Problem D: The Mastermind Master's Mind

The Advancement of Collegiate Mastermind (hey, that's ACM again...weird!) is an organization which (among other things) holds classes for college students to improve their Mastermind skills. Recall that basic Mastermind is a two-player game which works as follows: The first player – the codemaker – creates a secret 4-color code, each color taken from one of six colors (we'll use A,B,C,D,E and F for the colors). The other player – the codebreaker – now must try to guess the codemaker's code. After each guess, the codemaker uses black and white pegs to tell the codebreaker two things: the number of correct colors in the correct positions (the black pegs), and the number of remaining correct colors that are in the wrong positions (the white pegs). For example, if the true code is ABCC, and the codebreaker makes the guess ACCD, then the response would be 2 black and 1 white; if the guess was CCAA, the response would be 3 white. The codebreaker continues making guesses until he receives 4 blacks. More advanced versions of Mastermind increase both the length of the code and the number of colors to choose from.

The ACM's master instructor is Dee Sifer, and she has a unique ability: when given a set of n guesses and responses, she can immediately determine what the best $(n+1)^{\text{st}}$ guess should be. For each possible $(n+1)^{\text{st}}$ guess, Dee calculates (in her head) the number of codes left for each possible response she could get to that guess. The maximum of these numbers over all responses is called the *uncertainty* of the guess. The "best" guess is the one with the minimum uncertainty. For example, suppose that you get to a point in a game where you've narrowed down the answer to only three possible codes: ABBB, ABBC or ABCB. If your next guess is ABBB, there would be two possible responses: 4 blacks (leaving 0 remaining possibilities left) or 3 blacks (leaving 2 remaining possibilities – ABBC and ABCB). However, if instead of ABBB you try ABBC, then there are 3 possible responses: 4 blacks (leaving 0 possibilities), 3 blacks (leaving 1 possibility – ABBB) and 2 blacks and 2 whites (also leaving 1 possibility – ABCB). Thus ABBC would be a better guess in this case, since the uncertainty it leaves is 1, whereas the uncertainty for ABBB is 2.

This is all well and good, except for one thing. You have been selected as Dee's successor, and you do NOT have Dee's ability to pick the minimum uncertainty guess. Dee has been dropping hints (in code) that she plans to retire soon, so you need a program to help you simulate her ability.

Input

Input will consist of multiple test cases. The first line of the input file will contain a single integer indicating the number of test cases. Each test case will consist of several lines. The first line will contain three integers: l c n, where l is the length of the code being guessed, c is the number of colors to choose from, and n is the number of guesses made so far. These values will satisfy $1 \leq l \leq 15, 1 \leq c \leq 20, 0 \leq n \leq 10$. The values of l and c will be such that the total possible number of codes will be ≤ 32768. After this will come n lines of the form

`guess` b w

where `guess` is a length-l character string specifying a guess, and b and w are the number of black and white pegs in the response. All colors will be uppercase letters taken from the first c letters of the alphabet. For each test case, the guesses given will limit the total number of possible solutions to ≤ 1500.

Output

For each test case, output a single line containing the best guess and its uncertainty. Use a single blank to separate the guess from the uncertainty. If there is more than one guess with the same minimum uncertainty, use the one which comes first lexicographically.

Sample Input

```
3
4 6 2
AABC 1 2
BEAC 0 3
4 6 1
ABCD 0 0
3 20 4
ABE 1 0
ROM 1 0
INK 1 0
MOB 0 2
```

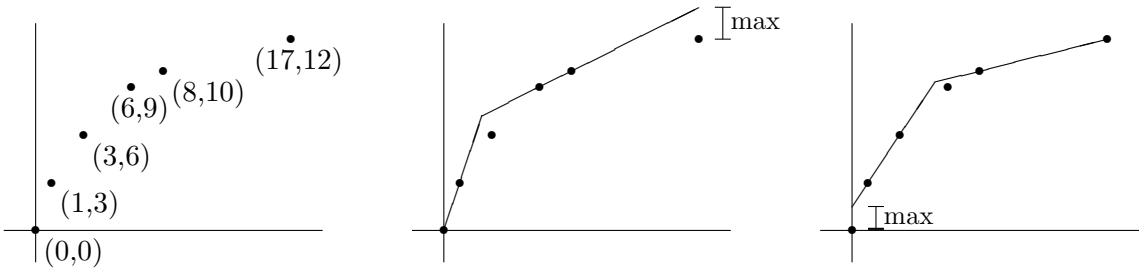
Sample Output

```
ABCD 4
AEEE 3
IBM 0
```

Problem F: Roofing It

Bill Eaves owns the *Shingle Minded* roofing company which is a sub-contracting firm specializing in putting roofs on buildings. Often, Bill will receive a design for a roof from some young hot-shot architect which, though it may have some aesthetic appeal, is totally impractical. In these cases, Bill will begin negotiations with the architect and the client to find a simpler design. Bill's only concern is that the roof be convex, to allow rain, snow and frisbees to roll off easily. The architect's main concern is that the maximum height between his original roof and the compromise roof be as small as possible. The client's main concern is to get out of this meeting as soon as possible and get back to watching TV.

The architect's plans for the roof come in the form of a series of n points (x_i, y_i) specifying the outline of the roof as seen from the side of the house. The roofs are always symmetrical, so the architect only shows the front side of the roof (from its start at the front of the house to its peak). Once Bill gets these plans and a decision is made on how many sections the convex roof should have, he must decide how to place the sections so as to 1) make sure that all the original (x_i, y_i) points lie on or below the new roof, and 2) to minimize the maximum vertical distance between any of the original (x_i, y_i) points and the new roof. All sections must lie on at least two of the initial points specified by the architect. An example is shown below. On the left are the initial points from the architect. The next two pictures show an approximation of the roof using only two sections. While both of these are convex, the second of the two is the one which minimizes the maximum distance.



Input

Input will consist of multiple test cases. Each case will begin with two positive integers n and k ($2 \le n \le 100, 1 \le k < n$) indicating the number of points used in the original roof plan and the number of sections used in the compromise roof. These will be followed by n lines each containing two floating points numbers x_i y_i, specifying the n points in the roof plan. These values will be given in increasing order of x, and the last point will be guaranteed to have the highest y value of any of the points. All values will be between 0.0 and 10000.0. The last case is followed by a line containing 0 0 which indicates end-of-input and should not be processed.

Output

For each test case, output the maximum distance between the best compromise roof and the initial points, rounded to the nearest thousandth.

Sample Input

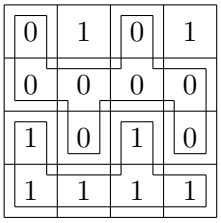
```
6 2
0.0 0.0
1.0 3.0
3.0 6.0
6.0 9.0
8.0 10.0
17.0 12.0
0 0
```

Sample Output

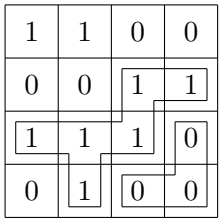
```
1.500
```

Problem H: Stake Your Claim

The designers at Gazillion Games Inc. have come up with a new, relatively simple game called "Stake Your Claim". Two players – 0 and 1 – initially select two values n and m and an $n \times n$ board is created with m 0's and m 1's randomly placed on the board. Starting with player 0, each player puts his/her number in one of the empty squares on the board. After the board is filled, each player's score is equal to the largest connected region on the board filled with that player's number (where a connected region is one where for any two squares in the region a path exists consisting of only N/S/E/W moves). The player with the highest score wins, and is awarded the difference between his/her score and the score of the other player. Two examples of finished games are shown below, with the largest connected regions for each player outlined. Note in the second example that the two sections with 2 0's each are not connected.



Player 0's score: 8
Player 1's score: 6
Player 0 awarded 2 points



Player 0's score: 3
Player 1's score: 6
Player 1 awarded 3 points

In order to test how good this game is, the gang at Gazillion has hired you to write a program which can play the game. Specifically, given any starting configuration, they would like a program to determine the best move for the current player, i.e., the score which maximizes the points awarded to that player (or minimizes those awarded to the player's opponent).

Input

Input will consist of multiple test cases. Each test case will start with a line containing a positive integer n (≤ 8) indicating the size of the board. Next will come n lines describing the current board layout (row 0 first, followed by row 1, etc). Each of these lines will contain n characters taken from `'0'`, `'1'` and `'.'`, where `'.'` represents an empty square. The first character will be in column 0, the second in column 1, etc. The number of 0's on the board will either be equal to the number of 1's or one greater, and there will be between 1 and 10 (inclusive) empty squares. The last case is followed by a line containing 0 which indicates end-of-input and should not be processed.

Output

For each test case, output a single line containing two items: the coordinates of the best move for the player and the best point total achieved by that player. In case of ties, print the move which comes first lexicographically. Use the format shown in the sample output.

Sample Input

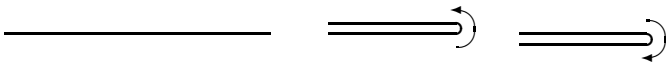
```
4
01.1
00..
.01.
...1
4
0.01
0.01
1..0
.1..
0
```

Sample Output

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

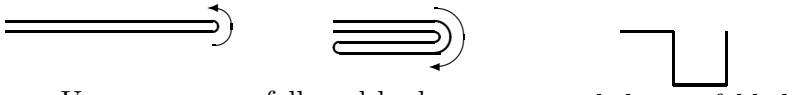
Problem B: A Foldy but a Goody

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 C: Give Me an E

Everyone knows that the letter "E" is the most frequent letter in the English language. In fact, there are one hundred sixteen E's on this very page ... no, make that one hundred twenty one. Indeed, when spelling out integers it is interesting to see which ones do NOT use the letter "E". For example 6030 (six thousand thirty) doesn't. Nor does 4002064 (four million two thousand sixty four).

It turns out that 6030 is the 64th positive integer that does not use an "E" when spelled out and 4002064 is the 838th such number. Your job is to find the n-th such number.

Note: 1,001,001,001,001,001,001,001,001,000 is "one octillion, one septillion, one sextillion, one quintillion, one quadrillion, one trillion, one billion, one million, one thousand". (Whew!)

Input

The input file will consist of multiple test cases. Each input case will consist of one positive integer n (less than 2^{31}) on a line. A 0 indicates end-of-input. (There will be no commas in the input.)

Output

For each input n you will print, with appropriate commas, the n-th positive integer whose spelling does not use an "E". You may assume that all answers are less than 10^{28}.

Sample Input

```
1
10
838
0
```

Sample Output

```
2
44
4,002,064
```

Problem D: Lemmings, Lemmings Everywhere. But Not For Long.

On an $n \times m$ board there is a lemming on each square. Every second, the lemmings try to move either north, south, east or west, according to rules which are explained below. To determine which direction to move in, each lemming has an agenda, which is an ordering of the four possible directions (for example, one possible agenda might be NWES). The rules for lemming movement are the following:

1. Initially each lemming sets its direction of movement **D** to the first direction in its agenda.
2. At each time step, each lemming tries to move in its direction **D**. Three things can happen to lemming L:
   (a) If L's current direction **D** causes it to move off the board, then the world has one less lemming in it. Otherwise, L's target destination will be to another square.
   (b) If L's target square is empty or about to become empty as a result of another lemming leaving it, and no other lemming wants to move to the same square, then L moves into its target square. In this case, the lemming will use the same direction **D** in the next time step.
   (c) Otherwise, if another lemming is trying to move into L's target square, or if the target square contains a lemming which cannot move, then L stays put. In this case, it will update its **D** by going to the next direction in its agenda (wrapping around to the beginning if necessary).

Two lemmings which want to exchange squares can do so, unless of course some other lemming is trying to move into one of their two squares (in which case all three of the lemmings would stay in their current squares). Lemmings being lemmings, they continue to move until all of them have moved off the board. Your job is to determine how long that takes.

Input

Input will consist of multiple test cases. Each test case will consist of multiple lines. The first line will contain two positive integers n m, specifying the number of rows and columns in the board. The maximum value for each of these is 100. The board is situated so that square $(0, 0)$ is in the southwest corner, and square $(0, m-1)$ is in the southeast corner. Following this will be several rows containing the agendas for the nm lemmings. Each agenda will be a permutation of the string `NESW`. There will be 16 agendas on each line (except perhaps the last), with a single space between each. The agendas are assigned row-wise to the lemmings, so that the first agenda is associated with the lemming on square $(0, 0)$, the second with the lemming on square $(0, 1)$, and so on. The last case will be followed by the line 0 0 which will terminate input.

Output

For each test case, output a single line containing the case number (using the format shown below) followed by the number of steps it takes until the last lemming(s) falls off the board. Use only single spaces to separate items on a line.

(over)

Sample Input

```
2 2
ENWS WSNE NESW WENS
2 2
ENWS WSNE NESW SWEN
0 0
```

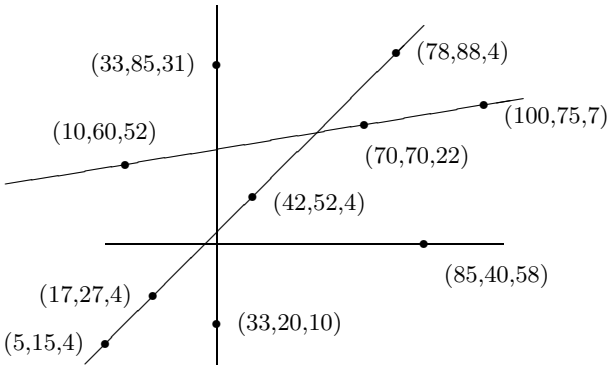
Sample Output

```
Case 1: 2
Case 2: 3
```

Problem H: Target Practice

There are many forms of contests where the contestants (shooters) try to hit targets, either moving or still. In this version there are a number of small balloons sitting on the tops of poles that are in turn stuck in the ground at various points in a large field. These poles are not all the same height. The shooter circles the field and fires at the balloons, the goal being to burst all the balloons with as few shots as possible. Since the balloons offer almost no resistance to a bullet, the bullet will pass right through and possibly hit one or more other balloons. So, by judiciously taking shots, the shooter might need only a very few shots to hit all the targets (provided the shooter is a good marksman, which we will assume is the case).

For example, the following field of 10 targets can be covered in only four shots, as shown. (The first two numbers at each position indicate the position of the balloon, and the third number the height.)



Your job is to determine the fewest number of shots necessary to hit all the targets in a given field.

Input

There will be multiple test cases. Each test case will consist of an integer n (≤ 50) indicating the number of target positions to follow. A value of $n = 0$ indicates end of input. There will follow n integer triples, x y h, indicating a balloon at position (x, y) in the field at height h. (There will be at most one balloon at any position (x, y).) All integers are greater than 0 and no greater than 100. Furthermore assume that the shooter can take shots from anywhere on the field at any height. For simplification, assume here that the balloons are points and that the bullets can pass through the poles on which the balloons are perched.

Output

Each test case should produce one line of output of the form:

```
Target set k can be cleared using only s shots.
```

where `k` is the number of the test case, starting at 1, and the value of `s` is the minimum number of shots needed to hit all the targets in the set.

(over)

Sample Input

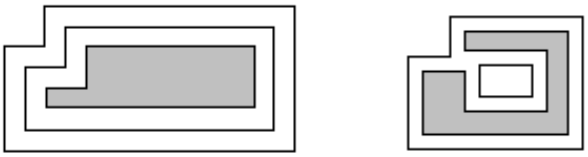
```
10
5 15 4 10 60 52 17 27 4 33 20 10 33 85 31 42 52 4 70 70 22
78 88 4 85 40 58 100 75 7
9
5 15 4 10 60 52 17 27 4 33 20 10 33 85 31 42 52 4 70 70 22
78 88 4 100 75 7
0
```

Sample Output

```
Target set 1 can be cleared using only 4 shots.
Target set 2 can be cleared using only 3 shots.
```

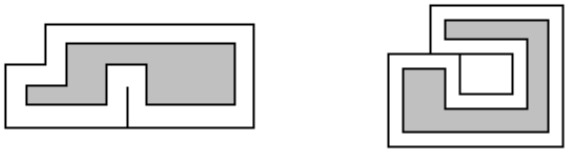
Problem A: Bordering on Madness

Bob Roberts owns a design business which creates custom artwork for various corporations. One technique that his company likes to use is to take a simple rectilinear figure (a figure where all sides meet at 90 or 270 degrees and which contains no holes) and draw one or more rectilinear borders around them. Each of these borders is drawn so that it is a set distance d away from the previously drawn border (or the original figure if it is the first border) and then the new area outlined by each border is painted a unique color. Some examples are shown below (without the coloring of the borders).



The example on the left shows a simple rectilinear figure (grey) with two borders drawn around it. The one on the right is a more complicated figure; note that the border may become disconnected.

These pieces of art can get quite large, so Bob would like a program which can draw prototypes of the finished pieces in order to judge how aesthetically pleasing they are (and how much money they will cost to build). To simplify things, Bob never starts with a figure that results in a border where 2 horizontal (or vertical) sections intersect, even at a point. This disallows such cases as those shown below:



Input

Input will consist of multiple test cases. The first line of the input file will contain a single integer indicating the number of test cases. Each test case will consist of two or more lines. The first will contain three positive integers n, m and d indicating the number of sides of the rectlinear figure, the number of borders to draw, and the distance between each border, where $n \leq 100$ and $m \leq 20$. The remaining lines will contain the n vertices of the figure, each represented by two positive integers indicating the x and y coordinates. The vertices will be listed in clockwise order starting with the vertex with the largest y value and (among those vertices) the smallest x value.

Output

For each test case, output three lines: the first will list the case number (as shown in the examples), the second will contain m integers indicating the length of each border, and the third will contain m integers indicating the additional area contributed to the artwork by each border. Both of these sets of numbers should be listed in order, starting from the border nearest the original figure. Lines two and three should be indented two spaces and labeled as shown in the examples. Separate test cases with a blank line.

Sample Input

```
2
6 2 10
20 30 100 30 100 0 0 0 0 10 20 10
10 1 7
20 50 70 50 70 0 0 0 0 30
20 30 20 10 60 10 60 40 20 40
```

Sample Output

```
Case 1:
  Perimeters: 340 420
  Areas: 3000 3800

Case 2:
  Perimeters: 380
  Areas: 2660
```

Problem B: Jack of All Trades

Jack Barter is a wheeler-dealer of the highest sort. He'll trade anything for anything, as long as he gets a good deal. Recently, he wanted to trade some red agate marbles for some goldfish. Jack's friend Amanda was willing to trade him 1 goldfish for 2 red agate marbles. But Jack did some more digging and found another friend Chuck who was willing to trade him 5 plastic shovels for 3 marbles while Amanda was willing to trade 1 goldfish for 3 plastic shovels. Jack realized that he could get a better deal going through Chuck (1.8 marbles per goldfish) than by trading his marbles directly to Amanda (2 marbles per goldfish).

Jack revels in transactions like these, but he limits the number of other people involved in a chain of transactions to 9 (otherwise things can get a bit out of hand). Normally Jack would use a little program he wrote to do all the necessary calculations to find the optimal deal, but he recently traded away his computer for a fine set of ivory-handled toothpicks. So Jack needs your help.

Input

Input will consist of multiple test cases. The first line of the file will contain an integer n indicating the number of test cases in the file. Each test case will start with a line containing two strings and a positive integer $m \leq 50$. The first string denotes the items that Jack wants, and the second string identifies the items Jack is willing to trade. After this will be m lines of the form

a_1 $name_1$ a_2 $name_2$

indicating that some friend of Jack's is willing to trade an amount a_1 of item $name_1$ for an amount a_2 of item $name_2$. (Note this does not imply the friend is also willing to trade a_2 of item $name_2$ for a_1 of item $name_1$.) The values of a_1 and a_2 will be positive and ≤ 20. No person will ever need more than $2^{31} - 1$ items to complete a successful trade.

Output

For each test case, output the phrase `Case i:` (where `i` is the case number starting at 1) followed by the best possible ratio that Jack can obtain. Output the ratio using 5 significant digits, rounded. Follow this by a single space and then the number of ways that Jack could obtain this ratio.

Sample Input

```
2
goldfish marbles 3
1 goldfish 2 marbles
5 shovels 3 marbles
1 goldfish 3 shovels
this that 4
7 this 2 that
14 this 4 that
7 this 2 theother
1 theother 1 that
```

Sample Output

```
Case 1: 1.8000 1
Case 2: 0.28571 3
```

Almost blank page

Problem G: The Worm Turns

Winston the Worm just woke up in a fresh rectangular patch of earth. The rectangular patch is divided into cells, and each cell contains either food or a rock. Winston wanders aimlessly for a while until he gets hungry; then he immediately eats the food in his cell, chooses one of the four directions (north, south, east, or west) and crawls in a straight line for as long as he can see food in the cell in front of him. If he sees a rock directly ahead of him, or sees a cell where he has already eaten the food, or sees an edge of the rectangular patch, he turns left or right and once again travels as far as he can in a straight line, eating food. He never revisits a cell. After some time he reaches a point where he can go no further so Winston stops, burps and takes a nap.

For instance, suppose Winston wakes up in the following patch of earth (X's represent stones, all other cells contain food):

	0	1	2	3	4
0					X
1					
2					
3		X	X		
4					

If Winston starts eating in row 0, column 3, he might pursue the following path (numbers represent order of visitation):

	0	1	2	3	4
0	4	3	2	1	X
1	5	18	17	16	15
2	6	19	20	21	14
3	7	X	X	22	13
4	8	9	10	11	12

In this case, he chose his path very wisely: every piece of food got eaten. Your task is to help Winston determine where he should begin eating so that his path will visit as many food cells as possible.

Input

Input will consist of multiple test cases. Each test case begins with two positive integers, m and n, defining the number of rows and columns of the patch of earth. Rows and columns are numbered starting at 0, as in the figures above. Following these is a non-negative integer r indicating the number of rocks, followed by a list of $2r$ integers denoting the row and column number of each rock. The last test case is followed by a pair of zeros. This should not be processed. The value $m \times n$ will not exceed 625.

Output

For each test case, print the test case number (beginning with 1), followed by four values:

```
amount row column direction
```

where `amount` is the maximum number of pieces of food that Winston is able to eat, `(row, column)` is the starting location of a path that enables Winston to consume this much food, and `direction` is

one of `E, N, S, W`, indicating the initial direction in which Winston starts to move along this path. If there is more than one starting location, choose the one that is lexicographically least in terms of row and column numbers. If there are optimal paths with the same starting location and different starting directions, choose the first valid one in the list `E, N, S, W`. Assume there is always at least one piece of food adjacent to Winston's initial position.

Sample Input

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

Sample Output

```
Case 1: 22 0 3 W
```

Problem B: Cover Up

"The Price is Right" is a popular game show where contestants play various games to win fabulous prizes. One of the games played on the show is called "Cover Up" whose object is to guess a 5-digit number (actually, the price of a new car). In the actual game, contestants are given two numbers to choose from for the first digit, three numbers to choose from for the second digit, and so on. A contestant selects one number for each digit (from the set of yet unpicked numbers for that digit) and then is told which ones are correct; if at least one is correct, the player is allowed to guess again for all incorrect digits. The contestant keeps guessing as long as they keep getting at least one new digit correct. The game ends when either all the digits are correct (a win for the contestant) or after a turn when no new digit is guessed correctly (a loss).

Typically this game is not sheer luck. For example, suppose you had the following five possibilities for the last digit: 1, 3, 5, 8 and 9. Many car prices end with either a 5 or a 9, so you might have, say, a 70% chance that one of these two numbers is correct; this breaks down to a 35% chance for either the 5 or the 9 and a 10% chance for each of the other three digits. Now say you pick the 5 and it's wrong, but some other guess you made was right so you still get to play. With this additional information the probabilities for the remaining 4 numbers change: the probability for the 9 is now close to around 54%, while each of the other three numbers now has a little over a 15% chance. (We'll let you figure out how we got these values). We'll call the 5 and the 9 in the original group the *known* candidates, and typically there are known candidates in other columns as well. For example, if the two numbers for the first (high order) digit are 1 and 9, the contestant can be 100% sure that the 1 is the correct digit (there aren't too many $90,000 cars to be given away).

For this problem, you are to determine the probability of winning the game if an optimal strategy for picking the numbers (based on probabilities such as those described above) is used.

Input

Each test case will consist of two lines. The first will be n, the number of digits in the number to be guessed. The maximum value of n will be 5. The second line will contain n triplets of numbers of the form m l p where m is the number of choices for a digit, l is the number of known candidates, and p is the probability that one of the known candidates is correct. In all cases $0 \leq l < m \leq 10$ and $0.0 \leq p \leq 1.0$. Whenever $l = 0$ (i.e., when there are no known candidates) p will always be 0.0. A line containing a single 0 will terminate the input.

Output

Output for each test case is the probability of winning using optimal strategy. All probabilities should be rounded to the nearest thousandth, and trailing 0's should not be output. (A 100% chance of winning should be output as 1.)

Sample Input

```
2
3 1 0.8 2 0 0.0
2
3 2 0.8 2 0 0.0
2
3 2 0.82 2 1 0.57
3
4 1 1.0 3 0 0.0 10 1 1.0
0
```

Sample Output

```
0.85
0.6
0.644
1
```

Problem C: Decompressing in a GIF

One well known method to compress image files is the Graphics Interchange Format (GIF) encoding, created by CompuServe in 1987. Here's a simplified version applied to strings of alphabetic characters. Essential for this compression is a dictionary which assigns numeric encodings (we'll use base 10 numbers for this problem) to different strings of characters. The dictionary is initialized with mappings for characters or substrings which may appear in the string. For example, if we expect to encounter all 26 letters of the alphabet, the dictionary will initially store the encodings $(A, 00)$, $(B, 01)$, $(C, 02)$, ..., $(Z, 25)$. If we are compressing DNA data, the dictionary will initially store only 4 entries: $(A, 0)$, $(T, 1)$, $(G, 2)$ and $(C, 3)$. Note that the length of each initial encoding is the same for all entries (2 digits in the first example, and 1 digit in the second).

The compression algorithm proceeds as follows:

1. Find the longest prefix of the uncompressed portion of the string which is in the dictionary, and replace it with its numeric encoding.
2. If the end of the string has not been reached, add a new mapping (s, n) to the dictionary, where $s =$ the prefix just compressed plus the next character after it in the string, and $n =$ the smallest number not yet used in the dictionary.

For example, assume we started with the string `ABABBAABB` and a dictionary with just two entries, $(A, 0)$ and $(B, 1)$. The table below shows the steps in compressing the string.

String	Longest Prefix	Replaced With	New Dictionary Entry
ABABBAABB	A	0	$(AB, 2)$
0BABBAABB	B	1	$(BA, 3)$
01ABBAABB	AB	2	$(ABB, 4)$
012BAABB	BA	3	$(BAA, 5)$
0123ABB	ABB	4	—

The final compressed string is `01234`.

There is only one other rule: the replacement strings used are always the size of the longest encoding in the dictionary at the time the replacement occurs. Thus, with the dictionary above, if the string to compress is long enough that an entry of the form $(s, 10)$ is added to the dictionary, then from this point on all numerical replacement strings used in the compressed string must be expanded to 2 digits long (i.e., `A` will now be encoded as 00, `B` as 01, `AB` as 02, etc.); if an entry $(s', 100)$ is added to the dictionary, all replacements from this point forward will increase to 3 digits long, and so on. Thus, the longer string `ABABBAABBAABAABAB` will be encoded as 01234027301, not 0123402731. Try it!

OK, now that you are experts at compressing, it's time to relax and decompress!

Input

Each test case will consist of two lines. The first line will contain a string of digits to decompress. The second line will contain the initial dictionary used in the compression. This line will start with a positive integer n indicating the number of entries in the dictionary ($1 \leq n \leq 100$), followed by n alphabetic strings. The first of these will be paired with 0 in the dictionary (or 00 if $n > 10$), the second with 1, and so on. The last test case will be followed by a line containing a single 0.

Output

For each test case, output a single line containing the case number (using the format shown below) followed by the decompressed string. All input strings will have been legally compressed.

Sample Input

```
01234
2 A B
01234027301
2 A B
02151120182729
26 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
21104
3 BA A C
01
2 JA VA
0
```

Sample Output

```
Case 1: ABABBAABB
Case 2: ABABBAABBAABAABAB
Case 3: CPLUSPLUS
Case 4: CAABAAA
Case 5: JAVA
```

Problem F: Here's a Product Which Will Make You Tensor

Most people are familiar with how to multiply two matrices together. However, an alternate form of multiplication known as tensor multiplication exists as well, and works more like you would expect matrix multiplication should. Let A be a $p \times q$ matrix and B be an $n \times m$ matrix, where neither A nor B is a 1×1 matrix. Then the tensor product $A \otimes B$ is a $pn \times qm$ matrix formed by replacing each element a_{ij} in A with the matrix $(a_{ij}) \cdot B$. Two examples are shown below, which also demonstrate that, like normal matrix multiplication, tensor multiplication is non-commutative:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 2 & 2 & 2 & 4 \\ 3 & 3 & 3 & 4 & 4 & 4 \\ 3 & 3 & 3 & 4 & 4 & 4 \\ 3 & 3 & 6 & 4 & 4 & 8 \end{bmatrix}, \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 2 & 4 \\ 3 & 4 & 3 & 4 & 6 & 8 \end{bmatrix}$$

Note that there is no restriction that the number of columns in the first matrix must equal the number of rows in the second, as there is with normal matrix multiplication. The object of this problem is to determine the number of ways (if any) a given matrix can be formed as a result of a tensor multiplication.

Input

The first line of input for a test case will contain two positive integers r and c indicating the number of rows and columns in the matrix. After this will follow r lines each containing c positive integers. The values of r and c will be ≤ 500, each entry in the matrix will be no greater than 65,536, and the last test case is followed by a line containing `0 0`.

Output

For each test case, output the number of different ways the matrix could be the tensor product of two positive integer matrices, neither of which is a 1×1 matrix.

Sample Input

```
6 6
1 1 1 2 2 2
1 1 1 2 2 2
1 1 2 2 2 4
3 3 3 4 4 4
3 3 3 4 4 4
3 3 6 4 4 8
2 2
3 6
4 9
2 4
15 18 30 36
20 24 40 48
0 0
```

Sample Output

```
1
0
4
```

Almost blank page

Problem G: Trip the Lights Fantastic

Bob Roberts (father of Little Bobby of problem D) works at the Traffic Commission for a medium size town. Bob is in charge of monitoring the traffic lights in the city and dispatching repair crews when necessary. Needless to say, Bob has a lot of free time, so to while away the hours he tries to figure out the quickest way to take short trips between various points in the city. Bob has at his disposal a lot of information: the layout of streets in the city and the location and cycle times for all of the traffic lights. To simplify the solution process, he makes the following assumptions:

1. All cars travel at the same top speed, and, if sitting at a red light, take 5 seconds to react and get up to speed. (That is, Bob assumes the car is essentially standing still for 5 seconds, then proceeds at top speed. Bob also assumes the light will not have turned back to red in the 5 seconds it takes to get going.)
2. Each car approaches a light at full speed and either passes through the light if it is green or yellow, or comes to an immediate stop if it is red. Cars are allowed to pass through a light if they hit it just as it is turning to green. Cars must stop if they reach the light just as it is turning to red.
3. The time to make turns through a light is ignored. It is possible to travel between any two lights, although perhaps not directly.

Furthermore, no u-turns are allowed nor will routes revisit an intersection. Even given these assumptions, Bob has difficulty coming up with minimum time paths. Let's see if you can help him.

Input

The first line of each test case will contain four positive integers n, m, s, and e, where n ($2 \leq n \leq 100$) is the number of traffic lights (numbered 0 through $n-1$), m is the number of roads between the traffic lights, and s and e ($s \neq e$) are the starting and ending lights for the desired trip. There will then follow n lines of the form g y r indicating the number of seconds that each light is green, then yellow, then red. ($1 \leq g, y, r \leq 100$.) The first of these lines refers to light 0, the second to light 1, and so on. Following these n lines will be m lines, each describing one road. These lines will have the form $l1$ $l2$ t, where $l1$ and $l2$ are the two lights being connected by the road and t is the time (in seconds, $t \leq 500$) to travel the length of the road at full speed — you should add 5 to this value to obtain the travel time when driving the road beginning at a standstill. All roads are two way. At time 0, all lights are just starting their green period and your car is considered to be at a standstill at traffic light s. Since it takes 5 seconds to get going, you may assume that $g + y$ is never less than or equal to 5. The last test case is followed by a line containing `0 0 0 0` indicating end-of-input.

Output

For each test case, output a single line containing the minimum time to travel from the start light to the end light. Output your results in the form `mm:ss` indicating the number of minutes and seconds the trip takes. If the number of seconds is less than 10 then preface it with a 0 (i.e., output `4:05`, not `4:5`). Likewise, if the number of minutes is less than 10, print just one digit (as in `4:05`).

Sample Input

```
3 3 0 2
3 4 5
3 3 3
2 4 4
0 1 1
1 2 2
0 2 12
3 3 0 2
3 4 5
3 4 3
2 4 4
0 1 1
1 2 2
0 2 12
0 0 0 0
```

Sample Output

```
0:16
0:08
```