Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

Consider the following algorithm:

1. input n
2. print n
3. if $n = 1$ then STOP
4. if n is odd then $n \longleftarrow 3n + 1$
5. else $n \longleftarrow n/2$
6. GOTO 2

Given the input 22, the following sequence of numbers will be printed

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers n such that $0 < n < 1,000,000$ (and, in fact, for many more numbers than this.)

Given an input n, it is possible to determine the number of numbers printed before and including the 1 is printed. For a given n this is called the *cycle-length* of n. In the example above, the cycle length of 22 is 16.

For any two numbers i and j you are to determine the maximum cycle length over all numbers between and including both i and j.

Input

The input will consist of a series of pairs of integers i and j, one pair of integers per line. All integers will be less than 10,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including i and j.

You can assume that no operation overflows a 32-bit integer.

Output

For each pair of input integers i and j you should output i, j, and the maximum cycle length for integers between and including i and j. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers i and j must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

Sample Input

```
1 10
100 200
201 210
900 1000
```

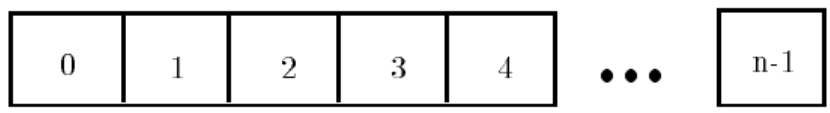
Sample Output

```
1 10 20
100 200 125
201 210 89
900 1000 174
```

Many areas of Computer Science use simple, abstract domains for both analytical and empirical studies. For example, an early AI study of planning and robotics (STRIPS) used a block world in which a robot arm performed tasks involving the manipulation of blocks.

In this problem you will model a simple block world under certain rules and constraints. Rather than determine how to achieve a specified state, you will "program" a robotic arm to respond to a limited set of commands.

The problem is to parse a series of commands that instruct a robot arm in how to manipulate blocks that lie on a flat table. Initially there are n blocks on the table (numbered from 0 to $n-1$) with block b_i adjacent to block b_{i+1} for all $0 \leq i < n-1$ as shown in the diagram below:



Initial Blocks World

The valid commands for the robot arm that manipulates blocks are:

- `move` a `onto` b

  where a and b are block numbers, puts block a onto block b after returning any blocks that are stacked on top of blocks a and b to their initial positions.

- `move` a `over` b

  where a and b are block numbers, puts block a onto the top of the stack containing block b, after returning any blocks that are stacked on top of block a to their initial positions.

- `pile` a `onto` b

  where a and b are block numbers, moves the pile of blocks consisting of block a, and any blocks that are stacked above block a, onto block b. All blocks on top of block b are moved to their initial positions prior to the pile taking place. The blocks stacked above block a retain their order when moved.

- `pile` a `over` b

  where a and b are block numbers, puts the pile of blocks consisting of block a, and any blocks that are stacked above block a, onto the top of the stack containing block b. The blocks stacked above block a retain their original order when moved.

- `quit`

  terminates manipulations in the block world.

Any command in which $a = b$ or in which a and b are in the same stack of blocks is an illegal command. All illegal commands should be ignored and should have no affect on the configuration of blocks.

Input

The input begins with an integer n on a line by itself representing the number of blocks in the block world. You may assume that $0 < n < 25$.

The number of blocks is followed by a sequence of block commands, one command per line. Your program should process all commands until the `quit` command is encountered.

You may assume that all commands will be of the form specified above. There will be no syntactically incorrect commands.

Output

The output should consist of the final state of the blocks world. Each original block position numbered i ($0 \leq i < n$ where n is the number of blocks) should appear followed immediately by a colon. If there is at least a block on it, the colon must be followed by one space, followed by a list of blocks that appear stacked in that position with each block number separated from other block numbers by a space. Don't put any trailing spaces on a line.

There should be one line of output for each block position (i.e., n lines of output where n is the integer on the first line of input).

Sample Input

```
10
move 9 onto 1
move 8 over 1
move 7 over 1
move 6 over 1
pile 8 over 6
pile 8 over 5
move 2 over 1
move 4 over 9
quit
```

Sample Output

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

Bin packing, or the placement of objects of certain weights into different bins subject to certain constraints, is an historically interesting problem. Some bin packing problems are NP-complete but are amenable to dynamic programming solutions or to approximately optimal heuristic solutions.

In this problem you will be solving a bin packing problem that deals with recycling glass.

Recycling glass requires that the glass be separated by color into one of three categories: brown glass, green glass, and clear glass. In this problem you will be given three recycling bins, each containing a specified number of brown, green and clear bottles. In order to be recycled, the bottles will need to be moved so that each bin contains bottles of only one color.

The problem is to minimize the number of bottles that are moved. You may assume that the only problem is to minimize the number of movements between boxes.

For the purposes of this problem, each bin has infinite capacity and the only constraint is moving the bottles so that each bin contains bottles of a single color. The total number of bottles will never exceed 2^{31}.

Input

The input consists of a series of lines with each line containing 9 integers. The first three integers on a line represent the number of brown, green, and clear bottles (respectively) in bin number 1, the second three represent the number of brown, green and clear bottles (respectively) in bin number 2, and the last three integers represent the number of brown, green, and clear bottles (respectively) in bin number 3. For example, the line

10 15 20 30 12 8 15 8 31

indicates that there are 20 clear bottles in bin 1, 12 green bottles in bin 2, and 15 brown bottles in bin 3.

Integers on a line will be separated by one or more spaces. Your program should process all lines in the input file.

Output

For each line of input there will be one line of output indicating what color bottles go in what bin to minimize the number of bottle movements. You should also print the minimum number of bottle movements.

The output should consist of a string of the three upper case characters 'G', 'B', 'C' (representing the colors green, brown, and clear) representing the color associated with each bin.

The first character of the string represents the color associated with the first bin, the second character of the string represents the color associated with the second bin, and the third character represents the color associated with the third bin.

The integer indicating the minimum number of bottle movements should follow the string.

If more than one order of brown, green, and clear bins yields the minimum number of movements then the alphabetically first string representing a minimal configuration should be printed.

Sample Input

```
1 2 3 4 5 6 7 8 9
5 10 5 20 10 5 10 20 10
```

Sample Output

```
BCG 30
CBG 50
```

Some concepts in Mathematics and Computer Science are simple in one or two dimensions but become more complex when extended to arbitrary dimensions. Consider solving differential equations in several dimensions and analyzing the topology of an n-dimensional hypercube. The former is much more complicated than its one dimensional relative while the latter bears a remarkable resemblance to its "lower-class" cousin.

Consider an n-dimensional "box" given by its dimensions. In two dimensions the box (2,3) might represent a box with length 2 units and width 3 units. In three dimensions the box (4,8,9) can represent a box $4 \times 8 \times 9$ (length, width, and height). In 6 dimensions it is, perhaps, unclear what the box (4,5,6,7,8,9) represents; but we can analyze properties of the box such as the sum of its dimensions.

In this problem you will analyze a property of a group of n-dimensional boxes. You are to determine the longest *nesting string* of boxes, that is a sequence of boxes $b_1, b_2, \ldots, b_k$ such that each box b_i nests in box b_{i+1} $(1 \leq i < k)$.

A box D = $(d_1, d_2, \ldots, d_n)$ nests in a box E = $(e_1, e_2, \ldots, e_n)$ if there is some rearrangement of the d_i such that when rearranged each dimension is less than the corresponding dimension in box E. This loosely corresponds to turning box D to see if it will fit in box E. However, since any rearrangement suffices, box D can be contorted, not just turned (see examples below).

For example, the box D = (2,6) nests in the box E = (7,3) since D can be rearranged as (6,2) so that each dimension is less than the corresponding dimension in E. The box D = (9,5,7,3) does NOT nest in the box E = (2,10,6,8) since no rearrangement of D results in a box that satisfies the nesting property, but F = (9,5,7,1) does nest in box E since F can be rearranged as (1,9,5,7) which nests in E.

Formally, we define nesting as follows: box D = $(d_1, d_2, \ldots, d_n)$ *nests* in box E = $(e_1, e_2, \ldots, e_n)$ if there is a permutation π of $1 \ldots n$ such that $(d_{\pi(1)}, d_{\pi(2)}, \ldots, d_{\pi(n)})$ "fits" in $(e_1, e_2, \ldots, e_n)$ i.e., if $d_{\pi(i)} < e_i$ for all $1 \leq i \leq n$.

Input

The input consists of a series of box sequences. Each box sequence begins with a line consisting of the the number of boxes k in the sequence followed by the dimensionality of the boxes, n (on the same line.)

This line is followed by k lines, one line per box with the n measurements of each box on one line separated by one or more spaces. The i-th line in the sequence $(1 \leq i \leq k)$ gives the measurements for the i-th box.

There may be several box sequences in the input file. Your program should process all of them and determine, for each sequence, which of the k boxes determine the longest nesting string and the length of that nesting string (the number of boxes in the string).

In this problem the maximum dimensionality is 10 and the minimum dimensionality is 1. The maximum number of boxes in a sequence is 30.

Output

For each box sequence in the input file, output the length of the longest nesting string on one line followed on the next line by a list of the boxes that comprise this string in order. The "smallest" or "innermost" box of the nesting string should be listed first, the next box (if there is one) should be listed second, etc.

The boxes should be numbered according to the order in which they appeared in the input file (first box is box 1, etc.).

If there is more than one longest nesting string then any one of them can be output.

Sample Input

```
5 2
3 7
8 10
5 2
9 11
21 18
8 6
5 2 20 1 30 10
23 15 7 9 11 3
40 50 34 24 14 4
9 10 11 12 13 14
31 4 18 8 27 17
44 32 13 19 41 19
1 2 3 4 5 6
80 37 47 18 21 9
```

Sample Output

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

The use of computers in the finance industry has been marked with controversy lately as programmed trading — designed to take advantage of extremely small fluctuations in prices — has been outlawed at many Wall Street firms. The ethics of computer programming is a fledgling field with many thorny issues.

Arbitrage is the trading of one currency for another with the hopes of taking advantage of small differences in conversion rates among several currencies in order to achieve a profit. For example, if \$1.00 in U.S. currency buys 0.7 British pounds currency, £1 in British currency buys 9.5 French francs, and 1 French franc buys 0.16 in U.S. dollars, then an arbitrage trader can start with \$1.00 and earn $1 \times 0.7 \times 9.5 \times 0.16 = 1.064$ dollars thus earning a profit of 6.4 percent.

You will write a program that determines whether a sequence of currency exchanges can yield a profit as described above.

To result in successful arbitrage, a sequence of exchanges must begin and end with the same currency, but any starting currency may be considered.

Input

The input file consists of one or more conversion tables. You must solve the arbitrage problem for each of the tables in the input file.

Each table is preceded by an integer n on a line by itself giving the dimensions of the table. The maximum dimension is 20; the minimum dimension is 2.

The table then follows in row major order but with the diagonal elements of the table missing (these are assumed to have value 1.0). Thus the first row of the table represents the conversion rates between country 1 and $n-1$ other countries, i.e., the amount of currency of country i ($2 \leq i \leq n$) that can be purchased with one unit of the currency of country 1.

Thus each table consists of $n+1$ lines in the input file: 1 line containing n and n lines representing the conversion table.

Output

For each table in the input file you must determine whether a sequence of exchanges exists that results in a profit of more than 1 percent (0.01). If a sequence exists you must print the sequence of exchanges that results in a profit. If there is more than one sequence that results in a profit of more than 1 percent you must print a sequence of minimal length, i.e., one of the sequences that uses the fewest exchanges of currencies to yield a profit.

Because the IRS (United States Internal Revenue Service) notices lengthy transaction sequences, all profiting sequences must consist of n or fewer transactions where n is the dimension of the table giving conversion rates. The sequence 1 2 1 represents two conversions.

If a profiting sequence exists you must print the sequence of exchanges that results in a profit. The sequence is printed as a sequence of integers with the integer i representing the i-th line of the conversion table (country i). The first integer in the sequence is the country from which the profiting sequence starts. This integer also ends the sequence.

If no profiting sequence of n or fewer transactions exists, then the line

```
no arbitrage sequence exists
```

should be printed.

Sample Input

```
3
1.2 .89
.88 5.1
1.1 0.15
4
3.1     0.0023     0.35
0.21    0.00353    8.13
200     180.559    10.339
2.11    0.089      0.06111
2
2.0
0.45
```

Sample Output

```
1 2 1
1 2 4 1
no arbitrage sequence exists
```

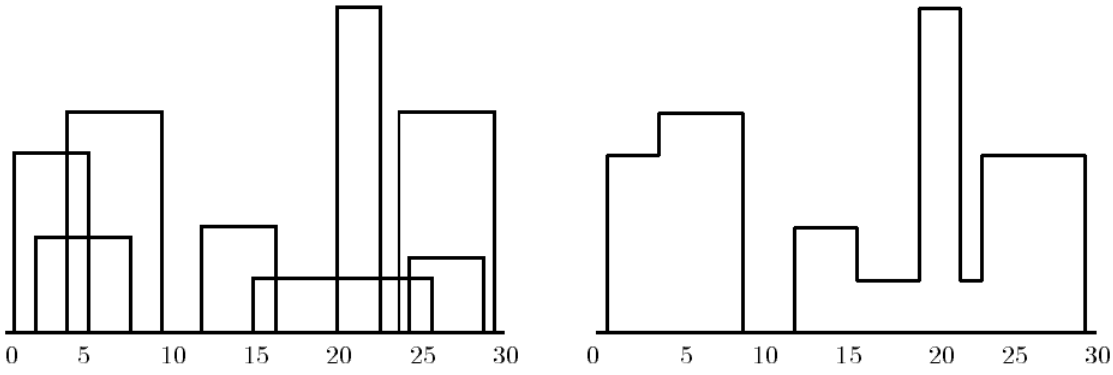
With the advent of high speed graphics workstations, CAD (computer-aided design) and other areas (CAM, VLSI design) have made increasingly effective use of computers. One of the problems with drawing images is the elimination of hidden lines — lines obscured by other parts of a drawing.

You are to design a program to assist an architect in drawing the skyline of a city given the locations of the buildings in the city. To make the problem tractable, all buildings are rectangular in shape and they share a common bottom (the city they are built in is very flat). The city is also viewed as two-dimensional. A building is specified by an ordered triple (L_i, H_i, R_i) where L_i and R_i are the left and right coordinates, respectively, of building i $(0 < L_i < R_i)$ and H_i is the height of the building. In the diagram below buildings are shown on the left with triples

$$(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)$$

the skyline, shown on the right, is represented by the sequence:

$$(1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19, 18, 22, 3, 23, 13, 29, 0)$$



Input

The input is a sequence of building triples. All coordinates of buildings are integers less than 10,000 and there will be at least one and at most 5,000 buildings in the input file. Each building triple is on a line by itself in the input file. All integers in a triple are separated by one or more spaces. The triples will be sorted by L_i, the left x-coordinate of the building, so the building with the smallest left x-coordinate is first in the input file.

Output

The output should consist of the vector that describes the skyline as shown in the example above. In the skyline vector $(v_1, v_2, v_3, \ldots, v_{n-2}, v_{n-1}, v_n)$, the v_i such that i is an even number represent a horizontal line (height). The v_i such that i is an odd number represent a vertical line (x-coordinate). The skyline vector should represent the "path" taken, for example, by a bug starting at the minimum x-coordinate and traveling horizontally and vertically over all the lines that define the skyline. Thus the last entry in all skyline vectors will be a '0'.

Sample Input

```
1 11 5
2 6 7
3 13 9
12 7 16
14 3 25
19 18 22
23 13 29
24 4 28
```

Sample Output

```
1 11 3 13 9 0 12 7 16 3 19 18 22 3 23 13 29 0
```

Computer generated and assisted proofs and verification occupy a small niche in the realm of Computer Science. The first proof of the four-color problem was completed with the assistance of a computer program and current efforts in verification have succeeded in verifying the translation of high-level code down to the chip level.

This problem deals with computing quantities relating to part of Fermat's Last Theorem: that there are no integer solutions of $a^n + b^n = c^n$ for $n > 2$.

Given a positive integer N, you are to write a program that computes two quantities regarding the solution of

$$x^2 + y^2 = z^2$$

where x, y, and z are constrained to be positive integers less than or equal to N. You are to compute the number of triples (x, y, z) such that $x < y < z$, and they are relatively prime, i.e., have no common divisor larger than 1. You are also to compute the number of values $0 < p \leq N$ such that p is not part of any triple (not just relatively prime triples).

Input

The input consists of a sequence of positive integers, one per line. Each integer in the input file will be less than or equal to 1,000,000. Input is terminated by end-of-file.

Output

For each integer N in the input file print two integers separated by a space. The first integer is the number of relatively prime triples (such that each component of the triple is $\leq N$). The second number is the number of positive integers $\leq N$ that are not part of any triple whose components are all $\leq N$. There should be one output line for each input line.

Sample Input

```
10
25
100
```

Sample Output

```
1 4
4 9
16 27
```

(An homage to Theodore Seuss Geisel)

> The Cat in the Hat is a nasty creature,
> But the striped hat he is wearing has a rather nifty feature.
>
> With one flick of his wrist he pops his top off.
>
> Do you know what's inside that Cat's hat?
> A bunch of small cats, each with its own striped hat.
>
> Each little cat does the same as line three,
> All except the littlest ones, who just say "Why me?"
>
> Because the littlest cats have to clean all the grime,
> And they're tired of doing it time after time!

A clever cat walks into a messy room which he needs to clean. Instead of doing the work alone, it decides to have its helper cats do the work. It keeps its (smaller) helper cats inside its hat. Each helper cat also has helper cats in its own hat, and so on. Eventually, the cats reach a smallest size. These smallest cats have no additional cats in their hats. These unfortunate smallest cats have to do the cleaning.

The number of cats inside each (non-smallest) cat's hat is a constant, N. The height of these cats-in-a-hat is $\frac{1}{N+1}$ times the height of the cat whose hat they are in.

> The smallest cats are of height one;
> these are the cats that get the work done.

All heights are positive integers.

Given the height of the initial cat and the number of worker cats (of height one), find the number of cats that are not doing any work (cats of height greater than one) and also determine the sum of all the cats' heights (the height of a stack of all cats standing one on top of another).

Input

The input consists of a sequence of cat-in-hat specifications. Each specification is a single line consisting of two positive integers, separated by white space. The first integer is the height of the initial cat, and the second integer is the number of worker cats.

A pair of '0's on a line indicates the end of input.

Output

For each input line (cat-in-hat specification), print the number of cats that are not working, followed by a space, followed by the height of the stack of cats. There should be one output line for each input line other than the '0 0' that terminates input.

Sample Input

```
216 125
5764801 1679616
0 0
```

Sample Output

```
31 671
335923 30275911
```

A problem that is simple to solve in one dimension is often much more difficult to solve in more than one dimension. Consider satisfying a boolean expression in conjunctive normal form in which each conjunct consists of exactly 3 disjuncts. This problem (3-SAT) is NP-complete. The problem 2-SAT is solved quite efficiently, however. In contrast, some problems belong to the same complexity class regardless of the dimensionality of the problem.

Given a 2-dimensional array of positive and negative integers, find the sub-rectangle with the largest sum. The sum of a rectangle is the sum of all the elements in that rectangle. In this problem the sub-rectangle with the largest sum is referred to as the *maximal sub-rectangle*.

A sub-rectangle is any contiguous sub-array of size 1×1 or greater located within the whole array. As an example, the maximal sub-rectangle of the array:

$$\begin{array}{rrrr} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{array}$$

is in the lower-left-hand corner:

$$\begin{array}{rr} 9 & 2 \\ -4 & 1 \\ -1 & 8 \end{array}$$

and has the sum of 15.

Input

The input consists of an $N \times N$ array of integers.

The input begins with a single positive integer N on a line by itself indicating the size of the square two dimensional array. This is followed by N^2 integers separated by white-space (newlines and spaces). These N^2 integers make up the array in row-major order (i.e., all numbers on the first row, left-to-right, then all numbers on the second row, left-to-right, etc.). N may be as large as 100. The numbers in the array will be in the range $[-127, 127]$.

Output

The output is the sum of the maximal sub-rectangle.

Sample Input

```
4
0 -2 -7  0 9  2 -6  2
-4  1 -4  1 -1
8  0 -2
```

Sample Output

```
15
```

Some problems are difficult to solve but have a simplification that is easy to solve. Rather than deal with the difficulties of constructing a model of the Earth (a somewhat oblate spheroid), consider a pre-Columbian flat world that is a 500 kilometer × 500 kilometer square.

In the model used in this problem, the flat world consists of several warring kingdoms. Though warlike, the people of the world are strict isolationists; each kingdom is surrounded by a high (but thin) wall designed to both protect the kingdom and to isolate it. To avoid fights for power, each kingdom has its own electric power plant.

When the urge to fight becomes too great, the people of a kingdom often launch missiles at other kingdoms. Each SCUD missile (Sanitary Cleansing Universal Destroyer) that lands within the walls of a kingdom destroys that kingdom's power plant (without loss of life).

Given coordinate locations of several kingdoms (by specifying the locations of houses and the location of the power plant in a kingdom) and missile landings you are to write a program that determines the total area of all kingdoms that are without power after an exchange of missile fire.

In the simple world of this problem kingdoms do not overlap. Furthermore, the walls surrounding each kingdom are considered to be of zero thickness. The wall surrounding a kingdom is the minimal-perimeter wall that completely surrounds all the houses and the power station that comprise a kingdom; the area of a kingdom is the area enclosed by the minimal-perimeter thin wall.

There is exactly one power station per kingdom.

There may be empty space between kingdoms.

Input

The input is a sequence of kingdom specifications followed by a sequence of missile landing locations.

A kingdom is specified by a number N ($3 \leq N \leq 100$) on a single line which indicates the number of sites in this kingdom. The next line contains the x and y coordinates of the power station, followed by $N-1$ lines of x, y pairs indicating the locations of homes served by this power station. A value of -1 for N indicates that there are no more kingdoms. There will be at least one kingdom in the data set.

Following the last kingdom specification will be the coordinates of one or more missile attacks, indicating the location of a missile landing. Each missile location is on a line by itself. You are to process missile attacks until you reach the end of the file.

Locations are specified in kilometers using coordinates on a 500 km by 500 km grid. All coordinates will be integers between 0 and 500 inclusive. Coordinates are specified as a pair of integers separated by white-space on a single line. The input file will consist of up to 20 kingdoms, followed by any number of missile attacks.

Output

The output consists of a single number representing the total area of all kingdoms without electricity after all missile attacks have been processed. The number should be printed with (and correct to) two decimal places.

A Hint: You may or may not find the following formula useful.

Given a polygon described by the vertices $v_0, v_1, \ldots, v_n$ such that $v_0 = v_n$, the signed area of the polygon is given by

$$a = \frac{1}{2} \sum_{i=1}^{n} (x_{i-1} y_i) - (x_i y_{i-1})$$

where the x, y coordinates of $v_i = (x_i, y_i)$; the edges of the polygon are from v_i to v_{i+1} for $i = 0 \ldots n-1$.

If the points describing the polygon are given in a counterclockwise direction, the value of a will be positive, and if the points of the polygon are listed in a clockwise direction, the value of a will be negative.

Sample Input

```
12
3 3
4 6
4 11
4 8
10 6
5 7
6 6
6 3
7 9
10 4
10 9
1 7
5
20 20
20 40
40 20
40 40
30 30
3
10 10
21 10
21 13
-1
5 5
20 12
```

Sample Output

```
70.50
```

Sorting holds an important place in computer science. Analyzing and implementing various sorting algorithms forms an important part of the education of most computer scientists, and sorting accounts for a significant percentage of the world's computational resources. Sorting algorithms range from the bewilderingly popular Bubble sort, to Quicksort, to parallel sorting algorithms and sorting networks. In this problem you will be writing a program that creates a sorting program (a meta-sorter).

The problem is to create several programs whose output is a standard Pascal programs that sorts n numbers where n is the only input to the program you will write. The Pascal program generated by your program must have the following properties:

- They must begin with `program sort(input,output);`
- They must declare storage for exactly n `integer` variables. The names of the variables must come from the first n letters of the alphabet (a,b,c,d,e,f).
- A single `readln` statement must read in values for all the integer variables.
- Other than `writeln` statements, the only statements in the program are `if then else` statements. The boolean conditional for each `if` statement must consist of one strict inequality (either `<` or `>`) of two integer variables. Exactly $n!$ `writeln` statements must appear in the program.
- Exactly three semi-colons must appear in the programs
  1. after the program header: `program sort(input,output);`
  2. after the variable declaration: `... : integer;`
  3. after the `readln` statement: `readln(...);`
- No redundant comparisons of integer variables should be made. For example, during program execution, once it is determined that $a < b$, variables a and b should not be compared again.
- Every `writeln` statement must appear on a line by itself.
- The programs must compile. Executing the program with input consisting of any arrangement of any n distinct integer values should result in the input values being printed in sorted order.

For those unfamiliar with Pascal syntax, the example at the end of this problem completely defines the small subset of Pascal needed.

Input

The input consist on a number in the first line indicating the number M of programs to make, followed by a blank line. Then there are M test cases, each one consisting on a single integer n on a line by itself with $1 \leq n \leq 8$.

There will be a blank line between test cases.

Output

The output is M compilable standard Pascal programs meeting the criteria specified above.

Print a blank line between two consecutive programs.

Sample Input

```
1

3
```

Sample Output

```
program sort(input,output);
var
a,b,c : integer;
begin
  readln(a,b,c);
  if a < b then
    if b < c then
      writeln(a,b,c)
    else if a < c then
      writeln(a,c,b)
    else
      writeln(c,a,b)
  else
    if a < c then
      writeln(b,a,c)
    else if b < c then
      writeln(b,c,a)
    else
      writeln(c,b,a)
end.
```

Many problems in Computer Science involve maximizing some measure according to constraints.

Consider a history exam in which students are asked to put several historical events into chronological order. Students who order all the events correctly will receive full credit, but how should partial credit be awarded to students who incorrectly rank one or more of the historical events?

Some possibilities for partial credit include:

1. 1 point for each event whose rank matches its correct rank
2. 1 point for each event in the longest (not necessarily contiguous) sequence of events which are in the correct order relative to each other.

For example, if four events are correctly ordered 1 2 3 4 then the order 1 3 2 4 would receive a score of 2 using the first method (events 1 and 4 are correctly ranked) and a score of 3 using the second method (event sequences 1 2 4 and 1 3 4 are both in the correct order relative to each other).

In this problem you are asked to write a program to score such questions using the second method.

Given the correct chronological order of n events $1, 2, \ldots, n$ as $c_1, c_2, \ldots c_n$ where $1 \leq c_i \leq n$ denotes the ranking of event i in the correct chronological order and a sequence of student responses $r_1, r_2, \ldots, r_n$ where $1 \leq r_i \leq n$ denotes the chronological rank given by the student to event i; determine the length of the longest (not necessarily contiguous) sequence of events in the student responses that are in the correct chronological order relative to each other.

Input

The input file contains one or more test cases, each of them as described below.

The first line of the input will consist of one integer n indicating the number of events with $2 \leq n \leq 20$. The second line will contain n integers, indicating the correct chronological order of n events. The remaining lines will each consist of n integers with each line representing a student's chronological ordering of the n events. All lines will contain n numbers in the range $[1 \ldots n]$, with each number appearing exactly once per line, and with each number separated from other numbers on the same line by one or more spaces.

Output

For each test case, the output must follow the description below

For each student ranking of events your program should print the score for that ranking. There should be one line of output for each student ranking.

Warning: Read carefully the description and consider the difference between 'ordering' and 'ranking'.

Sample Input

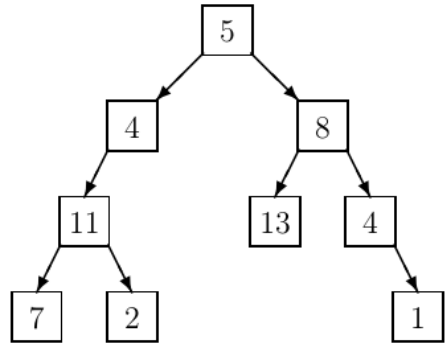
```
4
4 2 3 1
1 3 2 4
3 2 1 4
2 3 4 1
10
3 1 2 4 9 5 10 6 8 7
1 2 3 4 5 6 7 8 9 10
4 7 2 3 10 6 9 1 5 8
3 1 2 4 9 5 10 6 8 7
2 10 1 3 8 4 9 5 7 6
```

Sample Output

```
1
2
3
6
5
10
9
```

LISP was one of the earliest high-level programming languages and, with FORTRAN, is one of the oldest languages currently being used. Lists, which are the fundamental data structures in LISP, can easily be adapted to represent other important data structures such as trees.

This problem deals with determining whether binary trees represented as LISP S-expressions possess a certain property.



Given a binary tree of integers, you are to write a program that determines whether there exists a root-to-leaf path whose nodes sum to a specified integer.

For example, in the tree shown on the right there are exactly four root-to-leaf paths. The sums of the paths are 27, 22, 26, and 18. Binary trees are represented in the input file as LISP S-expressions having the following form.

empty tree ::= ()
tree ::= *empty tree* | (*integer tree tree*)

The tree diagrammed above is represented by the expression

```
(5 (4 (11 (7 () ()) (2 () ()) ) ()) (8 (13 () ()) (4 () (1 () ()) ) ) )
```

Note that with this formulation all leaves of a tree are of the form

(*integer* () ())

Since an empty tree has no root-to-leaf paths, any query as to whether a path exists whose sum is a specified integer in an empty tree must be answered negatively.

Input

The input consists of a sequence of test cases in the form of integer/tree pairs. Each test case consists of an integer followed by one or more spaces followed by a binary tree formatted as an S-expression as described above. All binary tree S-expressions will be valid, but expressions may be spread over several lines and may contain spaces. There will be one or more test cases in an input file, and input is terminated by end-of-file.

Output

There should be one line of output for each test case (integer/tree pair) in the input file. For each pair I, T (I represents the integer, T represents the tree) the output is the string '`yes`' if there is a root-to-leaf path in T whose sum is I and '`no`' if there is no path in T whose sum is I.

Sample Input

```
22 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))
20 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))
10 (3
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )
5 ()
```

Sample Output

```
yes
no
yes
no
```

Current work in cryptography involves (among other things) large prime numbers and computing powers of numbers modulo functions of these primes. Work in this area has resulted in the practical use of results from number theory and other branches of mathematics once considered to be of only theoretical interest.

This problem involves the efficient computation of integer roots of numbers.

Given an integer $n \geq 1$ and an integer $p \geq 1$ you are to write a program that determines $\sqrt[n]{p}$, the positive n-th root of p. In this problem, given such integers n and p, p will always be of the form k^n for an integer k (this integer is what your program must find).

Input

The input consists of a sequence of integer pairs n and p with each integer on a line by itself. For all such pairs $1 \leq n \leq 200$, $1 \leq p < 10^{101}$ and there exists an integer k, $1 \leq k \leq 10^9$ such that $k^n = p$.

Output

For each integer pair n and p the value $\sqrt[n]{p}$ should be printed, i.e., the number k such that $k^n = p$.

Sample Input

```
2
16
3
27
7
435718618402138220454 4
```

Sample Output

```
4
3
1234
```

Simulation is an important application area in computer science involving the development of computer models to provide insight into real-world events. There are many kinds of simulation including (and certainly not limited to) discrete event simulation and clock-driven simulation. Simulation often involves approximating observed behavior in order to develop a practical approach.

This problem involves the simulation of a simplistic *pinball* machine. In a pinball machine, a steel ball rolls around a surface, hitting various objects (*bumpers*) and accruing points until the ball "disappears" from the surface.

You are to write a program that simulates an idealized pinball machine. This machine has a flat surface that has some obstacles (bumpers and walls). The surface is modeled as an $m \times n$ grid with the origin in the lower-left corner. Each bumper occupies a grid point. The grid positions on the edge of the surface are walls. Balls are shot (appear) one at a time on the grid, with an initial position, direction, and lifetime.

In this simulation, all positions are integral, and the ball's direction is one of: up, down, left, or right. The ball bounces around the grid, hitting bumpers (which accumulates points) and walls (which does not add any points). The number of points accumulated by hitting a given bumper is the *value* of that bumper. The speed of all balls is one grid space per timestep. A ball "hits" an obstacle during a timestep when it would otherwise move on top of the bumper or wall grid point. A hit causes the ball to "rebound" by turning right (clockwise) 90 degrees, without ever moving on top of the obstacle and without changing position (only the direction changes as a result of a rebound). Note that by this definition sliding along a wall does not constitute "hitting" that wall.

A ball's lifetime indicates how many time units the ball will live before disappearing from the surface. The ball uses one unit of lifetime for each grid step it moves. It also uses some units of lifetime for each bumper or wall that it hits. The lifetime used by a hit is the *cost* of that bumper or wall. As long as the ball has a positive lifetime when it hits a bumper, it obtains the full score for that bumper. Note that a ball with lifetime one will "die" during its next move and thus cannot obtain points for hitting a bumper during this last move. Once the lifetime is non-positive (less than or equal to zero), the ball disappears and the game continues with the next ball.

Input

Your program should simulate one game of pinball. There are several input lines that describe the game. The first line gives integers m and n, separated by a space. This describes a cartesian grid where $1 \leq x \leq m$ and $1 \leq y \leq n$ on which the game is "played". It will be the case that $2 < m < 51$ and $2 < n < 51$. The next line gives the integer cost for hitting a wall. The next line gives the number of bumpers, an integer $p \geq 0$.

The next p lines give the x position, y position, value, and cost, of each bumper, as four integers per line separated by space(s). The x and y positions of all bumpers will be in the range of the grid. The value and cost may be any integer (i.e., they may be negative; a negative cost *adds* lifetime to a ball that hits the bumper).

The remaining lines of the file represent the balls. Each line represents one ball, and contains four integers separated by space(s): the initial x and y position of the ball, the direction of movement, and its lifetime. The position will be in range (and not on top of any bumper or wall). The direction will be one of four values: 0 for increasing x (right), 1 for increasing y (up), 2 for decreasing x (left), and 3 for decreasing y (down). The lifetime will be some positive integer.

Output

There should be one line of output for each ball giving an integer number of points accumulated by that ball in the same order as the balls appear in the input. After all of these lines, the total points for all balls should be printed.

Sample Input

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

Sample Output

```
0
0
1
2
2
5
```

Expression trees, B and B* trees, red-black trees, quad trees, PQ trees; trees play a significant role in many domains of computer science. Sometimes the name of a problem may indicate that trees are used when they are not, as in the Artificial Intelligence planning problem traditionally called the *Monkey and Bananas problem*. Sometimes trees may be used in a problem whose name gives no indication that trees are involved, as in the *Huffman code*.

This problem involves determining how pairs of people who may be part of a "family tree" are related.

Given a sequence of *child-parent* pairs, where a pair consists of the child's name followed by the (single) parent's name, and a list of query pairs also expressed as two names, you are to write a program to determine whether the query pairs are related. If the names comprising a query pair are related the program should determine what the relationship is. Consider academic advisees and advisors as exemplars of such a single parent genealogy (we assume a single advisor, i.e., no co-advisors).

In this problem the child-parent pair $p\ q$ denotes that p is the child of q. In determining relationships between names we use the following definitions:

- p is a *0-descendent* of q (respectively *0-ancestor*) if and only if the child-parent pair $p\ q$ (respectively $q\ p$) appears in the input sequence of child-parent pairs.
- p is a *k-descendent* of q (respectively *k-ancestor*) if and only if the child-parent pair $p\ r$ (respectively $q\ r$) appears in the input sequence and r is a $(k-1)$-descendent of q (respectively p is a $(k-1)$-ancestor of r).

For the purposes of this problem the relationship between a person p and a person q is expressed as exactly one of the following four relations:

1. child — grand child, great grand child, great great grand child, *etc.*

   By definition p is the "child" of q if and only if the pair $p\ q$ appears in the input sequence of child-parent pairs (i.e., p is a 0-descendent of q); p is the "grand child" of q if and only if p is a 1-descendent of q; and

   $$p \text{ is the "} \underbrace{\text{great great } \ldots \text{ great}}_{n \text{ times}} \text{grand child" of } q$$

   if and only if p is an $(n+1)$-descendent of q.

2. parent — grand parent, great grand parent, great great grand parent, *etc.*

   By definition p is the "parent" of q if and only if the pair $q\ p$ appears in the input sequence of child-parent pairs (i.e., p is a 0-ancestor of q); p is the "grand parent" of q if and only if p is a 1-ancestor of q; and

   $$p \text{ is the "} \underbrace{\text{great great } \ldots \text{ great}}_{n \text{ times}} \text{grand parent" of } q$$

   if and only if p is an $(n+1)$-ancestor of q.

3. cousin — 0-th cousin, 1-st cousin, 2-nd cousin, *etc.*; cousins may be once removed, twice removed, three times removed, *etc.*

   By definition p and q are "cousins" if and only if they are related (i.e., there is a path from p to q in the implicit undirected parent-child tree). Let r represent the least common ancestor of p and q (i.e., no descendent of r is an ancestor of both p and q), where p is an m-descendent of r and q is an n-descendent of r.

   Then, by definition, cousins p and q are "k-th cousins" if and only if $k = \min(n, m)$, and, also by definition, p and q are "cousins removed j times" if and only if $j = \mid n - m \mid$.

4. sibling — 0-th cousins removed 0 times are "siblings" (they have the same parent).

Input

The input consists of child-parent pairs of names, one pair per line. Each name in a pair consists of lower-case alphabetic characters or periods (used to separate first and last names, for example). Child names are separated from parent names by one or more spaces. Child-parent pairs are terminated by a pair whose first component is the string '`no.child`'. Such a pair is NOT to be considered as a child-parent pair, but only as a delimiter to separate the child-parent pairs from the query pairs. There will be no circular relationships, i.e., no name p can be *both* an ancestor and a descendent of the same name q.

The child-parent pairs are followed by a sequence of query pairs in the same format as the child-parent pairs, i.e., each name in a query pair is a sequence of lower-case alphabetic characters and periods, and names are separated by one or more spaces. Query pairs are terminated by end-of-file.

There will be a maximum of 300 different names overall (child-parent and query pairs). All names will be fewer than 31 characters in length. There will be no more than 100 query pairs.

Output

For each query-pair $p\ q$ of names the output should indicate the relationship p *is-the-relative-of* q by the appropriate string of the form

- `child`, `grand child`, `great grand child`, `great great ... great grand child`
- `parent`, `grand parent`, `great grand parent`, `great great ... great grand parent`
- `sibling`
- n `cousin removed` m
- `no relation`

If an m-cousin is removed 0 times then only 'm `cousin`' should be printed, i.e., '`removed 0`' should NOT be printed. Do not print *st, nd, rd, th* after the numbers.

Sample Input

```
alonzo.church oswald.veblen
stephen.kleene alonzo.church
dana.scott alonzo.church
martin.davis alonzo.church
pat.fischer hartley.rogers
mike.paterson david.park
dennis.ritchie pat.fischer
hartley.rogers alonzo.church
les.valiant mike.paterson
bob.constable stephen.kleene
david.park hartley.rogers
no.child no.parent
stephen.kleene bob.constable
hartley.rogers stephen.kleene
les.valiant alonzo.church
les.valiant dennis.ritchie
dennis.ritchie les.valiant
pat.fischer michael.rabin
```

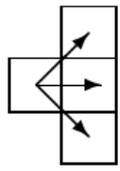
Sample Output

```
parent
sibling
great great grand child
1 cousin removed 1
1 cousin removed 1
no relation
```

Problems that require minimum paths through some domain appear in many different areas of computer science. For example, one of the constraints in VLSI routing problems is minimizing wire length. The Traveling Salesperson Problem (TSP) — finding whether all the cities in a salesperson's route can be visited exactly once with a specified limit on travel time — is one of the canonical examples of an NP-complete problem; solutions appear to require an inordinate amount of time to generate, but are simple to check.

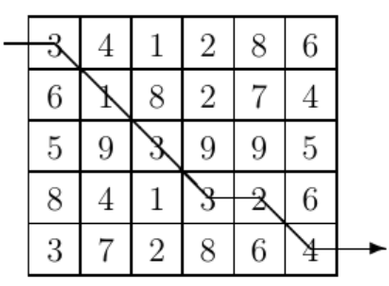
This problem deals with finding a minimal path through a grid of points while traveling only from left to right.

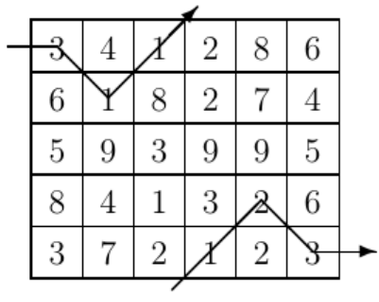
Given an $m \times n$ matrix of integers, you are to write a program that computes a path of minimal weight. A path starts anywhere in column 1 (the first column) and consists of a sequence of steps terminating in column n (the last column). A step consists of traveling from column i to column $i+1$ in an adjacent (horizontal or diagonal) row. The first and last rows (rows 1 and m) of a matrix are considered adjacent, i.e., the matrix "wraps" so that it represents a horizontal cylinder. Legal steps are illustrated on the right.



The *weight* of a path is the sum of the integers in each of the n cells of the matrix that are visited.

For example, two slightly different 5×6 matrices are shown below (the only difference is the numbers in the bottom row).





The minimal path is illustrated for each matrix. Note that the path for the matrix on the right takes advantage of the adjacency property of the first and last rows.

Input

The input consists of a sequence of matrix specifications. Each matrix specification consists of the row and column dimensions in that order on a line followed by $m \cdot n$ integers where m is the row dimension and n is the column dimension. The integers appear in the input in row major order, i.e., the first n integers constitute the first row of the matrix, the second n integers constitute the second row and so on. The integers on a line will be separated from other integers by one or more spaces. **Note:** integers are not restricted to being positive.

There will be one or more matrix specifications in an input file. Input is terminated by end-of-file.

For each specification the number of rows will be between 1 and 10 inclusive; the number of columns will be between 1 and 100 inclusive. No path's weight will exceed integer values representable using 30 bits.

Output

Two lines should be output for each matrix specification in the input file, the first line represents a minimal-weight path, and the second line is the cost of a minimal path. The path consists of a sequence of n integers (separated by one or more spaces) representing the rows that constitute the minimal path. If there is more than one path of minimal weight the path that is *lexicographically* smallest should be output.

Note: *Lexicographically* means the natural order on sequences induced by the order on their elements.

Sample Input

```
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6
3 7 2 8 6 4
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6
3 7 2 1 2 3
2 2
9 10 9 10
```

Sample Output

```
1 2 3 4 4 5
16
1 2 1 5 4 5
11
1 1
19
```

Graph algorithms form a very important part of computer science and have a lineage that goes back at least to Euler and the famous *Seven Bridges of Königsberg* problem. Many optimization problems involve determining efficient methods for reasoning about graphs.

This problem involves determining a route for a postal worker so that all mail is delivered while the postal worker walks a minimal distance, so as to rest weary legs.

Given a sequence of streets (connecting given intersections) you are to write a program that determines the minimal cost tour that traverses every street at least once. The tour must begin and end at the same intersection.

The "real-life" analogy concerns a postal worker who parks a truck at an intersection and then walks all streets on the postal delivery route (delivering mail) and returns to the truck to continue with the next route.

The cost of traversing a street is a function of the length of the street (there is a cost associated with delivering mail to houses and with walking even if no delivery occurs).

In this problem the number of streets that meet at a given intersection is called the *degree* of the intersection. There will be at most two intersections with odd degree. All other intersections will have even degree, i.e., an even number of streets meeting at that intersection.

Input

The input consists of a sequence of one or more postal routes. A route is composed of a sequence of street names (strings), one per line, and is terminated by the string '`deadend`' which is NOT part of the route. The first and last letters of each street name specify the two intersections for that street, the length of the street name indicates the cost of traversing the street. All street names will consist of lowercase alphabetic characters.

For example, the name `foo` indicates a street with intersections `f` and `o` of length 3, and the name `computer` indicates a street with intersections `c` and `r` of length 8. No street name will have the same first and last letter and there will be at most one street directly connecting any two intersections. As specified, the number of intersections with odd degree in a postal route will be at most two. In each postal route there will be a path between all intersections, i.e., the intersections are connected.

Output

For each postal route the output should consist of the cost of the minimal tour that visits all streets at least once. The minimal tour costs should be output in the order corresponding to the input postal routes.

Sample Input

```
one
two
three
deadend
mit
dartmouth
linkoping
tasmania
york
emory
cornell
duke
kaunas
hildesheim
concord
arkansas
williams
glasgow
deadend
```

Sample Output

```
11
114
```

Robotics, robot motion planning, and machine learning are areas that cross the boundaries of many of the subdisciplines that comprise Computer Science: artificial intelligence, algorithms and complexity, electrical and mechanical engineering to name a few. In addition, robots as "turtles" (inspired by work by Papert, Abelson, and diSessa) and as "beeper-pickers" (inspired by work by Pattis) have been studied and used by students as an introduction to programming for many years.

This problem involves determining the position of a robot exploring a pre-Columbian flat world.

Given the dimensions of a rectangular grid and a sequence of robot positions and instructions, you are to write a program that determines for each sequence of robot positions and instructions the final position of the robot.

A robot *position* consists of a grid coordinate (a pair of integers: x-coordinate followed by y-coordinate) and an orientation (`N`,`S`,`E`,`W` for north, south, east, and west). A robot *instruction* is a string of the letters '`L`', '`R`', and '`F`' which represent, respectively, the instructions:

- *Left*: the robot turns left 90 degrees and remains on the current grid point.
- *Right*: the robot turns right 90 degrees and remains on the current grid point.
- *Forward*: the robot moves forward one grid point in the direction of the current orientation and mantains the same orientation.

The direction *North* corresponds to the direction from grid point (x, y) to grid point $(x, y + 1)$.

Since the grid is rectangular and bounded, a robot that moves "off" an edge of the grid is lost forever. However, lost robots leave a robot "scent" that prohibits future robots from dropping off the world at the same grid point. The scent is left at the last grid position the robot occupied before disappearing over the edge. An instruction to move "off" the world from a grid point from which a robot has been previously lost is simply ignored by the current robot.

Input

The first line of input is the upper-right coordinates of the rectangular world, the lower-left coordinates are assumed to be 0,0.

The remaining input consists of a sequence of robot positions and instructions (two lines per robot). A position consists of two integers specifying the initial coordinates of the robot and an orientation (`N`,`S`,`E`,`W`), all separated by white space on one line. A robot instruction is a string of the letters '`L`', '`R`', and '`F`' on one line.

Each robot is processed sequentially, i.e., finishes executing the robot instructions before the next robot begins execution.

Input is terminated by end-of-file.

You may assume that all initial robot positions are within the bounds of the specified grid. The maximum value for any coordinate is 50. All instruction strings will be less than 100 characters in length.

Output

For each robot position/instruction in the input, the output should indicate the final grid position and orientation of the robot. If a robot falls off the edge of the grid the word '`LOST`' should be printed after the position and orientation.

Sample Input

```
5 3
1 1 E
RFRFRFRF
3 2 N
FRRFLLFFRRFLL
0 3 W
LLFFFLFLFL
```

Sample Output

```
1 1 E
3 3 N LOST
2 3 S
```

This problem involves determining, for a group of gift-giving friends, how much more each person gives than they receive (and vice versa for those that view gift-giving with cynicism).

In this problem each person sets aside some money for gift-giving and divides this money evenly among all those to whom gifts are given.

However, in any group of friends, some people are more giving than others (or at least may have more acquaintances) and some people have more money than others.

Given a group of friends, the money each person in the group spends on gifts, and a (sub)list of friends to whom each person gives gifts; you are to write a program that determines how much more (or less) each person in the group gives than they receive.

Input

The input is a sequence of gift-giving groups. A group consists of several lines:

- the number of people in the group,
- a list of the names of each person in the group,
- a line for each person in the group consisting of the name of the person, the amount of money spent on gifts, the number of people to whom gifts are given, and the names of those to whom gifts are given.

All names are lower-case letters, there are no more than 10 people in a group, and no name is more than 12 characters in length. Money is a non-negative integer less than 2000.

The input consists of one or more groups and is terminated by end-of-file.

Output

For each group of gift-givers, the name of each person in the group should be printed on a line followed by the net gain (or loss) received (or spent) by the person. Names in a group should be printed in the same order in which they first appear in the input.

The output for each group should be separated from other groups by a blank line. All gifts are integers. Each person gives the same integer amount of money to each friend to whom any money is given, and gives as much as possible. Any money not given is kept and is part of a person's "net worth" printed in the output.

Sample Input

```
5
dave laura owen vick amr
dave 200 3 laura owen vick
owen 500 1 dave
amr 150 2 vick owen
laura 0 2 amr vick
vick 0 0
3
liz steve dave
liz 30 1 steve
steve 55 2 liz dave
dave 0 2 steve liz
```

Sample Output

```
dave 302
laura 66
owen -359
vick 141
amr -150

liz -3
steve -24
dave 27
```

Stacks and Queues are often considered the bread and butter of data structures and find use in architecture, parsing, operating systems, and discrete event simulation. Stacks are also important in the theory of formal languages.

This problem involves both butter and sustenance in the form of pancakes rather than bread in addition to a finicky server who flips pancakes according to a unique, but complete set of rules.

Given a stack of pancakes, you are to write a program that indicates how the stack can be sorted so that the largest pancake is on the bottom and the smallest pancake is on the top. The size of a pancake is given by the pancake's diameter. All pancakes in a stack have different diameters.

Sorting a stack is done by a sequence of pancake "flips". A flip consists of inserting a spatula between two pancakes in a stack and flipping (reversing) all the pancakes on the spatula (reversing the sub-stack). A flip is specified by giving the position of the pancake on the bottom of the sub-stack to be flipped (relative to the whole stack). The pancake on the bottom of the whole stack has position 1 and the pancake on the top of a stack of n pancakes has position n.

A stack is specified by giving the diameter of each pancake in the stack in the order in which the pancakes appear.

For example, consider the three stacks of pancakes below (in which pancake 8 is the top-most pancake of the left stack):

8	7	2
4	6	5
6	4	8
7	8	4
5	5	6
2	2	7

The stack on the left can be transformed to the stack in the middle via *flip(3)*. The middle stack can be transformed into the right stack via the command *flip(1)*.

Input

The input consists of a sequence of stacks of pancakes. Each stack will consist of between 1 and 30 pancakes and each pancake will have an integer diameter between 1 and 100. The input is terminated by end-of-file. Each stack is given as a single line of input with the top pancake on a stack appearing first on a line, the bottom pancake appearing last, and all pancakes separated by a space.

Output

For each stack of pancakes, the output should echo the original stack on one line, followed by some sequence of flips that results in the stack of pancakes being sorted so that the largest diameter pancake is on the bottom and the smallest on top. For each stack the sequence of flips should be terminated by a '0' (indicating no more flips necessary). Once a stack is sorted, no more flips should be made.

Sample Input

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

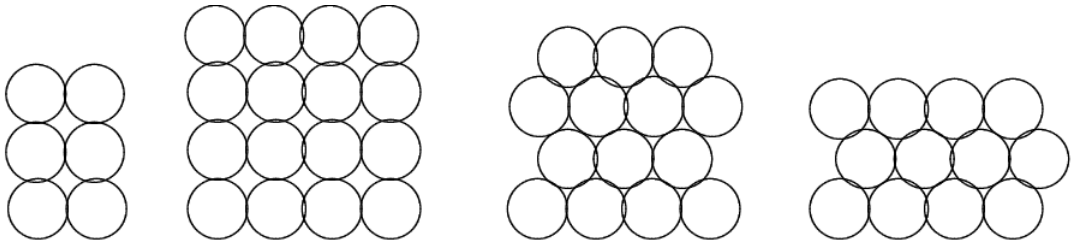
Sample Output

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

Filters, or programs that pass "processed" data through in some changed form, are an important class of programs in the UNIX operating system. A pipe is an operating system concept that permits data to "flow" between processes (and allows filters to be chained together easily.)

This problem involves maximizing the number of pipes that can be fit into a storage container (but it's a pipe fitting problem, not a bin packing problem).

A company manufactures pipes of uniform diameter. All pipes are stored in rectangular storage containers, but the containers come in several different sizes. Pipes are stored in rows within a container so that there is no space between pipes in any row (there may be some space at the end of a row), i.e., all pipes in a row are tangent, or touch. Within a rectangular cross-section, pipes are stored in either a *grid* pattern or a *skew* pattern as shown below: the two left-most cross-sections are in a grid pattern, the two right-most cross-sections are in a skew pattern.



grid patterns skew patterns

Note that although it may not be apparent from the diagram, there is no space between adjacent pipes in any row. The pipes in any row are tangent to (touch) the pipes in the row below (or rest on the bottom of the container). When pipes are packed into a container, there may be "left-over" space in which a pipe cannot be packed. Such left-over space is packed with padding so that the pipes cannot settle during shipping.

Input

The input is a sequence of cross-section dimensions of storage containers. Each cross-section is given as two real values on one line separated by white space. The dimensions are expressed in units of pipe diameters. All dimensions will be less than 2^7. Note that a cross section with dimensions $a \times b$ can also be viewed as a cross section with dimensions $b \times a$.

Output

For each cross-section in the input, your program should print the maximum number of pipes that can be packed into that cross section. The number of pipes is an integer — no fractional pipes can be packed. The maximum number is followed by the word '`grid`' if a grid pattern results in the maximal number of pipes or the word '`skew`' if a skew pattern results in the maximal number of pipes. If the pattern doesn't matter, that is the same number of pipes can be packed with either a grid or skew pattern, then the word '`grid`' should be printed.

Sample Input

```
3 3
2.9 10
2.9 10.5
11 11
```

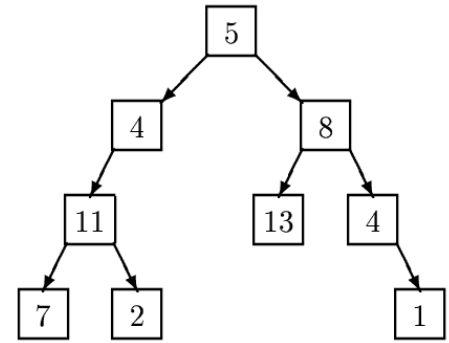
Sample Output

```
9 grid
29 skew
30 skew
126 skew
```

Trees are fundamental in many branches of computer science (Pun definitely intended). Current state-of-the art parallel computers such as Thinking Machines' CM-5 are based on *fat trees*. Quad- and octal-trees are fundamental to many algorithms in computer graphics.

This problem involves building and traversing binary trees.

Given a sequence of binary trees, you are to write a program that prints a level-order traversal of each tree. In this problem each node of a binary tree contains a positive integer and all binary trees have have fewer than 256 nodes.



In a *level-order* traversal of a tree, the data in all nodes at a given level are printed in left-to-right order and all nodes at level k are printed before all nodes at level $k+1$.

For example, a level order traversal of the tree on the right is: 5, 4, 8, 11, 13, 4, 7, 2, 1.

In this problem a binary tree is specified by a sequence of pairs '`(`n`,`s`)`' where n is the value at the node whose path from the root is given by the string s. A path is given be a sequence of '`L`'s and '`R`'s where '`L`' indicates a left branch and '`R`' indicates a right branch. In the tree diagrammed above, the node containing 13 is specified by `(13,RL)`, and the node containing 2 is specified by `(2,LLR)`. The root node is specified by `(5,)` where the empty string indicates the path from the root to itself. A binary tree is considered to be *completely specified* if every node on all root-to-node paths in the tree is given a value exactly once.

Input

The input is a sequence of binary trees specified as described above. Each tree in a sequence consists of several pairs '`(`n`,`s`)`' as described above separated by whitespace. The last entry in each tree is '`()`'. No whitespace appears between left and right parentheses.

All nodes contain a positive integer. Every tree in the input will consist of at least one node and no more than 256 nodes. Input is terminated by end-of-file.

Output

For each completely specified binary tree in the input file, the level order traversal of that tree should be printed. If a tree is not completely specified, i.e., some node in the tree is NOT given a value or a node is given a value more than once, then the string '`not complete`' should be printed.

Sample Input

```
(11,LL) (7,LLL) (8,R)
(5,) (4,L) (13,RL) (2,LLR) (1,RRR) (4,RR) ()
(3,L) (4,R) ()
```

Sample Output

```
5 4 8 11 13 4 7 2 1
not complete
```

Searching and sorting are part of the theory and practice of computer science. For example, binary search provides a good example of an easy-to-understand algorithm with sub-linear complexity. Quicksort is an efficient $O(n \log n)$ [average case] comparison based sort.

KWIC-indexing is an indexing method that permits efficient "human search" of, for example, a list of titles.

Given a list of titles and a list of "words to ignore", you are to write a program that generates a KWIC (Key Word In Context) index of the titles. In a KWIC-index, a title is listed once for each keyword that occurs in the title. The KWIC-index is alphabetized by keyword.

Any word that is not one of the "words to ignore" is a potential keyword.

For example, if words to ignore are "`the, of, and, as, a`" and the list of titles is:

```
Descent of Man
The Ascent of Man
The Old Man and The Sea
A Portrait of The Artist As a Young Man
```

A KWIC-index of these titles might be given by:

```
                    a portrait of the ARTIST as a young man
                                  the ASCENT of man
                                      DESCENT of man
                               descent of MAN
                            the ascent of MAN
                                 the old MAN and the sea
  a portrait of the artist as a young MAN
                                  the OLD man and the sea
                                    a PORTRAIT of the artist as a young man
                the old man and the SEA
      a portrait of the artist as a YOUNG man
```

Input

The input is a sequence of lines, the string '`::`' is used to separate the list of words to ignore from the list of titles. Each of the words to ignore appears in lower-case letters on a line by itself and is no more than 10 characters in length. Each title appears on a line by itself and may consist of mixed-case (upper and lower) letters. Words in a title are separated by whitespace. No title contains more than 15 words.

There will be no more than 50 words to ignore, no more than than 200 titles, and no more than 10,000 characters in the titles and words to ignore combined. No characters other than '`a`'–'`z`', '`A`'–'`Z`', and white space will appear in the input.

Output

The output should be a KWIC-index of the titles, with each title appearing once for each keyword in the title, and with the KWIC-index alphabetized by keyword. If a word appears more than once in a title, each instance is a potential keyword.

The keyword should appear in all upper-case letters. All other words in a title should be in lower-case letters. Titles in the KWIC-index with the same keyword should appear in the same order as they appeared in the input file. In the case where multiple instances of a word are keywords in the same title, the keywords should be capitalized in left-to-right order.

Case (upper or lower) is irrelevant when determining if a word is to be ignored.

The titles in the KWIC-index need NOT be justified or aligned by keyword, all titles may be listed left-justified.

Sample Input

```
is
the
of
and
as
a
but
::
Descent of Man
The Ascent of Man
The Old Man and The Sea
A Portrait of The Artist As a Young Man
A Man is a Man but Bubblesort IS A DOG
```

Sample Output

```
a portrait of the ARTIST as a young man
the ASCENT of man
a man is a man but BUBBLESORT is a dog
DESCENT of man
a man is a man but bubblesort is a DOG
descent of MAN
the ascent of MAN
the old MAN and the sea
a portrait of the artist as a young MAN
a MAN is a man but bubblesort is a dog
a man is a MAN but bubblesort is a dog
the OLD man and the sea
a PORTRAIT of the artist as a young man
the old man and the SEA
a portrait of the artist as a YOUNG man
```

Order is an important concept in mathematics and in computer science. For example, Zorn's Lemma states: "a partially ordered set in which every chain has an upper bound contains a maximal element." Order is also important in reasoning about the fix-point semantics of programs.

This problem involves neither Zorn's Lemma nor fix-point semantics, but does involve order.

Given a list of variable constraints of the form $x < y$, you are to write a program that prints all orderings of the variables that are consistent with the constraints.

For example, given the constraints $x < y$ and $x < z$ there are two orderings of the variables x, y, and z that are consistent with these constraints: xyz and xzy.

Input

The input consists of a sequence of constraint specifications. A specification consists of two lines: a list of variables on one line followed by a list of constraints on the next line. A constraint is given by a pair of variables, where 'x y' indicates that $x < y$.

All variables are single character, lower-case letters. There will be at least two variables, and no more than 20 variables in a specification. There will be at least one constraint, and no more than 50 constraints in a specification. There will be at least one, and no more than 300 orderings consistent with the constraints in a specification.

Input is terminated by end-of-file.

Output

For each constraint specification, all orderings consistent with the constraints should be printed. Orderings are printed in lexicographical (alphabetical) order, one per line.

Output for different constraint specifications is separated by a blank line.

Sample Input

```
a b f g
a b b f
v w x y z
v y x v z v w v
```

Sample Output

```
abfg
abgf
agbf
gabf

wxzvy
wzxvy
xwzvy
xzwvy
zwxvy
zxwvy
```

Problems that process input and generate a simple "yes" or "no" answer are called decision problems. One class of decision problems, the NP-complete problems, are not amenable to general efficient solutions. Other problems may be simple as decision problems, but enumerating all possible "yes" answers may be very difficult (or at least time-consuming).

This problem involves determining the number of routes available to an emergency vehicle operating in a city of one-way streets.

Given the intersections connected by one-way streets in a city, you are to write a program that determines the number of different routes between each intersection. A route is a sequence of one-way streets connecting two intersections.

Intersections are identified by non-negative integers. A one-way street is specified by a pair of intersections. For example, j k indicates that there is a one-way street from intersection j to intersection k. Note that two-way streets can be modeled by specifying two one-way streets: j k and k j.

Consider a city of four intersections connected by the following one-way streets:

```
0 1
0 2
1 2
2 3
```

There is one route from intersection 0 to 1, two routes from 0 to 2 (the routes are $0 \rightarrow 1 \rightarrow 2$ and $0 \rightarrow 2$), one route from 0 to 3, one route from 1 to 2, one route from 1 to 3, one route from 2 to 3, and no other routes.

It is possible for an infinite number of different routes to exist. For example if the intersections above are augmented by the street 3 2, there is still only one route from 0 to 1, but there are infinitely many different routes from 0 to 2. This is because the street from 2 to 3 and back to 2 can be repeated yielding a different sequence of streets and hence a different route. Thus the route $0 \rightarrow 2 \rightarrow 3 \rightarrow 2 \rightarrow 3 \rightarrow 2$ is a different route than $0 \rightarrow 2 \rightarrow 3 \rightarrow 2$.

Input

The input is a sequence of city specifications. Each specification begins with the number of one-way streets in the city followed by that many one-way streets given as pairs of intersections. Each pair 'j k' represents a one-way street from intersection j to intersection k. In all cities, intersections are numbered sequentially from 0 to the "largest" intersection. All integers in the input are separated by whitespace. The input is terminated by end-of-file.

There will never be a one-way street from an intersection to itself. No city will have more than 30 intersections.

Output

For each city specification, a square matrix of the number of different routes from intersection j to intersection k is printed. If the matrix is denoted M, then $M[j][k]$ is the number of different routes from intersection j to intersection k. The matrix M should be printed in row-major order, one row per line. Each matrix should be preceded by the string '`matrix for city` k' (with k appropriately instantiated, beginning with 0).

If there are an infinite number of different paths between two intersections a '`-1`' should be printed. **DO NOT** worry about justifying and aligning the output of each matrix. All entries in a row should be separated by whitespace.

Sample Input

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

Sample Output

```
matrix for city 0
0 4 1 3 2
0 0 0 0 0
0 2 0 2 1
0 1 0 0 0
0 1 0 1 0
matrix for city 1
0 2 1 0 0 3
0 0 0 0 0 1
0 1 0 0 0 2
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
matrix for city 2
-1 -1 -1 -1 -1
0 0 0 0 1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
0 0 0 0 0
```

The well-known physicist Alfred E Neuman is working on problems that involve multiplying polynomials of x and y. For example, he may need to calculate

$$(-x^8y + 9x^3 - 1 + y).(x^5y + 1 + x^3)$$

getting the answer

$$-x^{13}y^2 - x^{11}y + 8x^8y + 9x^6 - x^5y + x^5y^2 + 8x^3 + x^3y - 1 + y$$

Unfortunately, such problems are so trivial that the great man's mind keeps drifting off the job, and he gets the wrong answers. As a consequence, several nuclear warheads that he has designed have detonated prematurely, wiping out five major cities and a couple of rain forests.

You are to write a program to perform such multiplications and save the world.

Input

The file of input data will contain pairs of lines, with each line containing no more than 80 characters. The final line of the input file contains a '`#`' as its first character. Each input line contains a polynomial written without spaces and without any explicit exponentiation operator. Exponents are positive non-zero unsigned integers. Coefficients are also integers, but may be negative. Both exponents and coefficients are less than or equal to 100 in magnitude. Each term contains at most one factor in x and one in y.

Output

Your program must multiply each pair of polynomials in the input, and print each product on a pair of lines, the first line containing all the exponents, suitably positioned with respect to the rest of the information, which is in the line below.

The following rules control the output format:

1. Terms in the output line must be sorted in decreasing order of powers of x and, for a given power of x, in increasing order of powers of y.
2. Like terms must be combined into a single term. For example, $40x^2y^3 - 40x^2y^3$ is replaced by $2x^2y^3$.
3. Terms with a zero coefficient must not be displayed.
4. Coefficients of 1 are omitted, except for the case of a constant term of 1.
5. Exponents of 1 are omitted.
6. Factors of x^0 and y^0 are omitted.
7. Binary pluses and minuses (that is the pluses and minuses connecting terms in the output) have a single blank column both before and after.
8. If the coefficient of the first term is negative, it is preceded by a unary minus in the first column, with no intervening blank column. Otherwise, the coefficient itself begins in the first output column.
9. The output can be assumed to fit into a single line of at most 80 characters in length.
10. There should be no blank lines printed between each pair of output lines.
11. The pair of lines that contain a product should be the same length — trailing blanks should appear after the last non-blank character of the shorter line to achieve this.

Sample Input

```
-yx8+9x3-1+y
x5y+1+x3
1
1
#
```

Sample Output

```
  13 2    11      8      6    5     5 2     3    3
-x  y  - x  y + 8x y + 9x  - x y + x y  + 8x  + x y - 1 + y

1
```

You are to simulate the playing of games of "Accordian" patience, the rules for which are as follows:

> Deal cards one by one in a row from left to right, not overlapping. Whenever the card matches its immediate neighbour on the left, or matches the third card to the left, **it** may be moved onto that card. Cards match if they are of the same suit or same rank. After making a move, look to see if it has made additional moves possible. Only the top card of each pile may be moved at any given time. Gaps between piles should be closed up as soon as they appear by moving all piles on the right of the gap one position to the left. Deal out the whole pack, combining cards towards the left whenever possible. The game is won if the pack is reduced to a single pile.

Situations can arise where more than one play is possible. Where two cards may be moved, you should adopt the strategy of always moving the leftmost card possible. Where a card may be moved either one position to the left or three positions to the left, move it three positions.

Input

Input data to the program specifies the order in which cards are dealt from the pack. The input contains pairs of lines, each line containing 26 cards separated by single space characters. The final line of the input file contains a '`#`' as its first character. Cards are represented as a two character code. The first character is the face-value (`A`=Ace, `2`–`9`, `T`=10, `J`=Jack, `Q`=Queen, `K`=King) and the second character is the suit (`C`=Clubs, `D`=Diamonds, `H`=Hearts, `S`=Spades).

Output

One line of output must be produced for each pair of lines (that between them describe a pack of 52 cards) in the input. Each line of output shows the number of cards in each of the piles remaining after playing "Accordian patience" with the pack of cards as described by the corresponding pairs of input lines.

Sample Input

```
QD AD 8H 5S 3H 5H TC 4D JH KS 6H 8S JS AC AS 8D 2H QS TS 3S AH 4H TH TD 3C 6S
8C 7D 4C 4S 7S 9H 7C 5D 2S KD 2D QH JD 6D 9D JC 2C KH 3D QC 6C 9S KC 7H 9C 5C
AC 2C 3C 4C 5C 6C 7C 8C 9C TC JC QC KC AD 2D 3D 4D 5D 6D 7D 8D TD 9D JD QD KD
AH 2H 3H 4H 5H 6H 7H 8H 9H KH 6S QH TH AS 2S 3S 4S 5S JH 7S 8S 9S TS JS QS KS
#
```

Sample Output

```
6 piles remaining: 40 8 1 1 1 1
1 pile remaining: 52
```

You work for a company which uses lots of personal computers. Your boss, Dr Penny Pincher, has wanted to link the computers together for some time but has been unwilling to spend any money on the Ethernet boards you have recommended. You, unwittingly, have pointed out that each of the PCs has come from the vendor with an asynchronous serial port at no extra cost. Dr Pincher, of course, recognizes her opportunity and assigns you the task of writing the software necessary to allow communication between PCs.

You've read a bit about communications and know that every transmission is subject to error and that the typical solution to this problem is to append some error checking information to the end of each message. This information allows the receiving program to detect when a transmission error has occurred (in most cases). So, off you go to the library, borrow the biggest book on communications you can find and spend your weekend (unpaid overtime) reading about error checking.

Finally you decide that CRC (cyclic redundancy check) is the best error checking for your situation and write a note to Dr Pincher detailing the proposed error checking mechanism noted below.

> **CRC Generation**
> The message to be transmitted is viewed as a long positive binary number. The first byte of the message is treated as the most significant byte of the binary number. The second byte is the next most significant, etc. This binary number will be called "m" (for message). Instead of transmitting "m" you will transmit a message, "m2", consisting of "m" followed by a two-byte CRC value.
>
> The CRC value is chosen so that "m2" when divided by a certain 16-bit value "g" leaves a remainder of 0. This makes it easy for the receiving program to determine whether the message has been corrupted by transmission errors. It simply divides any message received by "g". If the remainder of the division is zero, it is assumed that no error has occurred.
>
> You notice that most of the suggested values of "g" in the book are odd, but don't see any other similarities, so you select the value 34943 for "g" (the generator value).

You are to devise an algorithm for calculating the CRC value corresponding to any message that might be sent. To test this algorithm you will write a program which reads lines from standard input and writes to standard output.

Input

Each input line will contain no more than 1024 ASCII characters (each line being all characters up to, but not including the end of line character) as input,

The input is terminated by a line that contains a '`#`' in column 1.

Output

For each input line calculates the CRC value for the message contained in the line, and writes the numeric value of the CRC bytes (in hexadecimal notation) on an output line.

Note that each CRC printed should be in the range 0 to 34942 (decimal).

Sample Input

```
this is a test

A
#
```

Sample Output

```
77 FD
00 00
0C 86
```

You have been employed by the organisers of a Super Krypton Factor Contest in which contestants have very high mental and physical abilities. In one section of the contest the contestants are tested on their ability to recall a sequenace of characters which has been read to them by the Quiz Master. Many of the contestants are very good at recognising patterns. Therefore, in order to add some difficulty to this test, the organisers have decided that sequences containing certain types of repeated subsequences should not be used. However, they do not wish to remove all subsequences that are repeated, since in that case no single character could be repeated. This in itself would make the problem too easy for the contestants. Instead it is decided to eliminate all sequences containing an occurrence of two adjoining identical subsequences. Sequences containing such an occurrence will be called "easy". Other sequences will be called "hard".

For example, the sequence ABACBCBAD is easy, since it contains an adjoining repetition of the subsequence CB. Other examples of easy sequences are:

- BB
- ABCDACABCAB
- ABCDABCD

Some examples of hard sequences are:

- D
- DC
- ABDAB
- CBABCBA

In order to provide the Quiz Master with a potentially unlimited source of questions you are asked to write a program that will read input lines from standard input and will write to standard output.

Input

Each input line contains integers n and L (in that order), where $n > 0$ and L is in the range $1 \leq L \leq 26$.

Input is terminated by a line containing two zeroes.

Output

For each input line prints out the n-th hard sequence (composed of letters drawn from the first L letters in the alphabet), in increasing alphabetical order (Alphabetical ordering here corresponds to the normal ordering encountered in a dictionary), followed (on the next line) by the length of that sequence. The first sequence in this ordering is '`A`'. You may assume that for given n and L there do exist at least n hard sequences.

As such a sequence is potentially very long, split it into groups of four (4) characters separated by a space. If there are more than 16 such groups, please start a new line for the 17th group.

Your program may assume a maximum sequence length of 80.

For example, with $L = 3$, the first 7 hard sequences are:

```
A
AB
ABA
ABAC
ABACA
ABACAB
ABACABA
```

Sample Input

```
7 3
30 3
0 0
```

Sample Output

```
ABAC ABA
7
ABAC ABCA CBAB CABA CABC ACBA CABA
28
```