists of a number of loading problems. The first line of each problem has five integers (A, F, P, B and C) separated by single spaces (all input takes this form – numbers separated by single spaces, no leading or trailing spaces). A is the number of other airports (numbered 1 to A, excluding the one you are working at); F is the number of flights for the day; P is the number of parcels that customers bought in during the day; B is the number of parcels already in the loading bay at the start of the day; and C is the capacity of the reception area. The next A lines give the total weight of parcels at each airport's loading bay. Then F lines describe flights (hops). Each of the flight lines has three integers: the starting airport(s), the destination airport(d), and the capacity(c) of the plane in kg. Your airport is numbered 0 in the flight information. Following that are P lines describing parcels bought in by customers (remember, some may not be accepted). Each parcel line has a timestamp(float t), a rounded weight(integer w), a final destination airport(d) and a value(integer v). These lines are in order of timestamp. Last, there are B lines for the parcels waiting in the loading bay. They take the same form as the P lines. These are also in timestamp order. End of input is indicated by an initial problem line with 5 zeroes. Note that there is at most one flight between any two airports on one day. Limits are as follows: $1 <= A <= 30$; $1 <= F <= 100$; $0 <= P+B <= 5000$; $1 <= C <= 150$

Output Specification.

For each flight leaving from your airport (airport 0), show the total value of parcels loaded. Write one line per flight, with the flight number (0, 1 …) and the total value of parcel loaded. Flights are numbered from 0 in the order input. Lines should be written flight number order.

Sample Input

```
4 6 2 2 20
50
100
100
100
0 3 7
1 3 7
1 4 7
0 1 7
3 4 7
3 2 7
2.5 2 4 2
2.6 5 4 9
1.7 3 4 6
1.8 3 4 6
0 0 0 0 0
```

Sample Output

```
Flight 0 value = 0
Flight 3 value = 12
```

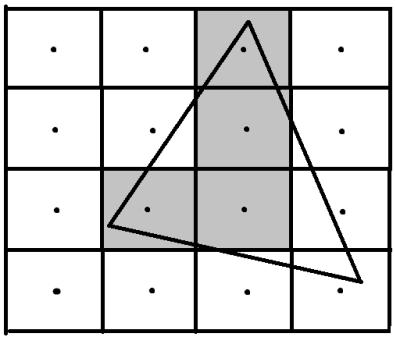
Problem P Ascii Art 100 Points

Modern computers provide high quality graphics systems for games and other (less important) uses. But some computer users choose to forgo this image goodness and work with old fashioned monochrome text-only interfaces. For them Ascii art is the only option. As part of a project to implement Doom in Ascii, you have been asked to build an Ascii renderer.

The input will be a list of triangles and viewing information (as shown in the input section below). For each triangle you will be given the X, Y and Z coordinates of each vertex (in no particular order), and a 'colour'. As the term 'colour' has little meaning in Ascii, we will interpret colour as meaning 'character'. Ie: A triangle provided with the 'colour' 'E' will be rendered as an area of letter E's on the screen.

The viewing information will include a 'screen size' – S – the number of rows and columns of characters that form an image (assume uniform letter spacing). For simplicity you will assume that an S by S block of characters is square - ie: that the 'pixels' are square. In practice the 'images' produced will probably look stretched vertically.

The coordinate system is right handed. Standing on the positive Z axis, looking towards the origin, positive Y is upward and positive X is to your right. You will be given a camera position, from which the scene is viewed; a 'look at' point – the point in the exact centre of the field of view, and an 'up' vector – indicating which way is up (ie: how you are rotating your camera about the line of view). All images will be produced with up being generally in the positive Y direction. You will never be required to produce an image viewing straight up or down.

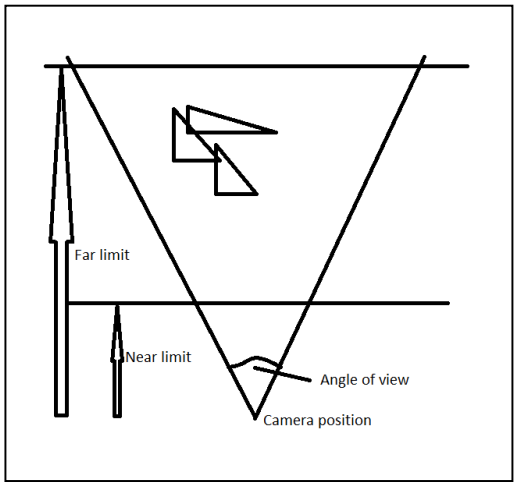


If S is 4 meaning that the screen is a 4 by 4 grid of characters, for example, vertex locations should be calculated as standard precision float values between 0 and 4. Using that measure, pixel centres occur at 0.125, 0.375, etc. A pixel should be 'coloured', if its centre falls inside or on the border of a triangle. See diagram. The final Ascii output for that image (with background as <space> and colour 'E'). is as shown.

```
 E
 E
EE
```

Input Specifications.

You will be asked to render a number of scenes. Each scene starts with a line holding two integers: S and T. S is the screen size (0 <= S <= 40) and T is the number of triangles to be rendered. (0 < T <= 20). S and T values of 0 terminate input. Next are T lines of input, each holding a triangle definition as: a single character (colour – which will not be a white space character); and 9 floats being coordinates of the three vertices of the triangle in order X1 Y1 Z1 X2 Y2 Z2 X3 Y3 Z3. All input items on a line are separated by single spaces. Following is a line with three floats. The X Y Z coordinates of the camera position. Then a line with X Y Z coordinates of the 'look at' point. Finally a last line with perspective data – the angle of view (degrees) and the distance to the near and far viewing limits (see diagram).



Output Specifications

For each problem output a blank line; a line of S asterisks; the rendered image; and another line of S asterisks.

Notes: You can assume that all triangles lie fully inside the viewing volume defined by the angle of view and the near and far viewing limits. Triangle display should respect the fact that near triangles obscure far triangles. Test data has been chosen to avoid exact alignment with pixel centres, so calculation to standard precision float levels should produce consistent results.

Sample Input

```
4 1
E 0.625 0.95 -0.001 0.275 0.325 -0.001 0.875 0.15 -0.001
0.5 0.5 0.5
0.5 0.5 0
90 0.5 1.0
0 0
```

Sample Output

```
****
   E
   E
  EE

****
```