Virginia Tech
ACM Local Contest 2007
Sponsored by Evergrid

Problem Statement

DO NOT OPEN UNTIL PC^2 SAYS THE CONTEST IS STARTED

Problem A
The Great Box Extruder

You have been tasked with programming the the poly-dimensional size allocator of the planning module of the great box extruder. The great box extruder creates boxes in N space dimensions. It is quite impressive. However, some troublesome physics requires that the length each dimension of the box be prime numbers. The target n-dimensional volume of each box are known, for each target volume you must find the prime dimensions that will create a box of the given volume. The volume can be computed by multiplying all the dimensions together.

Ignore the above text and write a prime factorization program. The prime factorization is the list of prime numbers that can be multiplied together to get the given number.

Input

The input will consist of positive integers which are greater than 1, each on its own line. A line containing "0" indicates the end of input.

Output

Print the prime factorization of the number in ascending order.

Sample Input

```
20
1008
13
0
```

Sample Output

```
2 2 5
2 2 2 2 3 3 7
13
```

Problem B
Sudoku Checker

Sudoku?

Sudoku is a number-placement puzzle. In its most common form, shown in the figure below, the player is tasked with filling in the missing numbers in a 9x9 grid. Numbers must be placed such that all 9 rows contain each of the numbers 1-9, all 9 columns contain the numbers 1-9, and all 9 of the 3x3 boxes contain the numbers 1-9. Clearly, no row, column, or box can contain duplicates.

4		6		9				
9		3	7	2				6
		8				9		
2		4		3	1		5	7
	6	1					4	
3	9		6	5		1		2
		5				8		
1				8	6	2		4
				1		7		5

4	2	6	1	9	3	5	7	8
9	5	3	7	2	8	4	1	6
7	1	8	4	6	5	9	2	3
2	8	4	9	3	1	6	5	7
5	6	1	8	7	2	3	4	9
3	9	7	6	5	4	1	8	2
6	3	5	2	4	7	8	9	1
1	7	9	5	8	6	2	3	4
8	4	2	3	1	9	7	6	5

Sudoku is very amenable to automation. To that end, you have decided to do your part to improve the world by writing a program to help combat Sudoku puzzles. Ignoring your colleagues, who have foolishly focused on writing Sudoku solvers, you have decided to direct your energies toward building a Sudoku checker. Now Sudoku players can first guess numbers blindly to fill in the gaps and then use your checker to confirm their results. A puzzle-solving process that used to take minutes is now shaved down to a mere billion minutes (give or take a few a few trillion).

Input

The input will consist of several Sudoku boards. Each board will consist of 9 lines, each containing 9 single digits, each separated by a single space. Only digits in the range 1 through 9, inclusive, should be considered valid. The end of input is indicated by a single blank line.

Ouput

For each Sudoku board, print a single line saying either VALID or INVALID, indicating the validity of the Sudoku board.

Sample Input

```
4 7 9 8 2 1 5 6 3
2 8 5 7 3 6 1 4 9
1 3 6 5 4 9 2 7 8
9 6 4 3 7 2 8 1 5
3 1 7 6 5 8 4 9 2
5 2 8 1 9 4 6 3 7
7 9 1 2 6 5 3 8 4
8 4 2 9 1 3 7 5 6
6 5 3 4 8 7 9 2 1
7 6 8 1 9 5 2 4 3
9 3 4 8 2 6 1 5 7
2 5 1 7 3 4 6 9 8
1 2 6 9 7 8 5 3 4
5 8 7 4 1 5 9 2 6
4 9 3 6 2 3 7 8 1
8 7 5 2 4 1 3 6 9
3 4 9 5 6 7 8 1 2
6 1 2 3 8 9 4 7 5
0 1 2 3 4 5 6 7 8
3 4 5 6 7 8 0 1 2
6 7 8 0 1 2 3 4 5
8 0 1 2 3 4 5 6 7
2 3 4 5 6 7 8 0 1
5 6 7 8 0 1 2 3 4
7 8 0 1 2 3 4 5 6
1 2 3 4 5 6 7 8 0
4 5 6 7 8 0 1 2 3
```

Sample Output

```
VALID
INVALID
INVALID
```

Problem C
Gossipy Children

Sometimes children put on shirts emblazoned with a single capital letter and gather together in a rectangle for no apparent reason. These kinds of children like to share secrets, but they are very sly about their gossip—they only tell other children secrets by whispering in their ears. Because they are so densely packed, they only have room to whisper gossip to fellow children in front of, behind, and on each side of them. Four direct confidantes is the maximum a child can expect during a Rectangle Meeting.

It is important to note that this is simply a maximum, because children have one more restriction that they take into account before spreading gossip: a child will only whisper a secret to another child that is wearing the same letter shirt.

Recently, children have been seen gathering together in ever larger rectangles. Adults are worried about what secrets may be spreading. You have been dispatched to investigate. You already know that N children know a secret. The question is, in which group of children has the secret spread?

You can presume that any secrets that have started spreading have already reached as far as possible in a given Rectangle Meeting. Also, remember that each child can only tell fellow letter-children the secret if they are adjacent; they never attempt to whisper across the pack. So there might be a pocket of children of the letter ‘D’, for example, that know the secret, without other ‘D’ children knowing.

Input

Input will consist of several test cases. The first line of each test case will be 3 space-separated positive integers <**S R C**>:

- **S** is the number of children that know the secret in question
- **R** is the number of rows of children in the rectangle meeting
- **C** is the number of columns of children in the rectangle meeting

The next **R** lines each contain **C** characters (only letters and numbers). These may be capital letters, in which case they describe devious children wearing lettered shirts. These characters may also be lowercase letters or numbers, in which case they do not describe children (perhaps they describe a tree or a large moose that got trapped by the Rectangle). The maximum size of the meeting is 1000 (**R** * **C** <= 400).

The end of input is indicated by a line containing only 3 zeros separated by spaces:
0 0 0

Output

For each test case, print a line consisting of whichever capital letter belongs to the group of children who possess the secret. There will always be exactly one letter that has at least one pocket of children of the correct size in the Rectangle.

Sample Input

```
5  7  6
DHHHHK
DPPPHK
DPPPKK
DPPPKK
mmKKKK
Mmmmmm
mmmmmm
3  4  4
BBAB
BABA
ABAB
BABB
1  10  10
oooooooooo
oooooooooo
oooooooooo
oooooooooo
oooDoooooo
oooooooooo
oooooooooo
oooooooooo
oooooooooo
oooooooooo
0  0  0
```

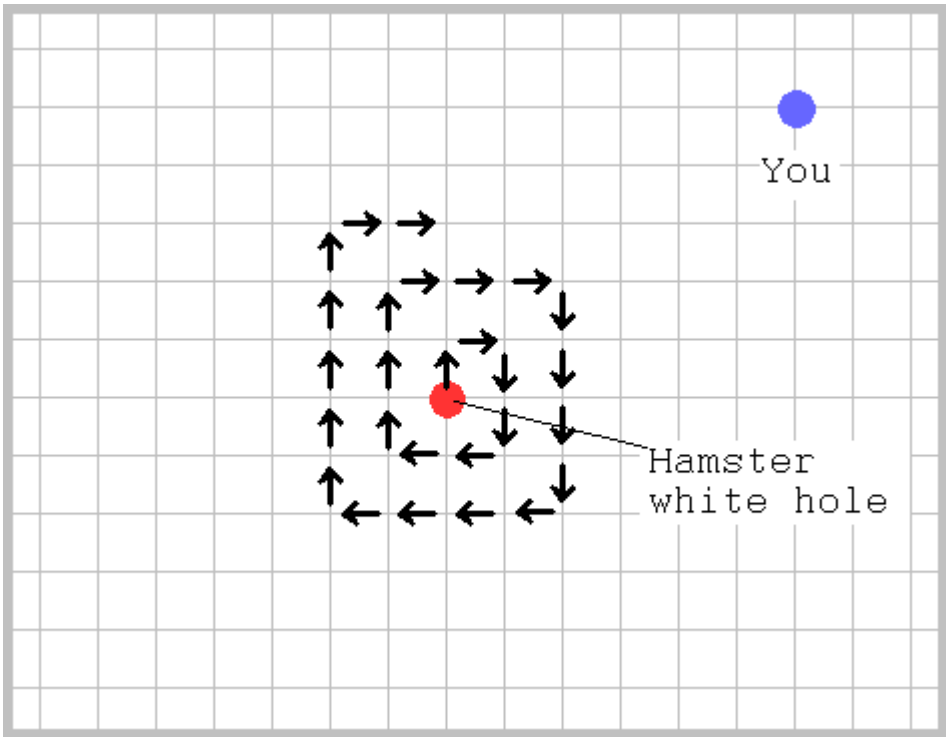
Sample Output

```
H
B
D
```

Problem D
Update on the Inevitable Hamster Invasion

Computer scientists are baffled by a steady stream of hamsters seen emerging out of a portal in Blacksburg, VA. Since Thursday morning, over 1000 hamsters have issued forth from a kind of hamster white hole near the center of the city. The procession of rodents is relentless and synchronized, with a new hamster marching out every “Hamster Period” seconds for no apparent reason.

The hamsters all follow the same orderly route out of their portal, spiraling clockwise around as shown in the figure below, advancing one meter along the path every Hamster Period seconds.



Given that we live in two-dimensions, the hamsters will—inevitably—take over the universe.

You have realized that your best chance for survival is to join the rodent revolution. Thus, you have decided to put your programming skills to the test by calculating when your current position will be overtaken by the onslaught. You are given your current position as (x, y) coordinates in meters (with the hamster portal at the origin) and the Hamster Period. Given the march-path of the hamsters, calculate how many seconds it will be before the horde reaches you.

The hamsters always start their path out of the portal by traveling to position (0, 1) as shown above, then proceeding in a clockwise spiral as shown. They always move along grid lines, eventually covering every set of integer coordinates. At time 0 a single hamster is located at the origin. After the first Hamster Period, there is one hamster at the origin and one hamster at position (0, 1).

Input

The input will consist of several invasion scenarios, each on one line. Each scenario will consist of 3 integers **H X Y**, separated by spaces where:

- **H** is the Hamster-Period in seconds, an integer between 1 and 100. Every **H** seconds, a new hamster emerges at the origin and every other hamster advances one meter along the path.
- **X** and **Y** describe your current position in meters from the origin. **X** and **Y** are integers in the range -600 to 600, inclusive. **X** and **Y** can not both be 0.

The end of input is indicating by a line containing 3 zeros. No output is generated for this line.

Output

For each scenario, print, on a line by itself, the number of seconds until the line of hamsters reach your current position.

Sample Input

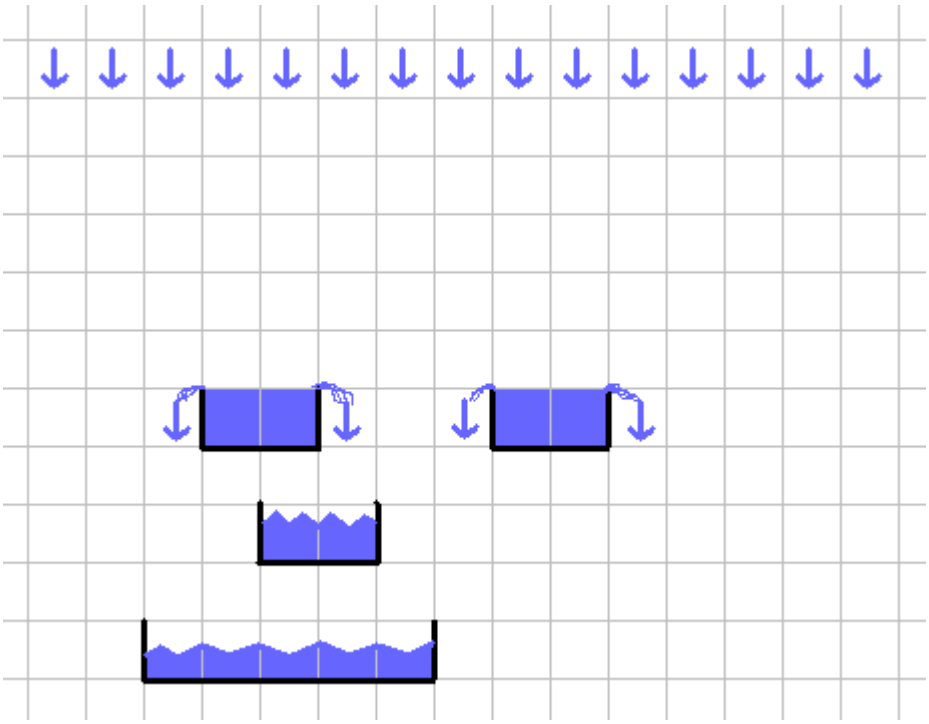
```
57 5 6
31 255 31
0 0 0
```

Sample Output

```
7467
8054234
```

Problem E
Rain Confusion

With today’s exorbitant bottled water prices, many people have decided to take control of their thirst-quenching destiny. To that end, everyone on you block has constructed a series of rain buckets on their houses in an effort to capture what the mad sky is giving away for free. Below is an example bucket configuration:



Due to poor planning, you and your neighbors have created a problem that is potentially more serious than the threat of dehydration: you can’t precisely calculate the amount of water you can expect to collect. Thirst is nothing if you can’t keep painstakingly detailed numbers on your do-it-yourself reservoir.

You must quickly write a program to assist in the data-gathering operation.

Since you live in a 2-dimensional universe, each reservoir can be described in a 2-dimensional coordinate system. Each reservoir is constructed from simple rectangle-buckets, whose vertices's lie on integer coordinates and whose edges are either parallel to or perpendicular to the ground. Each bucket has one edge missing to allow rain water to be collected—the missing edge is always the top edge. Rain falls straight down from above at a known rate per unit horizontal distance. Where the system gets interesting is the spillover effect between buckets. Most people on your block (to your chagrin, yourself included) simply threw buckets on the wall and tacked them where they landed. Thus, some buckets block the mouth of buckets lower on the wall. When a bucket fills its rectangular capacity, additional water proceeds to spill over the sides, adding to the rainfall in the adjacent unit of horizontal distance. The spillover is evenly split between the two sides. The figure below illustrates how rain can be expected to fall after a high bucket has filled to capacity:

No rain falls through a bucket, and the flow is increased in the squares adjacent to the filled bucket.

Rain that falls outside of any bucket is not collected.

Input

The input will consist of several reservoir systems. Each reservoir system will begin with a line containing 3 numbers T, F, B, separated by whitespace:

- T is the number seconds that rain will fall. T is an integer between 0 and 99, inclusive.
- F is the flow rate—the amount of water falling per horizontal distance of sky per second. F is a decimal number between 0.0 and 1.0.
- B is the number of buckets. B is an integer between 0 and 50.

The first line is then followed by B lines describing the coordinates of each bucket. The buckets may be specified in any order, but the coordinates of each bucket are always specified in the same order, with 4 integers separated by whitespace:

X Y W H
where:

- X is the x coordinate of the lower-left corner
- Y is the y coordinate of the lower-left corner
- W is the width of the bucket
- H is the height of the bucket

Every coordinate will be an integer between 0 and 100. All coordinates will be non-negative. The width and height will be positive. No buckets will overlap, intersect or touch.

The end of input is indicated by a 0 on a line by itself.

Output

For each system, print the system number (counting from 1). Then for each reservoir, print the bucket number followed by the total number of square units of rain collected to 2 decimal places on a line by itself. See the example output for exact layout. Print a blank line between systems.

Sample Input

```
12 0.1 4
2 0 5 1
4 2 2 1
3 4 2 1
8 4 2 1
5 0.1 2
0 0 10 1
0 2 10 1
100 0.0 0
0
```

Sample Output

```
System 1
Bucket 1: 2.60
Bucket 2: 1.40
Bucket 3: 2.00
Bucket 4: 2.00

System 2
Bucket 1: 0.00
Bucket 2: 5.00

System 3
```

Problem F
Driving Around, Spelling Things

In every state, there are 26 cities, all spelled with a single capital letter. Each state also has many roads connecting the various cities. One of our nation's most popular pastimes is to spell out various words in a state by driving through the cities of the appropriate letter, in order.

Before you take the plunge, you want to know if it's cracked up to be. Just what words can be spelled? Are they worth the time and gas money? You have decided to write a computer program that can test various words, given a road map of the state.

Input

The input will consist of several test cases. Each test case starts with a line containing a string **NAME**, followed by 2 integers, **R** and **W**, all separated by spaces:

- **NAME** is the name of the current state
- **R** is the number of roads through the current state
- **W** is the number of words that you want to check for this state

The next R lines each contain a description of a road. Each road description is two capital letters separated by a space. The roads are bidirectional, so any road description indicates a road connecting the cities in both directions. No roads are constructed that loop from one city back into itself, so the cities in the road description are always different.

The next W lines each contain a word to be spelled. The words are no more than 26 letters in length and consist of only capital letters.

The name state name of "DONE" indicates the end of input. No output should be printed for this case.

Output

For each word WORD to be spelled, print the results for that word as either

WORD can be spelled in NAME

or

WORD can not be spelled in NAME

On a single line, where NAME is the name of the current state.

Sample Input

```
DELAMONTANIDA 8 3
S I
R I
A R
N S
N I
B S
R B
A I
BRAINS
SCRABBLE
ZOMBIE
DONE
```

Sample Output

```
BRAINS can be spelled in DELAMONTANIDA
SCRABBLE can not be spelled in DELAMONTANIDA
ZOMBIE can not be spelled in DELAMONTANIDA
```

Problem G
Whack-a-mole Spawn From Hell

You are playing a game of whack-a-mole, in which moles pop their heads up in squares on a 2-dimensional grid and you whack them on the head, sending them back underground. But this is isn't your father's whack-a-mole, this is … *Whack-a-mole Spawn From Hell*. In this game, every time you whack a mole, 4 new moles are spawned, who then rise up in the four adjacent squares a second later.

Unfortunately, with all this spawning going on, things get a bit crowded underground. The moles are guided by a few devilish rules:

- Each mole starts out in a particular square and pops up at time 0, waiting patiently to be whacked
- Moles always pop up, like clockwork, on second-boundaries.
- When whacked, a mole retreats underground, spawns off 4 moles, sending each into an adjacent square, and then disappears. The spawned moles will pop up at the beginning of the next second.
- If there is already a mole in the square where a mole wants to pop up, the new mole disappears.
- If two moles arrive at the same time in a square, one of them stays in the square and one disappears.
- If spawned moles are sent into a square that is out of bounds, they disappear from the game.

You attack the hellspawn with 2 mallets, one held in each hand. In any given second, you are capable of hitting two spots on the grid, one with each mallet. First, the moles pop up, then you hit them with the mallet, then the moles go into whatever spawning frenzy suits them. When you use both mallets, you always do so in the same instant. If you whack adjacent moles, their spawn can move past one another to pop up the next second.

You are already dead-set on your whacking-plan—you know where you want to whack and when you'll do it. The only question that remains is how many moles are left standing at the end of the game. You lay down your mallets for a moment to write a computer program to find out.

Input

The input will consist of several game plans. Each game plan starts with a line containing 5 integers <**G**, **R**, **C**, **M**, **W**>, each separated by a single space:

- **G** is the number seconds that the game lasts, an integer between 1 and 100, inclusive.
- **R** is the number of rows in the game grid, an integer between 1 and 100, inclusive.
- **C** is the number of columns in the game grid, an integer between 1 and 100, inclusive.
- **M** is the number of moles at time 0, an integer between 0 and 10, inclusive.
- **W** is the number of whacks you have planned, an integer between 0 and 200, inclusive.

The next **M** lines describe the starting positions of the moles, each a line containing 2 integers separated by a space. The first integer describes the mole's starting *column* and is the range 0 to (**C**-1), inclusive. The second integer describes the mole's starting *row* and is in the range 0 to (**R**-1), inclusive. Note that this is (column, row), not (row, column).

The next W lines describe planned whacking moves, each a line of the form <**T H X Y**>, where:

- **T** is the time, in seconds when the whack will occur, an integer between 0 and (G-1), inclusive.
- **H** is the either L or R, indicating which hand is doing the whacking
- **X** is the *column* where the whacking occurs, an integer in the range 0 to (C-1), inclusive
- **Y** is the *row* where the whacking occurs, an integer in the range 0 to (R-1), inclusive

The whacking moves may be listed in any order. The 2 mallets ever whack the same location in the same second. Each mallet will whack at most one location each second.

Note: for a given 1-second timeslot, the moles move first, then the mallets move. The mallets move simultaneously if both are used during the timeslot.

End of input is indicated by a line containing 5 zeros, each separated by a single space.

Output

For each game plan, output, on a line by itself, a single integer indicating the number of moles that pop up at--or remain popped up at--time G.

Sample Input

```
10 3 8 2 2
6 1
6 2
4 L 6 1
4 R 1 1
10 10 10 1 3
1 1
1 R 1 1
2 R 1 2
3 R 1 3
0 0 0 0 0
```

Sample Output

```
Game 1: 4
Game 2: 10
```

Problem H
World of Wealth

You are playing a game called World of Wealth (WoW). In this game, you can make money by buying materials, making things with them and selling things. You are trying to raise enough money for a new hat (it will make even the noobiest noob understand that you will pwn them). When you sell things, it takes time for a buyer to purchase them, so you want to make your money in the least possible number of rounds of selling. Each round consists of buying items, making items and selling resulting items. The left-over money made from the previous round plus any sales from that round go into the money you have available for the next round. You can buy as many of any items as you can afford each round and make as many as you have supplies for. However, for simplicity you will sell only one item type each round.

Given your starting money amount, the target money amount, a list of prices (for buying or selling) and a list of transforms that you can perform, what items and how many should be made each round? When multiple items could types be created and sold, choose the item and quantity that results in the highest profit for that round.

Input

The input may consist of multiple cases. Each case will begin with two integers being the starting amount and target amount respectively. The end of input is indicated by either of these numbers being zero.

Next will be a positive integer indicating the number (N) of different items in the price list. N will be at most 40. This will be followed by N lines, one per item. Each line will consist of an integer price and the item name.

Next will be a non-negative integer indicating the number (M) of different items you can make. M will be at most 40. This will be followed by M lines, one per item. Each line will consist of an item name followed by a colon and a list of items that are required to make that item. Each item that goes into an item will be preceded by an integer indicating how many will be needed.

If an item does not appear on the price list, it cannot be bought or sold.

Example:
Red_Linen_Shirt : 2 Bolt_of_Linen_Cloth 1 Red_Dye 1 Course_Thread
This means that one Red_Linen_Shirt may be made if you have 2 Bolt_of_Linen_Cloth, 1 Red_Dye and 1 Course_Thread.

All item names will consist of upper or lower case letters, underscores (_) and hyphens (-). They will not contain any spaces or other whitespace.

All prices and quantities will be positive integers.

Output

For each round, print the round number, the item to sell and the quantity to sell. Print a blank line between output cases.

Sample Input

```
10 500
5
40 Red_Linen_Shirt
1 Linen_Cloth
10 Bolt_of_Linen_Cloth
2 Red_Dye
1 Coarse_Thread
2
Red_Linen_Shirt : 2 Bolt_of_Linen_Cloth 1 Red_Dye 1 Coarse_Thread
Bolt_of_Linen_Cloth : 2 Linen_Cloth
1 1000
5
1 Copper_Ore
3 Copper_Bar
3 Tin_Ore
3 Tin_Bar
20 Bronze_Bar
3
Tin_Bar : 1 Tin_Ore
Copper_Bar : 1 Copper_Ore
Bronze_Bar : 1 Tin_Bar 1 Copper_Bar
0 0
```

Sample Output

```
Round 1: Bolt_of_Linen_Cloth 5
Round 2: Red_Linen_Shirt 7
Round 3: Red_Linen_Shirt 40

Round 1: Copper_Bar 1
Round 2: Copper_Bar 3
Round 3: Bronze_Bar 2
Round 4: Bronze_Bar 10
Round 5: Bronze_Bar 50
```