Problem 1: Are We There Yet?

A palindrome is a word or sentence that has the same sequence of letters and/or digits when it is read forward or backward; spaces, punctuation, and capitalization are usually ignored. For example, "Aha! Mob Omaha!" and the very short "Lion Oil" are palindromes.

Consider a six-digit odometer in a car that displays the total number of miles the car has been driven; no digits are displayed for fractional miles. Also ignore leading non-significant zeros in the odometer reading, so that "002345" is considered the same as "2345". With these rules, some odometer readings, such as "002332" or "056765" are palindromes. Also note that some of these palindromic numbers have an even number of digits and others have an odd number of digits.

Given the current odometer reading on a car, calculate the number of miles the car must be driven to cause the odometer to display a palindrome.

Example

The current odometer reading is 005432. The next odometer reading that will be a palindrome is 005445, so the number of miles that must be driven is 13.

The problem

Write a program that reads an odometer reading and displays the number of miles to the next odometer reading that will be a palindrome, and that palindrome.

Input

There will be an arbitrary number of odometer readings in the input, each given as a line containing 6 decimal digits and the end of line character. A line containing **`-1`** and the end of line character will follow the last odometer reading.

Output

For each odometer reading in the input, calculate the number of miles to the closest odometer reading that is a palindrome. Display the input case number (**`1`**, **`2`**, …), the number of miles that must be driven, and the corresponding palindromic odometer reading. Follow the format of the output shown below in the sample.

Sample Input

```
005432
000012
100002
000005
-1
```

Output for the Sample Input

```
Case 1: 13 miles to 005445.
Case 2: 10 miles to 000022.
Case 3: 1099 miles to 101101.
Case 4: 0 miles to 000005.
```

Problem 2: Fast Times at Ridgemont's Planet

NASA has established communications with a new planet discovered by Dr. Ridgemont. This planet, coincidentally, has a calendar very similar to the calendar we use here on Earth. But the minutes, hours, days, and so forth are all of a different duration. For example, on Ridgemont's Planet, a minute might last 54 seconds, there might be 61 minutes per hour, 23 hours per day, and 340 days per year. The year is split into months, and as on Earth, each month may have a different number of days. The inhabitants have set the calendar this way so that there is no need to handle leap years or daylight savings time, but other than that everything is different.

NASA software engineers want to practice good code reuse, so naturally they want to port their Linux source code into the communications system they use to talk to the inhabitants on this distant world. Much of their software utilizes the function "difftime", which takes two specifications of a date and time and returns the number of seconds between them. For example, on Earth, if I ask for the difference between 12:35:00 on one day and 12:35:15 on the next day, the answer is 86,415. This represents one day's worth of seconds (24 hours times 60 minutes times 60 seconds is 86,400) plus an additional 15 seconds. All date/time specifications are expressed as a string with the format "**`MM/DD/YY-HH:MM:SS`**".

Intuitively, the procedure used by difftime is to determine how many complete days have elapsed between the input specifications and add in the correct number of seconds. Then determine how many more hours are needed, then how many more minutes are needed, and so forth.

Example

Suppose that the planet has 13 months and the number of days in each month as shown in the table below. There are a total of 399 days in a year.

Month	1	2	3	4	5	6	7	8	9	10	11	12	13
Days	30	30	31	31	31	31	31	31	31	31	31	30	30

The planet has 23 hours per day, each hour has 53 minutes, and each minute has 61 seconds.

A NASA engineer wants to know the difference between 02/29/11-22:30:05 and 03/01/11-10:00:00. The calculation performed by difftime might proceed as follows:

Description	Seconds on the Planet
02/29/11-22:30:05 to 02/29/11-22:31:00	56
02/29/11-22:31:00 to "midnight" 02/30/11-00:00:00	1342
02/30/11-00:00:00 to "midnight" 03/01/11-00:00:00	74359
03/01/11-00:00:00 to 03/01/11-10:00:00	32330
Total number of seconds	108087

And in this case the function returns 108087 "planet seconds".

The problem

Write a program that reads the characteristics of time on a distant planet and two date/time specifications, and then prints the difference between the two times. You are guaranteed that the first date/time specification is not after the second time.

Input

There will be an arbitrary number of cases to consider. The input for each case will be entirely contained in a single line of input. That line will contain the following data items in the order specified, separated by spaces. The allowable range of values for each item is given in parentheses following the item.

- The number of seconds in a minute (1..100)
- The number of minutes in an hour (1..100)
- The number of hours in a day (1..100)
- The number of months in a year (1..99), followed by that many integers, each of which represents the number of days in the corresponding month (1..99)
- A start date/time specification in the format MM/DD/YY-HH:MM:SS
- An end date/time specification in the format MM/DD/YY-HH:MM:SS

Like on Earth, months and days are numbered starting with 1, and years, hours, minutes, and seconds are numbered starting with 0.

The line following the input for the last case will contain a single zero.

Output

For each input line, determine the difference between the start and end date/time specifications, in seconds; the largest difference will be less than 4294967296. Then display the input case number (1, 2, …) and the difference. Follow the format of the output shown below in the sample.

Sample Input

```
61 53 23 13 30 30 31 31 31 31 31 31 31 31 31 30 30 02/29/11-22:30:05 03/01/11-10:00:00
0
```

Output for the Sample Input

```
Case 1: 108087 seconds
```

Problem 3: Symbolic Mathematics

Over the years software systems that can perform symbolic computation have improved considerably. Popular commercial systems currently include Mathematica and Maple; the Sage system is also freely available. In this problem you're just going to implement a small part of such a system: the ability to compute (symbolically) f(g(x)) for the functions f(x) and g(x) expressed as polynomials with terms having non-negative integer exponents and integer coefficients.

Input

Polynomials will be expressed in the input and output as a sequence of terms followed immediately by the end of line. Each term consists of a plus or minus sign (except the first term doesn't have a sign if it is plus), an integer coefficient (omitted if it is 1 and the term's exponent is non-zero), the lowercase letter `'x'`, a circumflex (`'^'`), and a non-negative exponent. If the exponent is 1, then the circumflex and the exponent are omitted. If the exponent is 0, the `'x'` is also omitted. Terms are given in decreasing order of their exponents, and each term must have a unique exponent. The table below shows a few example polynomials in this form along with their traditional algebraic representation.

Input/Output Format	Algebraic Format
`12x^2-3x+5`	$12x^2 - 3x + 5$
`-3x^4+x^3-5x-1`	$-3x^4 + x^3 - 5x - 1$
`x^99+2000x^31-450x^12-21x`	$x^{99} + 2000x^{31} - 450x^{12} - 21x$

The input will contain pairs of lines. The first line of each pair gives the polynomial f(x), and the second line of each pair gives the polynomial g(x). The input line after the last pair of polynomials will contain only an end of line character. Blanks and tabs will never appear in the input.

Coefficients and exponents in the input and output polynomials will never exceed 1000000 in absolute value.

Output

For each pair of input lines, display the input case number (1, 2, …), the input polynomials, and the polynomial representing the value of f(g(x)). Label the polynomials appropriately. Display a blank line after the output for each case. Follow the format shown in the samples below.

Sample Input	Output for the Sample Input

`x+1`
`4`
`2x^2-x+5`
`-3x^3+x`
(a line feed character)
	`Case 1:`
	`  f(x) = x+1`
	`  g(x) = 4`
	`  f(g(x)) = 5`
	
	`Case 2:`
	`  f(x) = 2x^2-x+5`
	`  g(x) = -3x^3+x`
	`  f(g(x)) = 18x^6-12x^4+3x^3+2x^2-x+5`

Problem 4: Gray Code

A Gray code is a binary numbering system where two successive values differ only in one bit position. Each value in an N-bit Gray code has N bits. The most obvious case is for N = 1, when the Gray code values are just 0 and 1.

If we have a list of the N-bit Gray code values, then the N+1-bit Gray code can be obtained relatively simply. List the existing values in a column, draw a line under the last value, and "reflect" the existing values below the line. That is, duplicate the list in the opposite order. Then add a 0 bit in front of the original values and a 1 bit in front of the new "reflected" values. (Incidentally, this is why the Gray code is also called the "reflected binary code.")

For example, here is an illustration showing how we might generate the Gray code for N = 2:

Step 1: Reflection	Step 2: Prefix with 0 or 1
$\left.\begin{matrix}\texttt{0}\\\texttt{1}\end{matrix}\right\}$ Original (N = 1) `---` $\left.\begin{matrix}\texttt{1}\\\texttt{0}\end{matrix}\right\}$ Reflection	$\left.\begin{matrix}\texttt{00}\\\texttt{01}\end{matrix}\right\}$ Original values prefixed with 0 $\left.\begin{matrix}\texttt{11}\\\texttt{10}\end{matrix}\right\}$ Reflected values prefixed with 1

So the list of Gray codes for N = 2 is 00, 01, 11, and 10.

Here is how we might generate the Gray code for N = 3:

Step 1: Reflection	Step 2: Prefix with 0 or 1
$\left.\begin{matrix}\texttt{00}\\\texttt{01}\\\texttt{11}\\\texttt{10}\end{matrix}\right\}$ Original (N = 2) `----` $\left.\begin{matrix}\texttt{10}\\\texttt{11}\\\texttt{01}\\\texttt{00}\end{matrix}\right\}$ Reflection	$\left.\begin{matrix}\texttt{000}\\\texttt{001}\\\texttt{011}\\\texttt{010}\end{matrix}\right\}$ Original values prefixed with 0 $\left.\begin{matrix}\texttt{110}\\\texttt{111}\\\texttt{101}\\\texttt{100}\end{matrix}\right\}$ Reflected values prefixed with 1

Your task in this problem is to compute and display only the k-th value in the N-bit Gray code, suitably labeled. Assume the values are numbered starting with 0. So if N = 3 and k = 4 your program will display `110`.

Input

There will be multiple input cases. For each case, the input will consist of a single line containing integer values for N and k separated by one or more blanks and followed by an end of line character. N will be between 1 and 72, and k will be between 0 and 2^{N-1}. The line following the input line for the last case will contain two zeroes.

If it isn't immediately obvious from the preceding text, note that the value of k may not fit in a 32 bit or even a 64 bit integer. It should also be noted that you will not want to compute all 2^N gray code values, but focus your attention on the computation of <u>only</u> the k-th such value.

Output

For each input case display the case number (1, 2, …), the values of N and k, and the k-th Gray code value (labeled as "g"). Follow the format shown in the samples below.

Sample Input

```
3 4
4 13
10 1023
12 1000
0 0
```

Output for the Sample Input

```
Case 1:
  n = 3
  k = 4
  g = 110

Case 2:
  n = 4
  k = 13
  g = 1011

Case 3:
  n = 10
  k = 1023
  g = 1000000000

Case 4:
  n = 12
  k = 1000
  g = 001000011100
```

Problem 5: Change by Mass

Your team is working on software for automated check-out machines (such as are found in supermarkets and large home-improvement stores). The usual change-making process for these machines minimizes the number of coins dispensed when dispensing amounts under one dollar. However, a customer wants to change the logic in the machines to minimize the mass of the coins dispensed (to "lighten the load" for their customers). Minimizing the number of coins dispensed remains a secondary goal.

The name, value, and mass of each United States[1] coin with a value less than one dollar are as follows:

Name	Value	Mass
Cent ("penny")	0.01	2.500 grams
File cents ("nickel")	0.05	5.000 grams
Dime	0.10	2.268 gram
Quarter dollar	0.25	5.670 gram
Half dollar	0.50	11.340 gram

Half dollar coins may or may not be available, as they do not circulate to the extent of the other coins.

Your team is to write a program that will, given an amount in cents, determine the coins to be dispensed that will minimize the total mass of the dispensed coins, and secondarily minimize the number of coins dispensed.

Input

There will be an arbitrary number of change requests, one per line. Each change request will consist of the number of cents to be dispensed (in the range 1 to 99, inclusive) starting in the first column, followed by a single space and the number of half-dollar coins in the machine (as an integer ≥ 0). There will be a line containing -1 starting in the first column following the last change request.

Output

For each change request, your program should produce a line identifying the coins to be dispensed. This line should first show the word "Request", a single space, the request number (1, 2, …), a colon (":") and a space. Then, on the same line, show the denominations being dispensed in decreasing order in the format NxD, where N is the number of coins to be dispensed and D is the denomination of the coins. For example, 35 cents would be dispensed as "1x25 1x10". Each entry on the line is to be separated from the next entry by a single space, and no trailing whitespace is to appear. Follow the format of the sample output.

Sample Input

```
35 1
1 0
18 3
60 0
-1
```

Output for the Sample Input

```
Request 1: 1x25 1x10
Request 2: 1x1
Request 3: 1x10 1x5 3x1
Request 4: 2x25 1x10
```

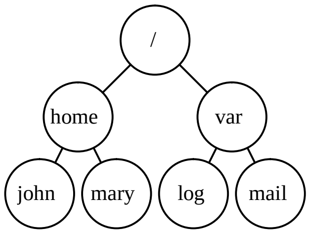
[1] No, our Canadian friends weren't forgotten. We had to pick only one set of values, though.

Problem 6: Relative Paths

Most users of modern (and many not-so-modern) operating systems are familiar with the concept of a path, which is a string giving the route in a filesystem tree from a given starting node (a directory) to a desired destination node. An absolute path starts at the root of the tree and always begins with "`/`". A relative path gives similar information, but starts from the current directory (also called the current working directory), which is just a specified node in the tree; a relative path never starts with "`/`". The names of nodes in the path are separated by "`/`"[1]. A node name of "`.`" refers to the current node, and a node name of "`..`" refers to the parent of the current node (one level higher in the tree). By convention, the node "`/`" is its own parent, so "`/..`" and "`/.`" are the same as "`/`".

Examples

Consider a few simple examples based on the simple filesystem tree shown to the left.



"`/home/john`" is an absolute path to the lower-left node in the tree, and **"`/var/mail`"** is an absolute path to the lower-right node in the tree. **"`/home/mary/..`"** is an absolute path equivalent to **"`/home`"**. If the current directory is **"`/var`"**, then the relative path **"`../home/./mary`"** is equivalent to the absolute path **"`/home/mary`"**. Finally, if the current directory is **"`/var/mail`"**, then the relative path **"`../../home/john/.`"** is equivalent to the absolute path **"`/home/john`"**.

The problem

Given a path *P* that is either absolute or relative, and the absolute path *C* to the current directory, determine the relative path *S* to the same node identified by *P* such that *S* is the shortest possible correct relative path (that is, has the fewest characters). Note that every path must have at least one character.

Input

There will be multiple cases to consider, each of which will contain two lines of input. The first line will contain the path *P*, and the second line will contain the absolute path *C*. The line following the last case will contain only the end of line. Node names will consist only of between 1 and 8 lowercase alphabetic characters, but "`/`", "`.`" and "`..`" may also appear in paths. No input line will contain whitespace (tab or space characters). Input and output paths will never be longer than 72 characters.

Output

For each input case, display five lines, as follows. On the first line, display the input case number (1, 2, …). On the next line display "P = " (indented by three spaces) and the value of *P*. On the next two lines, display *C* and *S* in a similar manner. The last line of output for each case should be a blank line. This format is illustrated in the samples.

[1] Apologies to Windows® users.

Sample Input

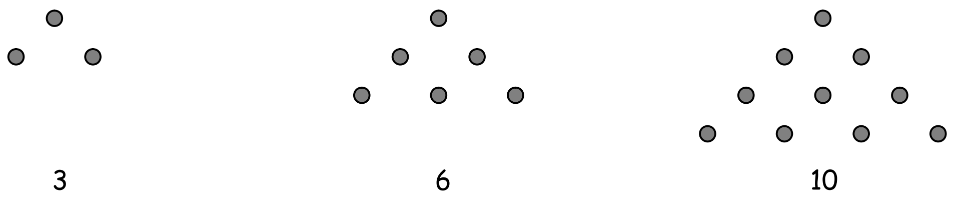
```
a/b
/c/d
/
/a/b/c
/a/b
/a/c
/
/
/x/y
/x/y
/a/b/../c
/a/d
/a/b
/c/d
/c/a/b
/c/d
/c/d/a/b
/c/d
```
(a line with only an end of line)

Output for the Sample Input

```
Case 1:
   P = a/b
   C = /c/d
   S = a/b

Case 2:
   P = /
   C = /a/b/c
   S = ../../..

Case 3:
   P = /a/b
   C = /a/c
   S = ../b

Case 4:
   P = /
   C = /
   S = .

Case 5:
   P = /x/y
   C = /x/y
   S = .

Case 6:
   P = /a/b/../c
   C = /a/d
   S = ../c

Case 7:
   P = /a/b
   C = /c/d
   S = ../../a/b

Case 8:
   P = /c/a/b
   C = /c/d
   S = ../a/b

Case 9:
   P = /c/d/a/b
   C = /c/d
   S = a/b
```

Problem 7: Inverse Triangular Numbers

A triangular number is the number of dots that can be packed in an equilateral triangle with *n* dots on a side. Here are a few example triangles, with their corresponding triangular numbers:



You can easily see that a triangular number is the additive equivalent of a factorial:

$$T(n) = n + (n - 1) + \cdots + 1$$

You are to write a program that determines if a given integer k ($0 < k < 10^9$) is a triangular number, and if it is, the number of dots on a side of the corresponding triangle.

Example

If k is 10, your program will report that it is a triangular number with 4 dots on a side, since 10 = 4 + 3 + 2 + 1. If k is 11, your program will report that it is not a triangular number.

Input

There will be an arbitrary number of cases to consider. The input for each case consists of a single line containing the value of k, with no leading zeroes or spaces and no trailing spaces. There will be a line containing -1 starting in the first column following the line for last case.

Output

For each input case, display a single output line containing the case number (1, 2, …). If the value of k for the case is a triangular number, then display the corresponding number of dots that are on a side of the triangle. Otherwise display the word "bad". Follow the format of the sample output.

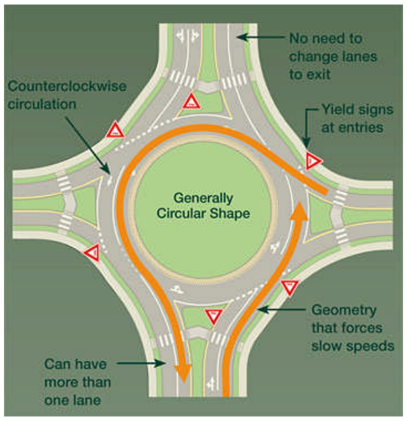
Sample Input

```
55
1
91
587
499500
-1
```

Output for the Sample Input

```
Case 1: 10
Case 2: 1
Case 3: 13
Case 4: bad
Case 5: 999
```

Problem 8: I'll Be Round About


No need to change lanes to exit
Counterclockwise circulation
Yield signs at entries
Generally Circular Shape
Geometry that forces slow speeds
Can have more than one lane

A common traffic intersection in Europe, and one that is becoming popular in the United States, is the "roundabout". When a driver enters a roundabout, the driver yields to cars that are already in the roundabout, then proceeds in and around to the correct exit. Roundabouts may have multiple lanes entering, circling, and exiting the traffic structure, which contains a center area that is usually circular. In the U.S. the cars travel in a counter-clockwise direction, while in the United Kingdom and areas where the norm is to drive on the left, the cars circle in a clockwise manner.

A particular country, Roundoslavia, uses roundabouts for all road intersections. Also, because roundabouts are so popular, the government has decided that the center circle could be fairly large. They plan on placing parks in the center. This presents a problem for the tourism bureau at Roundoslavia; the distance between two points on a map, something that they would like to print in the tourism guides, will be longer because of the increased distance at the intersections.

Roundoslavians drive on the right side of the road, so they proceed in a counter-clockwise direction. Thus a car entering the intersection from the right (East) and exiting at the bottom (South) would make three-quarters ($\frac{3}{4}$) of a circle. If the center island is 500 meters in diameter, the car will have travelled $\frac{3}{4}$ the circumference of a circle 500 meters across. In general if we consider East to be 0 degrees, North to be 90 degrees, West to be 180 degrees, and so on, we can restate this as "the car enters from zero degrees and exits at 270 degrees." Roundabouts are placed to minimize the amount of road that is needed between them, so cars may enter at an arbitrary point, say 48 degrees, and exit at 190 degrees. Also, other cars could be entering at 190 and exiting at 48.

When calculating the distances between locations on a map, the travel bureau wishes to include these extra roundabout travel distances so that the visitors to the country have an accurate estimate of how long their driving times will be.

The problem

Given data on a set of roundabouts, the roads connecting them, and the starting and ending roundabouts, determine the minimum distance between the starting and ending roundabouts and the corresponding route. The correct route will always be unique.

Input

There will be an arbitrary number of cases to consider. The input begins with a line containing a single integer that specifies the number of cases. That line is followed by the input for the first case, then the input for the second case, and so forth.

The input for each case begins with a line containing a single integer *NRB* ($1 \le NRB \le 25$) which is the number of roundabouts. This line is followed by *NRB* lines, with the *I*-th line ($1 \le I \le NRB$) giving the diameter of the *I*-th roundabout, in meters.

The next line in the input for each case contains a single integer *NRD* ($1 \le NRD \le 100$) which is the number of roads connecting the roundabouts. This line is followed by *NRD* lines (that is, one for each road), each containing five (5) integers separated by one or more spaces. The first two integers identify the roundabouts that are connected by the road. The third integer gives the length of the road, in meters, but does not include any of the distance that might be travelled inside the roundabouts. The fourth integer specifies the angle (in degrees) at which the road connects to the roundabout specified by the first integer. Similarly, the fifth integer specifies the angle at which the road connects to the roundabout specified by the second integer. The angles will always be non-negative and less than 360 degrees.

The last line in the input for each case contains two integers separated by whitespace. These integers identify the starting and ending roundabouts between which the minimum distance and corresponding route are to be determined.

Roads may curve slightly; leaving a roundabout at 90 degrees does not necessarily imply entry to another at 270 degrees. We are not concerned with this. Also, there are no one-way roads, and there is at most one road directly connecting any pair of roundabouts.

In determining the distance traveled *within* a single roundabout, the calculation should be done using real numbers, and then truncated to yield an integer.

Example

Consider the input shown in the **Sample Input** (below). There are two cases to consider. The first case has 9 roundabouts (with diameters 500, 450, …, 1500) and 12 roads. The first road connects roundabouts 6 and 1, and has a distance of 15000 meters. It connects to roundabout 6 at 0 degrees, and to roundabout 1 at 180 degrees. For this case, you are to find the appropriate route between roundabouts 6 and 9.

Note that there is no extra distance associated with entering or leaving the endpoints. In the first example case, initially leaving roundabout six adds no additional distance, and entering roundabout nine also adds no additional distance. As a simple example, suppose you were asked for the distance between roundabouts one and six. This would be just 15,000 meters. As another example, the distance from one to one is zero meters. The route in this case is 1.

Output

For each input case, print the case number (1, 2, …), the distance between the starting and ending points, and the route using the format shown. Indent the "`Distance: `" and "`Path: `" lines three spaces, and include a blank line as the last line in the output for each case. Follow the format shown.

Sample Input

```
2
9
500
450
600
1000
950
800
1200
700
1500
12
6 1 15000 0 180
6 3 40000 290 120
1 2 60000 10 190
2 3 70000 210 45
3 5 45000 270 90
5 8 78000 330 170
3 4 36000 10 190
8 4 78000 70 220
2 4 18000 280 100
2 7 68900 350 170
4 7 60000 40 220
4 9 95000 330 120
6 9
5
700
900
250
1000
750
7
1 2 10000 10 45
2 3 20000 30 0
1 5 10000 180 90
2 5 5000 45 200
2 4 40000 35 20
5 4 35000 200 300
3 4 30000 125 65
1 4
```

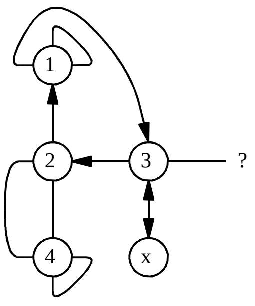
Output for the Sample Input

```
Case 1:
   Distance: 173529
   Route: 6,3,4,9

Case 2:
   Distance: 45719
   Route: 1,5,4
```

Problem 9: Jill and Phil's Palindromic Peregrinations

Jill and her friend Phil are going to ride bikes around Warsaw to do the "tourist thing" – that is, see the sights. Not being familiar with Warsaw, they plan to travel routes that are somewhat simple to remember. Each intersection encountered in their trips will have exactly four roads by which the intersection can be exited, one at each of the compass points north, east south, and west. They will leave each intersection on the road that is relatively left (L), right (R), or forward (F) of the road on which they enter the intersection (which is also conveniently at one of the four compass points). They won't pass through any intersection more than twice. And the sequence of letters (from the set F, L and R) describing their route – that is, their action at each intersection – forms a palindrome!

For example, consider the map shown to the right, with north at the top of the map. Each of the numbered circles represents an intersection; each has four roads connected to it, although some may be like the one going east from intersection 3 – a dead end. The circle labeled "x" is Jill and Phil's hotel, from which all their routes will start by going Forward. The lines (edges) between the circles represent roads.



One palindromic route from the hotel can be specified as FLRLF: forward from the hotel to 3, then left from 3 to 2, then right from 2 to 1, then left from 1 to 3, and finally forward from 3 to the hotel. This route is shown with the directed edges on the map. Another palindromic route is FLLFFLLF. This route goes from the hotel to intersections 3, 2, 4, 4 again, 2, 1, 3, and finally back to the hotel.

Given a map, find all the palindromic routes that Jill and Phil might take.

Input

The input contains several maps followed by a line containing a single zero.

The first line of input for each map gives the number of intersections, N, which will be no larger than 15. The line then has the intersection number (1..N) immediately followed by the capital letter compass point ('N', 'E', 'S', or 'W') where the road Forward from the hotel will take Jill and Phil.

Then there are N lines, one for each intersection (numbered 1 to N). The four space-separated fields on the K–th of these lines indicate the intersections to which the four roads from intersection K are connected; the fields correspond to the roads leaving the north, east, south, and west of intersection K, respectively. Each field contains one of the following:

- an intersection number L immediately followed by one of the capital letter compass points 'N', 'E', 'S', or 'W'; this indicates that the corresponding road leaving intersection K connects to intersection L at the specified compass point.
- 0; this indicates the corresponding road is a dead-end on which travel is not possible.
- –1; this marks the road that leads to the hotel. There will only be one such road in the map.

Output

For each input map, display the map number and each of the possible palindromic routes in lexicographic order. Display the message "`NO ROUTES`" if there are no palindromic routes possible. Follow the format of the sample output.

Sample Input

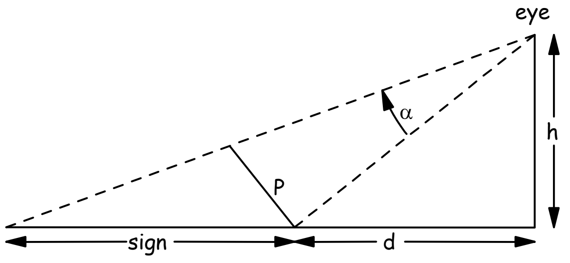
```
4 3S
1E    1N    2N    3N
1S    3W    4N    4W
1W    0    -1     2E
2S    4S    4E    2W
2 1S
1E    1N    -1    2E
0     1W     0    0
2 1S
2S 2E -1 0
0 1E 1N 0
0
```

Output for the Sample Input

```
Map 1:
   FFLFF
   FLFFFFLF
   FLFLFLF
   FLFRRFLF
   FLLFFLLF
   FLLLLLLF
   FLLRLLF
   FLRLF
Map 2:
   FRF
Map 3:
   NO ROUTES
```

Problem 10: Pavement Signage

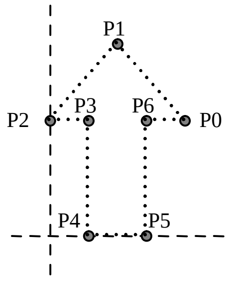
Words and symbols painted on roads must be elongated because drivers view roads at an angle. For example, using the figure below, a safety engineer uses an assumed eye level above the road of *h*, and designates how big the symbol should appear at a distance *d* from the driver's position. The apparent size is expressed as the angular size of the driver's field of view, α. The road (the horizontal line in the figure) is assumed to be perfectly flat.



The sign's appearance to the driver is intended to be as if it was positioned in the plane labeled P in the figure, exactly filling the field of view. The plane is perpendicular to the dashed line from the driver's eye to the point at distance *d* along the road from the driver's position. Of course, a point in a sign that is "virtually" in this plane is actually on the pavement at the point where a line from the driver's eye through the point in the plane would intersect the road. Thus the sign on the pavement will fill the area labeled "sign" in the figure.

The Problem

Write a program that will provide information to be used in preparing a stencil for painting signs on roadways, given h, d, α, and a description of the "un-stretched" symbol. The symbol description is given as a sequence of points in a Cartesian coordinate system so that the "un-stretched" symbol could be drawn by "connecting the dots" – draw a line from the first point to the second point, then to the third point, and so forth, and finally draw a line back to the first point. For example, an arrow indicating the driver should travel straight in the lane with the symbol might look like the figure shown to the left, which is specified by giving the coordinates of the seven points labeled P0 through P6. The dashed lines represent the X and Y axes, and the dotted lines represent the outline of the figure. The technician who measures the un-stretched symbol aligns it so that the leftmost point(s) touch the Y axis, and the bottommost point(s) touch the X axis.



The horizontal perspective of the symbol is not adjusted; only the y values for the points in the symbol are modified.

Input

There will be an arbitrary number of cases to consider. The input for each case begins with a single line containing three real numbers *h*, *d*, and α, and an integer *n* separated from each other by whitespace. *h* and *d* are given in meters, and α is given in degrees. This line is followed by *n* lines, each containing two real numbers *x* and *y* representing the Cartesian coordinates of a point in the symbol to be drawn. n will never be larger than 20.

Input for the last case is followed by a line containing four zeros.

Output

For each input case, first display a line containing the case number (1, 2, …). After this, display one line giving the x and y coordinates for each point in the stretched symbol. Display two digits after the decimal point in each x and y value. Indent each of these lines by three (3) spaces, put one space between the x and y values, and no trailing whitespace after the y value. Display a blank line after the output for each case. Follow the format of the sample output.

Sample Input

```
1.5 8 6.0 7
0.7   0.6
0.35 1
0     0.6
0.2   .6
0.2  0
0.5  0
0.5 .6
0 0 0 0
```

Output for the Sample Input

```
Case 1:
   0.70 4.20
   0.35 10.56
   0.00 4.20
   0.20 4.20
   0.20 0.00
   0.50 0.00
   0.50 4.20

```