ACM International Collegiate Programming Contest
2001 East Central Regional Contest
Ashland University
University of Cincinnati
Western Michigan University
Sheridan University
November 10, 2001

Sponsored by IBM

Rules:

1. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
2. All programs will be re-compiled prior to testing with the judges' data.
3. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed.
4. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
5. All communication with the judges will be handled by the PC^2 environment.
6. The allowed programming languages are C, C++ and Java.
7. Judges' decisions are to be considered final. No cheating will be tolerated.
8. There are **eight** questions to be completed in **five hours**.

Problem A: What's In A Name?

The FBI is conducting a surveillance of a known criminal hideout which serves as a communication center for a number of men and women of nefarious intent. Using sophisticated decryption software and good old fashion wiretaps, they are able to decode any e-mail messages leaving the site. However, before any arrest warrants can be served, they must match actual names with the user ID's on the messages. While these criminals are evil, they're not stupid, so they use random strings of letters for their ID's (no `dillingerj` ID's found here). The FBI knows that each criminal uses only one ID. The only other information they have which will help them is a log of names of the people who enter and leave the hideout. In many cases, this is enough to link the names to the ID's.

Input

Input consists of one problem instance. The first line contains a single positive integer n indicating the number of criminals using the hideout. The maximum value for n will be 20. The next line contains the n user ID's, separated by single spaces. Next will be the log entries in chronological order. Each entry in the log has the form *type arg* , where *type* is either `E`, `L` or `M`: `E` indicates that criminal *arg* has entered the hideout; `L` indicates criminal *arg* has left the hideout; `M` indicates a message was intercepted from user ID *arg*. A line containing only the letter `Q` indicates the end of the log. Note that not all user ID's may be present in the log but each criminal name will be guaranteed to be in the log at least once. At the start of the log, the hideout is presumed to be empty. All names and user ID's consist of only lowercase letters and have length at most 20. Note: The line containing only the user ID's may contain more than 80 characters.

Output

Output consists of n lines, each containing a list of criminal names and their corresponding user ID's, if known. The list should be sorted in alphabetical order by the criminal names. Each line has the form *name:userid* , where *name* is the criminal's name and *userid* is either their user ID or the string `???` if their user ID could not be determined from the surveillance log.

Sample Input

```
7
bigman mangler sinbad fatman bigcheese frenchie capodicapo
E mugsy
E knuckles
M bigman
M mangler
L mugsy
E clyde
E bonnie
M bigman
M fatman
M frenchie
L clyde
M fatman
E ugati
M sinbad
E moriarty
E booth
Q
```

Sample Output

```
bonnie:fatman
booth:???
clyde:frenchie
knuckles:bigman
moriarty:???
mugsy:mangler
ugati:sinbad
```

Problem B: Sorting It All Out

An ascending sorted sequence of distinct values is one in which some form of a less-than operator is used to order the elements from smallest to largest. For example, the sorted sequence A, B, C, D implies that $A < B$, $B < C$ and $C < D$. In this problem, we will give you a set of relations of the form $A < B$ and ask you to determine whether a sorted order has been specified or not.

Input

Input consists of multiple problem instances. Each instance starts with a line containing two positive integers n and m. The first value indicates the number of objects to sort, where $2 \leq n \leq 26$. The objects to be sorted will be the first n characters of the uppercase alphabet. The second value m indicates the number of relations of the form $A < B$ which will be given in this problem instance. Next will be m lines, each containing one such relation consisting of three characters: an uppercase letter, the character "`<`" and a second uppercase letter. No letter will be outside the range of the first n letters of the alphabet. Values of $n = m = 0$ indicate end of input.

Output

For each problem instance, output consists of one line. This line should be one of the following three:

`Sorted sequence determined after` *xxx* `relations:` *yyy...y*.
`Sorted sequence cannot be determined.`
`Inconsistency found after` *xxx* `relations.`

where *xxx* is the number of relations processed at the time either a sorted sequence is determined or an inconsistency is found, whichever comes first, and *yyy...y* is the sorted, ascending sequence.

Sample Input

```
4 6
A<B
A<C
B<C
C<D
B<D
A<B
3 2
A<B
B<A
26 1
A<Z
0 0
```

Sample Output

```
Sorted sequence determined after 4 relations: ABCD.
Inconsistency found after 2 relations.
Sorted sequence cannot be determined.
```

Problem C: Web Navigation

Standard web browsers contain features to move backward and forward among the pages recently visited. One way to implement these features is to use two stacks to keep track of the pages that can be reached by moving backward and forward. In this problem, you are asked to implement this.

The following commands need to be supported:

BACK: Push the current page on the top of the forward stack. Pop the page from the top of the backward stack, making it the new current page. If the backward stack is empty, the command is ignored.

FORWARD: Push the current page on the top of the backward stack. Pop the page from the top of the forward stack, making it the new current page. If the forward stack is empty, the command is ignored.

VISIT ⟨url⟩: Push the current page on the top of the backward stack, and make the URL specified the new current page. The forward stack is emptied.

QUIT: Quit the browser.

Assume that the browser initially loads the web page at the URL `http://www.acm.org/`

Input

Input is a sequence of commands. The command keywords `BACK`, `FORWARD`, `VISIT`, and `QUIT` are all in uppercase. URLs have no whitespace and have at most 70 characters. You may assume that no problem instance requires more than 100 elements in each stack at any time. The end of input is indicated by the `QUIT` command.

Output

For each command other than `QUIT`, print the URL of the current page after the command is executed if the command is not ignored. Otherwise, print "`Ignored`". The output for each command should be printed on its own line. No output is produced for the `QUIT` command.

Sample Input

```
VISIT http://acm.ashland.edu/
VISIT http://acm.baylor.edu/acmicpc/
BACK
BACK
BACK
FORWARD
VISIT http://www.ibm.com/
BACK
BACK
FORWARD
FORWARD
FORWARD
QUIT
```

Sample Output

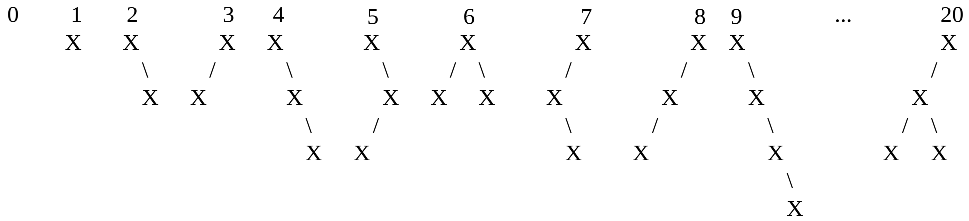
```
http://acm.ashland.edu/
http://acm.baylor.edu/acmicpc/
http://acm.ashland.edu/
http://www.acm.org/
Ignored
http://acm.ashland.edu/
http://www.ibm.com/
http://acm.ashland.edu/
http://www.acm.org/
http://acm.ashland.edu/
http://www.ibm.com/
Ignored
```

Problem D: Trees Made to Order

We can number binary trees using the following scheme:

- The empty tree is numbered 0.
- The single-node tree is numbered 1.
- All binary trees having m nodes have numbers less than all those having $m+1$ nodes.
- Any binary tree having m nodes with left and right subtrees L and R is numbered n such that all trees having m nodes numbered $> n$ have either
  - left subtrees numbered higher than L, or
  - a left subtree = L and a right subtree numbered higher than R.

The first 10 binary trees and tree number 20 in this sequence are shown below:



Your job for this problem is to output a binary tree when given its order number.

Input

Input consists of multiple problem instances. Each instance consists of a single integer n, where $1 \leq n \leq 500{,}000{,}000$. A value of $n = 0$ terminates input. (Note that this means you will never have to output the empty tree.)

Output

For each problem instance, you should output one line containing the tree corresponding to the order number for that instance. To print out the tree, use the following scheme:

- A tree with no children should be output as `X`.
- A tree with left and right subtrees `L` and `R` should be output as `(L')X(R')`, where `L'` and `R'` are the representations of `L` and `R`.
  - If `L` is empty, just output `X(R')`.
  - If `R` is empty, just output `(L')X`.

Sample Input

```
1
20
31117532
0
```

Sample Output

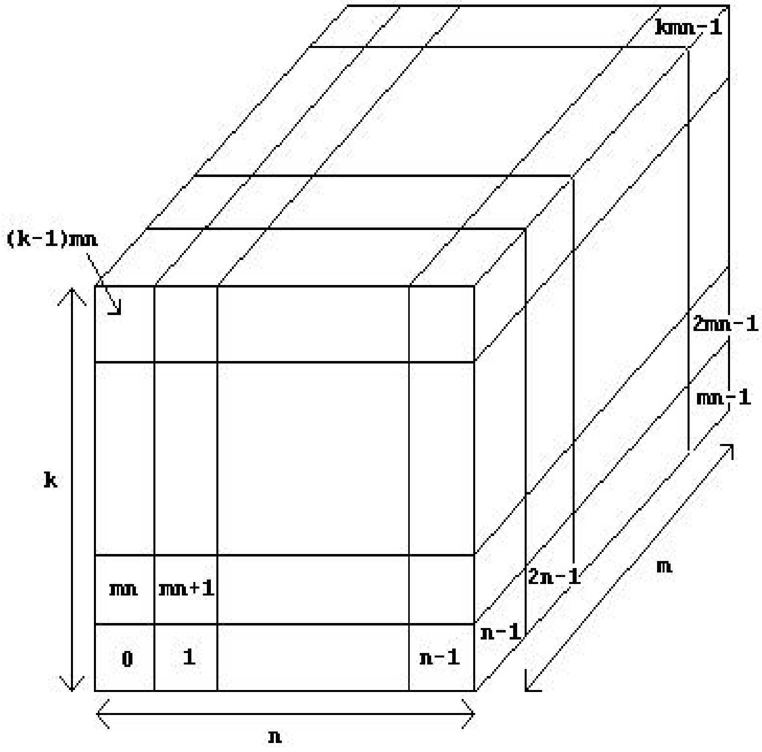
```
X
((X)X(X))X
(X(X(((X(X))X(X))X(X))))X(((X((X)X((X)X)))X)X)
```

Problem E: Space Station Shielding

Roger Wilco is in charge of the design of a low orbiting space station for the planet Mars. To simplify construction, the station is made up of a series of Airtight Cubical Modules (ACM's), which are connected together once in space. One problem that concerns Roger is that of (potentially) lethal bacteria that may reside in the upper atmosphere of Mars. Since the station will occasionally fly through the upper atmosphere, it is imperative that extra shielding be used on all faces of the ACMs which make up the external surface of the station. Roger has made certain that where two ACMs touch, either edge-to-edge or face-to-face, that joint is sealed so no bacteria can sneak through. Any face of an ACM shared by another ACM will not need shielding, of course, nor will a face which cannot be reached from the outside. Roger could just put extra shielding on all of the faces of every ACM, but the cost would be prohibitive. Therefore, he wants to know the exact number of ACM faces which need the extra shielding.

Input

Input consists of multiple problem instances. Each instance consists of a specification of a space station. We assume that each space station can fit into an $n \times m \times k$ grid ($1 \leq n, m, k \leq 60$), where each grid cube may or may not contain an ACM. We number the grid cubes $0, 1, 2, \ldots, kmn - 1$ as shown in the diagram below. Each space station specification then consists of the following: the first line contains four positive integers $n\ m\ k\ l$, where n, m and k are as described above and l is the number of ACM's in the station. This is followed by a set of lines which specify the l grid locations of the ACM's. Each of these lines contain 10 integers (except possibly the last). Each space station is fully connected (i.e., an astronaut can move from one ACM to any other ACM in the station without leaving the station). Values of $n = m = k = l = 0$ terminate input.



Output

For each problem instance, you should output one line of the form

`The number of faces needing shielding is` s.

where s is for you to determine.

Sample Input

```
2 2 1 3
0 1 3
3 3 3 26
0 1 2 3 4 5 6 7 8 9
10 11 12 14 15 16 17 18 19 20
21 22 23 24 25 26
0 0 0 0
```

Sample Output

```
The number of faces needing shielding is 14.
The number of faces needing shielding is 54.
```

Problem F: Roads Scholar

The Hines Sign company has been contracted to supply roadside signs for the state highway system. The head of the company has put his son Myles Hines in charge of one particular class of signs, those which indicate the number of miles to various towns. Myles is given a layout of the highway system and a set of locations where the signs should go; he is in charge of determining the mileage to nearby cities. To select which cities should be listed on any sign, he uses the following strategy: City X is put on the sign if the sign is on a road such that the shortest path from the intersection immediately preceeding the sign to X uses the road. You may assume that there is only one shortest path between each pair of intersections.

Input

Input consists of a single problem instance consisting of a description of the highway system, followed by a set of sign locations. The highway system is defined as a set of intersections (some of which are also city locations) and a set of roads connecting the intersections. The first line of a problem instance contains three positive integers n m k: n specifies the number of intersections (numbered $0, 1, 2, \cdots, n-1$), m indicates the number of roads between connections, and k indicates the number of intersections which are also cities. Following this are m lines of the form i_1 i_2 d, which specifies a two-way road between intersections i_1 and i_2 of distance d. The next k lines are of the form i *name*, which specifies that intersection i is a city called *name*. After this is a line with a single positive integer s indicating the number of signs to place on the highway. The remaining s lines are of the form i_1 i_2 d, indicating that a sign is to be placed on the road going from i_1 to i_2 a distance d from i_1 (d will always be non-zero and less than the distance from i_1 to i_2). For all problem instances, the length of *name* will be ≤ 18 characters, and $5 \leq n \leq 30$. All distances will be non-zero and to the nearest hundredth mile.

Output

Each sign should be output as follows:

$name_1$ d_1
$name_2$ d_2
...

where each $name_i$ should be left justified in a field of width 20, and each distance d_i is rounded to the nearest mile. (Round .50 up. For example, 7.50 should be rounded to 8.) Each name-distance pair should be sorted by the rounded distance; pairs with the same rounded distance should be printed in alphabetical order. Signs should be output in the order in which they were listed in the input, and you should separate each sign output with a blank line. You may assume that every sign will have at least one city listed on it.

Sample Input

```
8 17 4
0 1 7.12
0 2 8.34
0 3 5.33
0 4 5.36
1 2 4.21
1 6 6.99
1 7 10.26
2 3 2.74
2 6 5.04
3 4 4.12
3 5 7.72
3 6 5.71
4 5 8.94
4 6 10.29
5 6 5.47
5 7 8.55
6 7 6.01
0 Allentown
1 Bobtown
6 Charlestown
7 Downville
3
0 3 2.17
3 2 0.45
4 3 3.14
```

Sample Output

```
Charlestown         9
Downville           15

Bobtown             7

Charlestown         7
Bobtown             8
Downville           13
```

Problem G: Robots

The Robots game is a one-player game played on a 31×31 board. The board is partitioned into 1×1 cells arranged in 31 rows and 31 columns. Each cell is indexed by (r, c) where r is the row and c is the column (starting from 1), and it may be empty, occupied by you, occupied by a robot, or occupied by debris. The object of this game is to move in such a way to destroy all the robots before the robots destroy you.

Initially, you occupy the cell at $(15, 15)$, and there are R robots ($1 \leq R \leq 50$) located in R different cells other than $(15, 15)$. All other cells are empty. You are also given a list of T ($0 \leq T \leq 20$) cells that are potential teleport locations. You make the first move, and then the robots and you alternate moves. At your move, you are allowed to walk to an adjacent cell in any one of the eight compass directions, teleport to one of the specified teleport locations, or remain stationary. You may walk to an adjacent cell if it is empty. In addition, you may walk to an adjacent cell which contains debris by pushing the debris to the next adjacent cell along the line of movement, provided that the next cell does not already contain debris. If a cell that debris is pushed to contains a robot, that robot is destroyed. If you choose to teleport, the destination must be an empty cell. You may not make any move that will leave you or any debris that you push outside of the board.

When the robots move, each robot walks to the adjacent cell (even if it is not empty) in the eight compass directions such that the destination cell is closest to your current position (i.e. after your last move). The distance between two cells (r_1, c_1) and (r_2, c_2) is defined to be $|r_1 - r_2| + |c_1 - c_2|$. All robots walk at the same time during a move. If two or more robots walk to the same cell, or if a robot walks to a cell containing debris, all of these robots are destroyed. Destroyed robots become debris.

You lose the game if any robot walks to your current position, even if multiple robots do so and destroy each other. You win the game if all robots are destroyed and none has moved to your current position.

In order to stay in the game as long as possible, you will only consider moves that do not lead to an immediate loss (a loss before your next move). A plausible strategy is to always walk to a cell (or remain stationary) such that the number of robots remaining after your move and the robots' move (i.e. just before your next move) is minimized. In case of a tie, choose the move that maximizes the minimum distance to the remaining robots just before your next move. If there are still ties, choose the move that also minimizes the row index of the destination cell, and finally, break remaining ties by also minimizing the column index.

If it is not possible to make a move by walking or by remaining stationary without leading to an immediate loss, you should teleport to the first unused legal destination chosen from a list of locations given to you, as long as it does not lead to an immediate loss. When you search for a teleport site, you should always start the search at the beginning of the list. If no such teleport destination is available, you should remain stationary, leading to an immediate loss.

In this problem, you will implement this strategy and see how well it works.

Input

The input consists of a number of instances. The first line of each instance contains the integers R and T separated by a space. This is followed by R lines containing two integers separated by a space, indicating the row and column of the initial positions of the R robots. You may assume that each robot initially occupies a cell which is not $(15, 15)$ and the locations of the robots are distinct. The next T lines give the list of teleport destinations available. Each line is given by the row and column of the destination cell, separated by a space. The input is terminated by a case with $R = T = 0$.

Output

For each case, print the case number (starting from 1), in the format shown in the Sample Output, on its own line. For each teleport taken, print one line of the form:

`Move` m`: teleport to` (r, c)

where m is the number of moves you have made (including this one), and (r, c) is the destination of the teleport. This is followed by three lines containing the result of the game. If you win the game, print

`Won game after making m moves.`
`Final position:` (r, c)
`Number of cells with debris:` d

where m is the number of moves you have made when you won the game, (r, c) is your final position, and d is the number of cells with debris (use the word "moves" even if $m = 1$). If you lose the game, print

`Lost game after making m moves.`
`Final position:` (r, c)
`Number of cells with debris:` d
`Number of robots remaining:` n

where m is the number of moves you have made when you lost the game, (r, c) is the location at which you are destroyed, d is the number of cells with debris, and n is the number of robots remaining when you lost the game (use the word "moves" even if $m = 1$).

Separate the output for each case by a blank line.

Sample Input

```
4 0
17 18
13 18
8 12
10 12
4 0
17 17
13 17
13 13
17 13
3 3
17 18
13 18
5 31
15 16
16 15
3 7
0 0
```

Sample Output

```
Case 1:
Won game after making 5 moves.
Final position: (14,16)
Number of cells with debris: 1

Case 2:
Lost game after making 2 moves.
Final position: (15,15)
Number of cells with debris: 1
Number of robots remaining: 0

Case 3:
Move 30: teleport to (16,15)
Move 58: teleport to (15,16)
Move 86: teleport to (3,7)
Lost game after making 114 moves.
Final position: (1,29)
Number of cells with debris: 1
Number of robots remaining: 1
```

Problem H: Square Ice

Square Ice is a two-dimensional arrangement of water molecules H_2O, with oxygen at the vertices of a square lattice and one hydrogen atom between each pair of adjacent oxygen atoms. The hydrogen atoms must stick out on the left and right sides but are not allowed to stick out the top or bottom. One 5×5 example is shown below.

```
H-O H-O-H O-H O-H O-H
  |         |   |   |
  H   H   H   H   H
      |
H-O-H O H-O H-O-H O-H
      |   |         |
  H   H   H   H   H
  |           |
H-O H-O-H O-H O H-O-H
          |   |
  H   H   H   H   H
  |   |           |
H-O H-O H-O H-O-H O-H
          |
  H   H   H   H   H
  |   |       |   |
H-O H-O H-O-H O-H O-H
```

Note that each hydrogen atom is attached to exactly one of its neighboring oxygen atoms and each oxygen atom is attached to two of its neighboring hydrogen atoms. (Recall that one water molecule is a unit of one `O` linked to two `H`'s.)

It turns out we can encode a square ice pattern with what is known as an *alternating sign matrix* (ASM): horizontal molecules are encoded as 1, vertical molecules are encoded as -1 and all other molecules are encoded as 0. So, the above pattern would be encoded as:

$$
\begin{array}{ccccc}
0 & 1 & 0 & 0 & 0 \\
1 & -1 & 0 & 1 & 0 \\
0 & 1 & 0 & -1 & 1 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0
\end{array}
$$

An ASM is a square matrix with entries 0, 1 and -1, where the sum of each row and column is 1 and the non-zero entries in each row and in each column must alternate in sign. (It turns out there is a one-to-one correspondence between ASM's and square ice patterns!)

Your job is to display the square ice pattern, in the same format as the example above, for a given ASM. Use dashes (–) for horizontal attachments and vertical bars (|) for vertical attachments. The pattern should be surrounded with a border of asterisks (*), be left justified and there should be exactly one character between neighboring hydrogen atoms (H) and oxygen atoms (O): either a space, a dash or a vertical bar.

Input

Input consists of multiple cases. Each case consists of a positive integer m (≤ 11) on a line followed by m lines giving the entries of an ASM. Each line gives a row of the ASM with entries separated by a single space. The end of input is indicated by a line containing $m = 0$.

Output

For each case, print the case number (starting from 1), in the format shown in the Sample Output, followed by a blank line, followed by the corresponding square ice pattern in the format described above. Separate the output of different cases by a blank line.

Sample Input

```
2
0 1
1 0
4
0 1 0 0
1 -1 0 1
0 0 1 0
0 1 0 0
0
```

Sample Output

```
Case 1:

***********
*H-O H-O-H*
*  |      *
*  H   H  *
*      |  *
*H-O-H O-H*
***********

Case 2:

*******************
*H-O H-O-H O-H O-H*
*  |       |   |  *
*  H   H   H   H  *
*      |          *
*H-O-H O H-O H-O-H*
*      |   |      *
*  H   H   H   H  *
*  |           |  *
*H-O H-O H-O-H O-H*
*      |          *
*  H   H   H   H  *
*  |       |   |  *
*H-O H-O-H O-H O-H*
*******************
```