A: Vowel Frequencies

Time Limit: 2 sec

The English alphabet consists of 26 letters. Five of these (a, e, i, o and u) are classified as vowels, the remaining 21 as consonants. Almost every English word contains at least one vowel ("rhythm" is one of the few exceptions).

In this problem you will be given a number of pieces of English text. Your task is to determine the frequency of each vowel that is found in the piece, and to display the answers sorted by frequency, highest frequency first. Where two vowels are equally frequent, they are to be displayed in alphabetical order.

As you can see from the examples below, upper case and lower case letters are considered to be the same letter in this problem. Use lower case in your output. As you can see from the second example, a frequency of zero must still be displayed.

Input

Each piece of text to be analysed is on a separate line of the input file. Each line has at most 200 characters. A single # on a line indicates the end of input.

Output

Output for a problem must be on a single line. Each vowel must be output in lower case, followed by a colon, followed by the frequency of that vowel. There must be one space before the next letter, and a dot at the end.

Sample Input

```
This piece of text was written in the city of Auckland.
ACM Programming Contest.
#
```

Sample Output

```
e:5 i:5 a:3 o:2 u:1.
a:2 o:2 e:1 i:1 u:0.
```

B: Jelly

Time Limit: 2 sec

A local school provides jelly for their pupils every day, and the school staff are very careful to see that each child has exactly the same amount.

The jelly is prepared the previous day; the liquid jelly is poured into rectangular sided moulds, one mould per child, and then put in the fridge where it sets. The moulds may differ by the length and width of their sides but are filled to different heights so that they all have the same volume; length, width, and height are always integer numbers.

Unfortunately, one of the cleaners loves practical jokes! Whenever he can, before the jelly has set, he tips liquid jelly from one of the moulds into another. He is happy if he succeeds just once and doesnt repeat the joke with other moulds.

Your task is to help the school staff by preparing a report for them. They need to know who has lost jelly and who has gained it so that they can correct matters before the children arrive.

Input

The input consists of one or more scenarios. Each scenario begins with a single integer n, $1 \leq n \leq 100$, representing the number of children for whom jelly was prepared. Following this are n lines, each line representing one child. The data for a child consists of the child's name and 3 integer numbers in the range 1 to 100, respectively representing the length, width and height of the jelly in that child's mould, all separated by single spaces. A child's name consists of a sequence of 1 up to 10 letters (upper and/or lower case), and no two children have the same name. A single 0 on a line by itself marks the end of input.

Output

Your report consists of one line of text per scenario. If the cleaner did not manage to transfer any jelly before it set, your report must say

No child has lost jelly.

If the cleaner did manage to transfer jelly, your report must be of the form

ChildA has lost jelly to ChildB.

where ChildA is the actual name of the child that has lost jelly and ChildB is the actual name of the child that has gained jelly.

Sample Input

```
3
Joe 10 10 2
Susan 10 5 4
Bill 5 5 8
4
```

```
Zoe 10 2 2
Lee 6 5 2
Alan 5 4 4
Tommy 12 5 1
0
```

Sample Output

```
No child has lost jelly.
Zoe has lost jelly to Alan.
```

C: House Numbering

Time Limit: 2 sec

The government of Acmonia has decided that henceforth all house numbers should be given in binary instead of decimal notation. Householders will now have to purchase 0 and 1 binary digits to display on their houses. For reasons much too complicated to discuss here it seems that the cost to a householder of a 0 binary digit and of a 1 binary digit may well differ. Your task is to write a program which will report to householders the cost of their new numbers.

Input

The input text consists of a number of sets of problems. The first line of a set is of the form "COST a b". For that set:

- a and b are both integers, $0 \leq a, b \leq 1000$
- a 0 binary digit costs a dollars
- a 1 binary digit costs b dollars

The first line is followed by one or more lines each consisting of a single integer n

- $0 \leq n \leq 2\ 000\ 000$
- n indicates a house number, expressed as a standard decimal number

A single # on a line indicates the end of input.

Output

Each set of output data must begin with a single output line showing consisting of the word "Set", followed by a space and the current set number (counted from 1). This is followed by the cost of the binary digits for each house number, each cost being displayed as a decimal number on a separate line.

Sample Input

```
COST 1 1
1
34
15
COST 1 10
1
34
15
COST 10 1
1
34
15
COST 0 5
1
16
#
```

Sample Output

```
Set 1
1
6
4
Set 2
10
24
40
Set 3
1
42
4
Set 4
5
5
```

D: Prime Digital Roots

Time Limit: 2 sec

The *digital root* of a number is found by adding together the digits that make up the number. If the resulting number has more than one digit, the process is repeated until a single digit remains.

Your task in this problem is to calculate a variation on the digital root - *a prime digital root*. The addition process described above stops when there is only one digit left, but will also stop if the original number, or any of the intermediate numbers (formed by addition) are prime numbers. If the process continues and results in a single digit that is not a prime number, then the original number has no prime digital root.

An integer *greater than* one is called a prime number if its only positive divisors (factors) are one and itself.

- For example, the first six primes are 2, 3, 5, 7, 11, and 13.
- Number 6 has four positive divisors: 6, 3, 2, and 1. Thus number 6 is *not* a prime.
- Caveat: number 1 is *not* a prime.

EXAMPLE OF PRIME DIGITAL ROOTS

1	This is not a prime number, so 1 has no prime digital root.
3	This is a prime number, so the prime digital root of 3 is 3.
4	This not a prime number, so 4 has no prime digital root.
11	This is a prime number, so the prime digital root of 11 is 11.
642	This is not a prime number, so adding its digits gives $6 + 4 + 2 = 12$. This is not a prime number, so adding again gives $1 + 2 = 3$. This is a prime number, so the prime digital root of 642 is 3.
128	This is not a prime number, so adding its digits gives $1 + 2 + 8 = 11$. This is a prime number, so the prime digital root of 128 is 11.
886	This is not a prime number, so adding its digits gives $8 + 8 + 6 = 22$. This is not a prime number, so adding again gives $2 + 2 = 4$. This is not a prime number, so 886 has no prime digital root.

Input

The input will contain a single integer on each line in the range 0 to 999 999 inclusive. The end of the input will be indicated by the value 0.

Output

If the input number has a prime digital root, then the input number must be output right aligned with a field width of 7. It must be followed by a single space, and then by the calculated prime digital root also right aligned with a field width of 7.

If the input number has no prime digital root, then the input number should be output as defined above followed by 4 spaces followed by the word none (in lowercase). The terminating zero should not be output.

Sample Input

```
1
3
4
11
642
128
886
0
```

Sample Output

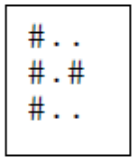
```
      1    none
      3       3
      4    none
     11      11
    642       3
    128      11
    886    none
```

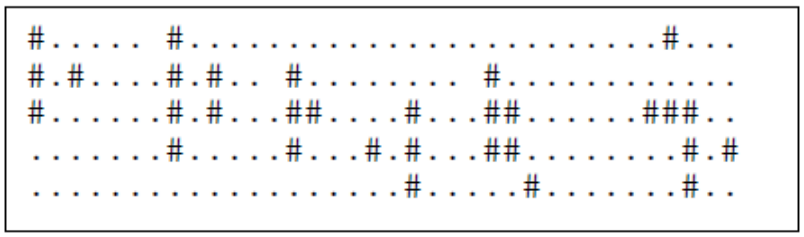
E: Desert Bitmap

Time Limit: 2 sec

This problem requires you to search a black and white satellite image of a desert for a secret building complex with a given shape. A complex of this given shape may host an installation for producing the strategic xeenium macgillicudamate ingredient, and must keep its orientation with regard to cardinal axes (North-East-South-West). Rotations and mirror images are not allowed because they would interfere with the delicate alchemy required for the production process. You must determine how many times the given complex may possibly occur in the image.

Consider the following images, both on the same scale, where a # (sharp) is a "black" pixel representing a part of a building, and a . (dot) is a "white" pixel, representing sand. On the left is an image of the complex you are trying to locate, on the right is an image of the desert with some buildings on it.





- How many possible locations for the given secret buildings do we count?
- The answer is *four*: one at the top-left corner, two overlapped possibilities to its right, and one in the bottom right. The shapes near the top-right corner, and in the centre bottom don't count because they are rotated (remember that rotated and/or mirrored images do not count).
- Note that, as this answer implies, the sand pixels in the image of the building complex simply establish the necessary relationships between the building parts. In the actual image they may contain *either* sand *or* other building parts (possibly for disguising the true nature of the complex).
- *Assume* that images representing strategic complexes are already trimmed of any unneeded dot "white" pixels on the edges, i.e., these images will always contain *at least one # character on each edge* (as our example shows). An edge here is the first or last row or column.

Input

Each problem will give you the specification for the building complex image followed by the specification for the desert image. There may be several problems in the input data, which will be terminated by a line containing just 0 0.

In each problem the input is:

- Line 1: 2 positive integers, $B1$, $B2$, respectively representing the number of lines and the numbers of columns in the following Buildings image. Both numbers will be in the range 1 to 16 inclusive.

- Next $B1$ lines: $B2$ characters (# or .) on each line to represent part of the image of the building complex.
- Next Line: 2 positive integers, $D1$, $D2$, respectively representing the number of lines and the numbers of columns in the following Desert image. Both numbers will be in the range 1 to 64 inclusive.
- Next $D1$ lines: $D2$ characters (# or .) on each line to represent the desert image.

Output

The output for each test case consists of a single integer value on a line by itself being the number of matches found.

Sample Input

```
2 2
#.
##
3 5
#.#.#
#####
.###.
1 3
#.#
3 6
##..##
.#.#.#
#.#...
3 3
#..
#.#
#..
5 36
#......#.........................#...
#.#....#.#...#.........#............
#......#.#...##....#...##......###..
.......#.....#...#.#...##........#.#
...................#.....#.......#..
0 0
```

Sample Output

```
4
3
4
```

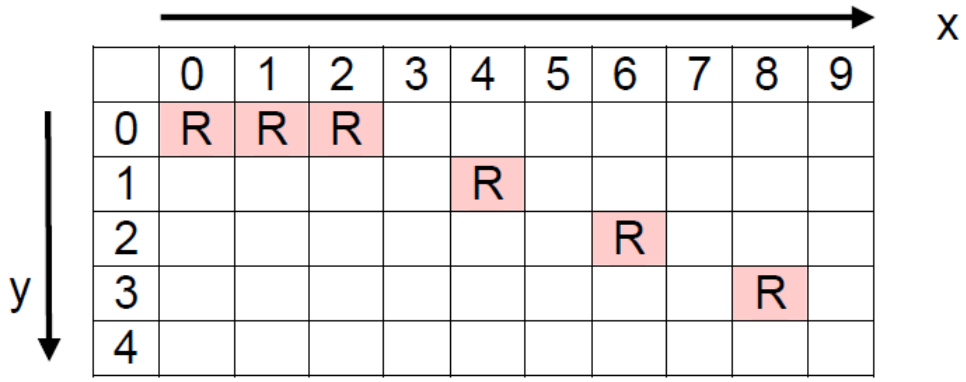
F: Spots

Time Limit: 5 sec

Painting tiles is hard work. John has convinced his two sons Raul and George to do a painting job, by agreeing to pay one dollar per tile painted by each of them. Raul will paint red tiles and George green tiles at the places indicated by their father. However, they are still not convinced: How are they going to divide up the area to be painted, what happens if both want to paint the same tile at the same time, what are the rules, and, last, but not least, how much will they receive in the end?

To avoid arguments and to have some fun, John would like to show them a simulation of the problem before the hard work begins. The simulation will start with each son at opposite ends of a rectangular grid of width N and height M ($N, M \geq 2$), Raul at (0,0) and George at $(N-1, M-1)$. The two sons will begin by painting a spot in their respective colours on their starting tiles. Next, Raul and George are each given a series of individual instructions for moving to the next tile to paint. Each move can be repeated one or more times and is defined by a pair of increments along the horizontal x-axis and vertical y-axis, in this order; these increments can be positive, negative, or zero.

For example, assume that Raul is on his initial tile (0,0) on a grid with $N = 10$, $M = 5$, and receives the instructions to hop 2 times in the direction of (1,0) and then 3 times in the direction of (2,1). He will begin by painting a red spot on the tile (0,0). Then, according to these instructions, he will successively land on and paint red spots on the following tiles: (1,0), (2,0), (4,1), (6,2), (8,3). The following diagram uses the letter 'R' to mark the tiles spotted in red by Raul according to his instructions:



At each simulation step the two sons move in lock-step, each hopping according to his own instructions. To avoid conflicts, each time Raul and George are about to hop to the next tile the simulation must check whether they would land on the same tile. If so, the simulation ends without them moving, and the landing tile remains painted in its previous colour, if any. Otherwise, the simulation will end as soon as one of the sons ends his instructions.

Each tile can contain only one colour spot, either red or green. Since it is possible that Raul and George land on the same tile at different times, the simulation should only count the tile towards the son landing there last.

To avoid Raul and George falling off the edge of the tile grid, they are allowed to wrap around, in all directions. For example, for a grid of the same size as above, one hypothetical move from (8,3) in the direction (2,3) results in the tile (0,1).

Your task is to write a program that computes the results of such a simulation given the grid size and the instructions for each son.

Input

The input consists of one or more scenarios. Each scenario consists of 3 lines. The first line contains two numbers separated by a space, N and M, $2 \leq N, M \leq 1000$, respectively representing the width and the height of the grid. The second line contains the instructions for Raul and the third line the instructions for George. Each instruction line starts with a number C, in the range 1 to 100, followed by C groups of 3 numbers. In each group the first number is a repetition count in the range 1 to 1,000,000,000, the second number is an increment along the x-axis, and the third number an increment along the y-axis - both increments are in the range -1,000 to 1,000. All numbers are separated by single spaces. The end of the input is indicated by a "grid of size 0", i.e., a '0' on a line by itself.

Output

Output one line for each input scenario. Each output line should show two numbers separated by a single space, representing the earnings of Raul and George, in this order.

Sample Input

```
10 5
2 2 1 0 3 2 1
3 4 -2 0 1 0 -9 2 1 0
10 10
2 1 15 5 1 0 0
2 1 0 0 2 -4 -4
10 7
2 1000 2 -1 1000 0 -1
2 1000 -1 0 1000 -1 0
10 10
5 1000 2 -1 1000 1 0 2 5 5 3 1 1 10 1 1
5 1000 -1 0 1000 0 1 3 -4 -4 2 -1 -1 10 -1 -1
0
```

Sample Output

```
5 6
2 1
30 10
18 18
```

G: Mobiles

Time Limit: 5 sec

ACMIA mobile phones have a shortcut mode for typing text messages using the numerical phone keypad. In this mode, the system uses a dictionary of known words. After a sequence of digits is entered the system checks for and displays all possible matches in the dictionary. The ACMIA phone keypad for the English alphabet is as follows:

1	2 abc	3 def
4 ghi	5 jkl	6 mno
7 pqrs	8 tuv	9 wxyz
	0 (space)	

Your task is to write a program that displays all possible matches for given digit sequences, using a given dictionary.

A digit sequence corresponds to a sequence of words, with zero digits ('0') indicating spaces. Leading and trailing zeros are ignored, and multiple consecutive embedded zeros are treated as a single zero. For each sequence of non-zero digits, display the matching word from the dictionary. When more than one match is available, display all matches in dictionary order between round parentheses and separated by bars ('|'). If there is no matching word, display a sequence of asterisks ('*') of the same length. For example, with a dictionary consisting solely of the words `i`, `loud`, `love`, `programming`, the digit sequence

```
00405683000776472664640777 70
```

will be displayed as the text

```
i (loud|love) programming ****
```

Input

The input will consist of one or more scenarios, each scenario consisting of a dictionary of permitted words and a series of digit sequences to be interpreted as text messages.

The dictionary consists of 1 to 1,000 words, one word per line, in increasing dictionary order, with no duplicates. Each word consists of 1 to 30 lowercase letters. For any given non-zero digit sequence there will be no more than 10 matching words in the dictionary. The end of the dictionary is indicated by a line consisting of a single '#'.

The digit sequences to interpret as text messages follow the dictionary, one per line. Each message line consists of 1 to 100 digits, with at least 1 non-zero digit. The end of messages is indicated by a line consisting of a single '#'.

The end of input is indicated by an empty dictionary (a dictionary with zero words).

Output

For each scenario output a line consisting of the word 'SET' (all uppercase) followed by a space and then the scenario number, starting with 1. Following this output the list of interpreted text messages, one message per line.

Sample Input

```
i
loud
love
programming
#
00405683000776472664640777770
#
a
game
go
golf
good
hand
hold
hole
home
in
me
of
to
#
2046630426306304653
46086020466304663
#
#
```

Sample Output

```
SET 1
i (loud|love) programming ****
SET 2
a (good|home) (game|hand) (me|of) (golf|hold|hole)
(in|go) to a (good|home) (good|home)
```

H: Encryption Scheme

Time Limit: 2 sec

Most text encryption schemes use a secret key string to convert the plain text to the enciphered text in some way. A novel method being tested by the Australian Security Service consists of a transformation of a key string K into a target string P using block moves. Each block move is of the form copy(start, length), where start indicates a position in K and length is the number of characters to be copied from K to P. Since the idea is to eventually transmit only the block moves, the principle is to use as few block moves as possible. For example if:

```
K: abaabba
P: aaabbbabbbaaa
```

Assuming that here string positions start with 1, two shortest block move sequences would be:

copy(3,2);copy(4,3);copy(2,2);copy(5,2);copy(2,3);copy(1,1)

or

copy(7,1);copy(3,3);copy(5,2);copy(4,2);copy(5,3);copy(3,2)

The actual shortest block move sequences are not unique but the minimum number is 6 in this case. If the moves are now transmitted, then it is possible to construct the plaintext message P from the key string K.

The Australian Security Service is now automating this procedure, so given K and P they need to count the minimum number of block moves from K to P. To make things simple at the beginning, they are considering strings comprised of lowercase letters and digits. The set of characters within string P is a subset of the set of characters of the key string K.

You are to help the Australian Security Service by writing a program to get two strings K and P as above, and print the minimum number of block moves from K to P.

Assume that each of K and P is made up of 1 to 120 characters (K is allowed to be longer than P).

Input

Input will consist of a sequence of lines. Odd lines are to be used as the key strings K, and even lines to be used as target strings P. The input will be terminated by a '#' by itself in the place of a K string.

Output

The output will consist solely of the minimum number of block moves for each pair, one result per line.

Sample Input

```
abaabba
aaabbbabbbaaa
xy0z
zzz0yyy0xxx
#
```

Sample Output

```
6
10
```

COMMENTS

The first sample is discussed on the first page. Here follows a minimal sequence of block moves for the second sample:

copy(4,1);copy(4,1);copy(4,1);copy(3,1);
copy(2,1);copy(2,1);copy(2,2);
copy(1,1);copy(1,1);copy(1,1)

I: Bouncy Balls

Time Limit: 2 sec

The Department of Defence of a certain country (No, not Australia or New Zealand), in conjunction with the Department of Police, have devised a brilliant method of saving money on crowd control really bouncy rubber bullets. They had noticed that the rubber bullets they had been using were largely being wasted those that hit anyone or anything usually just fell to the ground, whereas if they were really, really bouncy, they would bounce off and possibly hit several more people before their energy was spent.

They decided to test this idea by building a special circular test rig. The bullet would be fired into the rig horizontally and at some predetermined angle to the tangent to the rig at that point. It would have sufficient energy to travel some considerable distance before stopping. (You may assume cartoon physics, i.e. that it travels horizontally until it reaches the end of its travel, at which time it drops to the floor.) However, as so often happens with lucrative defence contracts, the contractor made off with the money, so they decided to simulate the whole process on a computer. This is where you come in.

Write a program that will read in details of a test rig and a series of test firings and determine how many times the rubber bullet would bounce before it stops. You may assume that the bullet is a point and that, because of problems in determining the exact sequence of events, any test firing where the bullet stops within 1 mm of the rig is deleted from consideration.

Input

Input will be a series of tests, each test consisting of a series of test firings. Each test starts with an integer specifying the radius of the test rig in millimetres and a value of 0 for the radius terminates the input. Each test firing occurs on a line by itself and consists of a distance in millimetres (between 100 and 10,000 inclusive) that the bullet will travel, and an angle, in degrees, (between 10 and 170 inclusive, where 90 means directly towards the centre of the rig). The series of test firings will be terminated by a line containing two zeroes (0 0).

Output

For each test rig, output a line with the words "Test Rig" followed by a space and then the number of the test rig (a running number starting at 1) followed by a series of lines, one for each test firing for that rig with each line giving the number of times a bullet bounces off a wall before it stops. This number is to be written without any leading or trailing spaces. A blank line should appear between test rigs. Follow the example given below.

Sample Input

```
100
1000 23
1200 47
0 0
0
```

Sample Output

```
Test Rig 1
12
8
```

J: Lollies

Time Limit: 5 sec

Every day on his way home, little Billy passes by his great aunt Clara Mitchum's house. Generally he stops in for a chat with the great ACM (as he lovingly refers to her) and sometimes he asks for some lollies. When he does, she generally gives him some, but then adds "now don't be asking for any more for another N days" where N is some positive integer. If $N = 1$ that means he can ask for some on the next day, but for example if it is April 6 and $N = 4$ then he must wait until April 10 or later before asking for more lollies.

One day Billy happened to catch sight of the great ACM's calendar, and noted that each day was marked with two integers. He also noted that the first of these referred to the number of lollies the great ACM would give him on a particular day, and the second to the delay that would then be required before making another request. He copied down as much of the information as he could, and has passed it to you to analyse. His objective, of course, is to get as many lollies as he can.

Your task is to write a program which will report the total number of lollies that can be obtained by Billy, and provide a schedule for obtaining that amount. In the event that there are two or more ways to obtain the maximum number of lollies, Billy will choose the one where his first collection is as late as possible, and among all collections with that first date, his second collection is as late as possible, and so on.

Input

The input text consists of a number of sets of unrelated problems. The first line of a set is a problem title consisting of a string of 1 to 20 letters. A single # on a line indicates the end of input.

The "title" line is followed by a sequence of "day" lines. Each problem set contains between 1 and 100 days, including the limits. In the given order, the first "day" line corresponds to day number 1, the second line to day number 2, the n^{th} line to day number n. Each "day" line consists of two integers separated by a single space:

- an integer L, which is the number of lollies available on that day ($1 \leq L \leq 100$),
- an integer N, which is the associated delay ($1 \leq N \leq 100$).

Conventionally, a delay N pointing to a day beyond the end of the current problem refers to a day with zero lollies and zero further delays ($L = 0, N = 0$).

Output

Each report must follow the format outlined in the Sample Output section below (use single spaces for spacing).

In < *problem_title* > < *total_amount* > < *lollies* > can be obtained: On day < *day_number* > collect < *day_amount* > < *lollies* >. On day < *day_number* > collect < *day_amount* > < *lollies* >. ...

In this notation, < *problem_title* > represents the actual problem title, < *total_amount* >, < *day_amount* >, and < *day_number* > are numbers with self-described meaning, and < *lollies* > stands for either "lolly" or "lollies", as required by the context (the singular and plural forms must be used appropriately). Days must be given in increasing sequence numbers. Each group report should be separated from the next by a blank line.

Sample Input

```
January
1 1
2 2
3 3
February
10 3
7 1
5 2
1 1
March
2 3
1 1
3 7
2 7
#
```

Sample Output

```
In January 4 lollies can be obtained:
On day 1 collect 1 lolly.
On day 3 collect 3 lollies.
In February 12 lollies can be obtained:
On day 2 collect 7 lollies.
On day 3 collect 5 lollies.
In March 4 lollies can be obtained:
On day 2 collect 1 lolly.
On day 3 collect 3 lollies.
```

K: Spreading Gossip

Time Limit: 5 sec

We have a remote village with $n \leq 20$ houses $(h_0, h_1, h_2, \ldots h_{n-1})$ and several secure telephone lines linking neighbouring houses (there is exactly one line between each pair of neighbouring houses). For any pair of houses h_i and h_j there is at least one path of telephone lines connecting them (this can be viewed as an undirected connected graph with houses as vertices and lines as edges).

Gossip can travel over telephone lines. Each house can call at most one neighbour house at a time. Calls may begin at the beginning of each hour (e.g., 9 am, 1 pm, 6 pm, etc.), and last for exactly one hour. The local telephone company charges a fortune for each call, but has a quirk that any number of calls can be made in parallel at the same price as any single call.

Given this scenario, we want to find the minimum total price (minimum number of used hours) to disseminate some gossip from house h_0 to all other houses.

Input

The input involves a series of scenarios. Within each scenario the first line has an integer number n, the number of houses. This first line is followed by n other lines, one for each house, in the order h_0, h_1, h_2, $\ldots h_{n-1}$. Each "house" line contains a list of indices of its neighbouring houses (in no particular order), separated by single spaces.

The series is terminated by a scenario with $n = 0$, which isn't processed.

Output

The output must be "`Village` s`:` p", where s is the scenario sequence number starting at 1 and p is the answer for each input village (use single spaces as separators, i.e., one space after the word "`Village`" and another space after the colon "`:`").

Sample Input

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

Sample Output

```
Village 1: 2
Village 2: 4
```

L: Swiss Draw

Time Limit: 2 sec

Many sports and games hold tournaments to determine at least a winner and, very often, a ranking or ordering as well. In two player games (such as Tennis, Chess and Scrabble), the two most common forms of tournament are 'knockout' (usually based on an initial ranking or 'seeding') and 'round robin' (where everybody plays everybody else). The disadvantage in knockout is that a promising newcomer could meet a very much stronger player early in the tournament and not reach their true position. Round Robin eliminates this but at a huge cost in time - a Round Robin involving 128 players needs 127 rounds whereas it would take only 7 rounds in a knockout competition.

An alternative known as Swiss Draw is very popular in games such as Scrabble. To maximize competition, any one player will play any other player no more than once. After each round, players are ranked on the number of games they have won, where a draw is equal to half a win (more is better) and, within that, by 'spread' - the cumulative difference between their scores and their opponents' scores (again bigger is better). If by chance two or more players tie in this ranking then they appear in inverse order of their previous ranking, i.e. the initially lower-ranked players move ahead. In each round each player either plays someone above them or the highest ranked player below them that allows everyone to play someone they have not played before. The input will specify the (usually random) ordering before the first game.

Write a program to determine the final ranking of a group of Scrabble players, given the initial draw and the scores for each individual for each round.

Input

Input will consist of one or more scenarios. The first line of each scenario will consist of two integers, P and R, ($16 \leq P \leq 64, 4 \leq R \leq P/4$) denoting the number of players (a multiple of two) and the number of rounds respectively. This will be followed by P lines, each line consisting of a name (a string of 1 through 20 alphabetic characters without any spaces) followed by R integers (separated from each other and the name by at least one space) representing the R scores for that individual. The list will be in the initial order of play, thus in the first round player $2n+1$ played player $2n+2$ ($0 \leq n < P/2$). Input will be terminated by a line containing two zeroes (i.e. P and R both zero).

Output

Output will consist of a list of all the players ranked according to the above criteria, together with the number of wins and the spread. Note that a draw is counted as half a win, so indicate an odd number of draws by a plus sign (+) after the number of wins. The name is left justified in a field of width 20, the number of wins is right justified in a field of width 3, specification of draws occupies 1 character position and the spread is right justified in a field of width 6. Leave one blank line between scenarios.

Sample Input

```
16 4
Absalom 280 334 319 426
Betsheba 374 514 459 417
Carolyne 318 415 445 481
Davidian 402 361 375 278
Eleanor 425 302 447 522
Frances 425 513 306 327
Gabriel 330 337 365 398
Hermione 539 254 442 450
Ishmael 485 305 540 522
Jeremiah 288 295 367 476
Kenneth 532 304 452 445
Laurence 426 437 260 474
Meredith 438 489 274 475
Nicholas 307 357 380 482
Octavia 426 498 305 497
Patricia 333 253 370 412
0 0
```

Sample Output

```
Ishmael 4 619
Meredith 3 247
Carolyne 3 186
Betsheba 3 158
Kenneth 3 126
Eleanor 2+ -11
Hermione 2 264
Laurence 2 -93
Nicholas 2 -114
Davidian 2 -150
Frances 1+ -119
Octavia 1 -85
Absalom 1 -187
Jeremiah 1 -188
Gabriel 1 -338
Patricia 0 -315
```

M: Maze Madness

Time Limit: 5 sec

You have been placed somewhere in a maze and you wish to escape by the shortest possible route. Fortunately you have been given a map of the maze. Before setting off, you wish to calculate the distance you need to travel. Your task is to write a program that will calculate the shortest distance to leave the maze. Note that there may be more than one exit and the specified start position could be at any location within the maze.

The maze is set on a grid that has M columns and N rows, with $1 \leq M, N \leq 100$. Some squares of this grid have impenetrable walls of negligible thickness between them (or on their outside border). You may move from any square to a horizontally or vertically adjacent square (possibly outside the maze, thus escaping) provided that there is no wall between them. Each single move between squares adds 1 meter to the distance travelled.

Input

The input involves a series of scenarios. Within each of the scenarios the first line has the size of the maze. This is given as two numbers M and N. Then the maze is drawn on $2 * N + 1$ lines and $2 * M + 1$ columns using the printable characters "-", "|", "+", ".", " " (space), and "s":

- "|" is used for "vertical" walls,
- "-" is used for "horizontal" walls,
- "+" is used to indicate boundaries between rows and columns (there are always $(N+1) * (M+1)$ of these),
- "." is used for wall openings,
- " " (space) is used for empty squares,
- "s" is used to show your start location (there is exactly one "s").

A line with "0 0" indicates the end of the scenarios.

Output

Output a single line for each of the scenarios. This line should contain either "`Maze` i`:` d" or "`Maze` i`: No escape!`", where i is the scenario number (counting from 1) and d is the minimum distance (in meters) needed to escape (use single spaces for spacing).

Sample Input

```
1 1
+-+
|s.
+-+
3 2
+-+-+.+
| .s| |
+-+.+-+
| . . .
+-+-+-+
5 6
+-+-+-+-+-+
| . . . . |
+-+-+.+-+-+
| |s. . . |
+.+-+-+-+.+
| | . . . |
+.+-+-+-+.+
| | . . . |
+.+-+.+-+.+
| . . | | |
+-+-+.+.+.+
. . . | . |
+-+-+-+-+-+
3 2
+-+-+.+
| .s| |
+-+-+-+
| . . .
+-+-+-+
0 0
```

Sample Output

```
Maze 1: 1
Maze 2: 3
Maze 3: 12
Maze 4: No escape!
```

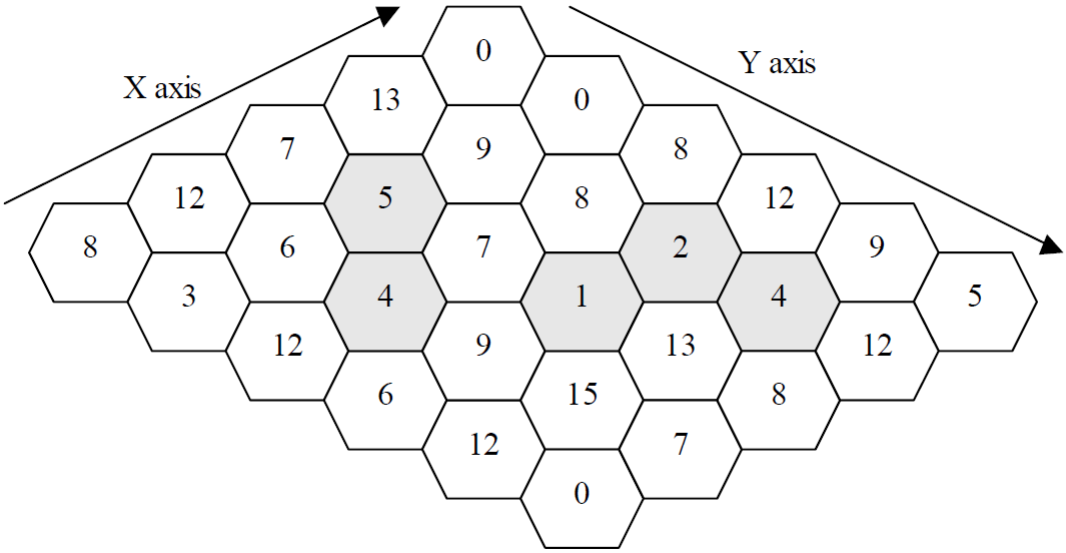
N: Basalt Buckets

Time Limit: 5 sec

Under certain conditions volcanic basalt forms large crystals, like hexagonal pillars. At Fingal Head, at the border of New South Wales and Queensland, there is a dramatic example, called Giant's Causeway, where a peninsula formed of such columns juts into the Pacific. It is particularly dramatic when the big Pacific rollers break on the causeway, leaving streams of water cascading over the basalt pillars.

Your task is to find out how much water could collect in hexagonal hollows, formed when some pillars are shorter than others, and can act as wells.

Here is a diagram of a set of hexagons. Each hexagon has an integer height. Water can always cascade off the edge of the set of hexagons, but it will collect in the five shaded hexagons, since they form wells completely surrounded by higher hexagons. Water drains from the left hand pair over two pillars of height six, and from the three on the right water drains over a pillar of height seven. Assuming each hexagon has unit area, the volume of water that can collect is 17 units.



Your program must handle input as a series of problem descriptions.

Input

Each problem begins with two integers X and Y, on a line by themselves, in the range 1 to 200 inclusive, giving the number of hexagons along an X-axis, and along a Y-axis, as shown in the diagram. Then there are given a further $X * Y$ integers, in the range 0 to 5,000 inclusive, which are the heights of the hexagons. The order of input heights is given as Y rows of X integers, but they may be split across lines arbitrarily. The given diagram is described by input as given in the Sample Input section.

Output

The output must consist only of one integer for each problem given, on a line by itself with no spaces, giving the volume of water that could collect for that set of columns.

Sample Input

```
5 6
8 12 7 13 0 3 6 5 9 0 12
4 7 8 8 6 9 1 2 12 12 15 13 4 9 0 7 8 12 5
0 0
```

Sample Output

```
17
```

O: Four in a Line

Time Limit: 5 sec

Four in a Line is a game similar to 3-dimensional noughts and crosses. It consists of a horizontal table on which 16 pegs, each of which can hold 4 beads, are arranged in a 4 x 4 grid. Each player has a supply of either green or red beads which are placed on the pegs in turn, starting with red. Obviously, as each bead is placed on a peg, it slides down as far as it can - until it either hits another bead or the supporting table. The winner is the first to get 4 beads of their colour in a line (hence the name). The line can be in any plane and in any orientation, as long as the four beads are all of the same colour and form a straight line.

As with most games, the interesting part comes towards the end, when each player (colour) is attempting to build a line and block the opponent's incipient lines. Write a program that will read in details of a game position and determine whether green (the next player) can be guaranteed to win the game within 5 *plies*. A *ply* is half a turn, in this situation placing one bead, thus 5 plies implies three moves by green and two by red.

Input

Input consists of a number of games. Each game consists of 4 lines of characters, each line consisting of 4 blocks of 4 characters – 'R' for red, '' for green or '' for empty - where each block represents the contents of a single peg with the left end representing the bottom. Thus the block 'GRR' represents a peg with a green bead on the bottom with two red beads above it. Note that the entire state of the game is always given, thus the starting state would consist of 64 '' characters arranged in 16 blocks of 4. You can assume that the position is valid, i.e. that there will be exactly one more red bead than green beads, and that there will not be any 'holes' in the description (the 'block' GRG, for instance). There will be one blank line after each game and input will be terminated by a line containing only a single ''.

Output

For each game description in the input, output a single line of the form "Green can win in N move(s)", where $1 \leq N \leq 3$, and where N is the smallest such number, or "Green cannot win in 3 moves". Use the singular form when $N = 1$ and the plural form otherwise.

Sample Input

```
GGG# RR## R### R###
#### #### #### ####
#### #### #### ####
#### #### #### ####

RRR# RGG# RGG# ####
#### #### #### ####
#### #### #### ####
#### #### #### ####

#
```

Sample Output

```
Green can win in 1 move
Green cannot win in 3 moves
```