ACM International Collegiate Programming Contest
2005 East Central Regional Contest
Ashland University
University of Michigan – Dearborn
Sheridan Institute of Technology
University of Cincinnati
November 5, 2005

Sponsored by IBM

Rules:

1. There are **eight** questions to be completed in **five hours**.
2. 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.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contest officials (e.g., that might generate a security violation).
6. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
7. All communication with the judges will be handled by the PC^2 environment.
8. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A: ACM (ACronym Maker)

The sadists who design problems for ACM programming contests often like to include the abbreviation "ACM" somewhere in their problem descriptions. Thus, in years past, the World Finals has had problems involving "Apartment Construction Management," the "Atheneum of Culture and Movies," the "Association of Cover Manufacturers," "ACM Airlines," the "Association for Computational Marinelife," and even an insect named "Amelia Cheese Mite." However, the number of word combinations beginning with A, C, and M that make sense is finite and the problem writers are starting to run out of ideas (it's hard to think of problems about "Antidisestablishmentarianistic Chthonian Metalinguistics"). Fortunately, modern culture allows more flexibility in designing abbreviations — consider, for example:

GDB — Gnu DeBugger

LINUX — either "LINus's UniX" or "LINUs's miniX" or "Linux Is Not UniX"

SNOBOL — StriNg Oriented symBOlic Language

SPITBOL — SPeedy ImplemenTation of snoBOL

The rules used in these examples seem to be:

- Insignificant words (such as "of", "a", "the", etc.) are ignored.
- The letters of the abbreviation must appear, in the correct order, as an ordered sublist of the letters in the significant words of the phrase to be abbreviated.
- At least one letter of the abbreviation must come from every significant word (multiple occurrences of a letter are, of course, treated as distinct).

Of course these rules are often broken in real life. For instance, RADAR is an abbreviation for "RAdio Detecting And Ranging". Under our rules (assuming that "and" is an insignificant word), this would not be a valid abbreviation (however, RADR or RADRAN or DODGING would be valid). You have been asked to take a list of insignificant words and a list of abbreviations and phrases and to determine in how many ways each abbreviation can be formed from the corresponding phrase according to the rules above.

Input

The input file consists of multiple scenarios. Each scenario begins with an integer $100 \geq n \geq 1$ followed by n insignificant words, all in lower case, one per line with no extra white space. (A line containing 0 indicates end of input.) Following this are one or more test cases for this scenario, one per line, followed by a line containing the phrase "LAST CASE". Each line containing a test case begins with an abbreviation (uppercase letters only) followed by a phrase (lowercase letters and spaces only). The abbreviation has length at least 1 and the phrase contains at least one significant word. No input line (including abbreviation, phrase, and spaces) will contain more than 150 characters. Within these limits, however, abbreviations and phrase words may be any length.

Output

For each test case, output the abbreviation followed by either

`is not a valid abbreviation`

or

`can be formed in` i `ways`

where i is the number of different ways in which the letters of the abbreviation may be assigned to the letters in the phrase according to the rules above. The value of i will not exceed the range of a 32-bit signed integer.

Sample Input

```
2
and
of
ACM academy of computer makers
RADAR radio detection and ranging
LAST CASE
2
a
an
APPLY an apple a day
LAST CASE
0
```

Sample Output

```
ACM can be formed in 2 ways
RADAR is not a valid abbreviation
APPLY can be formed in 1 ways
```

Problem B: Countdown

Ann Sister owns a genealogical database service, which maintains family tree history for her clients. When clients login to the system, they are presented with a variety of services: searching, printing, querying, etc. One recent question that came up which the system was not quite prepared for was the following: "Which member of my family had the most grandchildren?" The client who posed this question eventually had to answer it by manually searching the family tree database herself. Ann decided to have software written in case this question (or ones similar to it asking for great-grandchildren, or great-great-grandchildren, etc.) is asked in the future.

Input

Input will consist of multiple test cases. The first line of the input will contain a single integer indicating the number of test cases. Each test case starts with a single line containing two positive integers n and d, where n indicates the number of lines to follow containing information about the family tree, and d indicates the specific question being asked about the tree: if $d = 1$, then we are interested in persons with the most children (1 generation away); if $d = 2$, then we are interested in persons with the most grandchildren (2 generations away), and so on. The next n lines are of the form

$name\ m\ dname_1\ dname_2\ \ldots\ dname_m$

where *name* is one of the family members' names, m is the number of his/her children, and $dname_1$ through $dname_m$ are the names of the children. These lines will be given in no particular order. You may assume that all n lines describe one single, connected tree. There will be no more than 1000 people in any one tree, and all names will be at most 10 characters long.

Output

For each test case, output the three names with the largest number of specified descendants in order of number of descendants. If there are ties, output the names within the tie in alphabetical order. Print fewer than three names if there are fewer than three people who match the problem criteria (you should not print anyone's name who has 0 of the specified descendants), and print more than three if there is a tie near the bottom of the list. Print each name one per line, followed by a single space and then the number of specified descendants. The output for each test case should start with the line

`Tree` i`:`

where i is the test case number (starting at 1). Separate the output for each problem with a blank line.

Sample Input

```
3
8 2
Barney 2 Fred Ginger
Ingrid 1 Nolan
Cindy 1 Hal
Jeff 2 Oliva Peter
Don 2 Ingrid Jeff
Fred 1 Kathy
Andrea 4 Barney Cindy Don Eloise
Hal 2 Lionel Mary
6 1
Phillip 5 Jim Phil Jane Joe Paul
Jim 1 Jimmy
Phil 1 Philly
Jane 1 Janey
Joe 1 Joey
Paul 1 Pauly
6 2
Phillip 5 Jim Phil Jane Joe Paul
Jim 1 Jimmy
Phil 1 Philly
Jane 1 Janey
Joe 1 Joey
Paul 1 Pauly
```

Sample Output

```
Tree 1:
Andrea 5
Don 3
Cindy 2

Tree 2:
Phillip 5
Jane 1
Jim 1
Joe 1
Paul 1
Phil 1

Tree 3:
Phillip 5
```

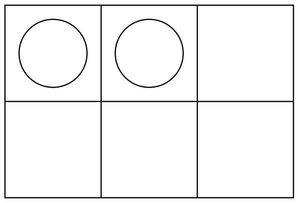
Problem C: The Game of Efil

Almost anyone who has ever taken a class in computer science is familiar with the "Game of Life," John Conway's cellular automata with extremely simple rules of birth, survival, and death that can give rise to astonishing complexity.

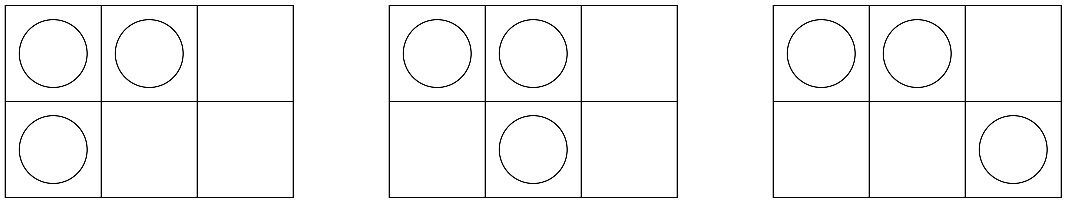
The game is played on a rectangular field of cells, each of which has eight neighbors (adjacent cells). A cell is either occupied or not. The rules for deriving a generation from the previous one are:

- If an occupied cell has 0, 1, 4, 5, 6, 7, or 8 occupied neighbors, the organism dies (0, 1: of loneliness; 4 thru 8: of overcrowding).
- If an occupied cell has two or three occupied neighbors, the organism survives to the next generation.
- If an unoccupied cell has three occupied neighbors, it becomes occupied (a birth occurs).

One of the major problems researchers have looked at over the years is the existence of so-called "Garden of Eden" configurations in the Game of Life — configurations that could not have arisen as the result of the application of the rules to some previous configuration. We're going to extend this question, which we'll call the "Game of Efil": Given a starting configuration, how many possible parent configurations could it have? To make matters easier, we assume a finite grid in which edge and corner cells "wrap around" (i.e., a toroidal surface). For instance, the 2 by 3 configuration:



has exactly three possible parent configurations; they are:



You should note that when counting neighbors of a cell, another cell may be counted as a neighbor more than once, if it touches the given cell on more than one side due to the wrap around. This is the case for the configurations above.

Input

There will be multiple test cases. Each case will start with a line containing a pair of positive integers m and n, indicating the number of rows and columns of the configuration, respectively. The next line will contain a nonnegative integer k indicating the number of "live" cells in the configuration. The following k lines each contain the row and column number of one live cell, where row and column numbering both start at zero. The final test case is followed by a line where $m = n = 0$ — this line should not be processed. You may assume that the product of m and n is no more than 16.

Output

For each test case you should print one line of output containing the case number and the number of possible ancestors. Imitate the sample output below. Note that if there are 0 ancestors, you should print out

```
Garden of Eden.
```

Sample Input

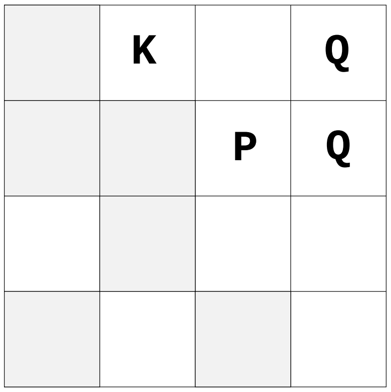
```
2 3
2
0 0
0 1
3 3
4
0 0
0 1
0 2
1 1
3 3
5
0 0
1 0
1 2
2 1
2 2
0 0
```

Sample Output

```
Case 1: 3 possible ancestors.
Case 2: 1 possible ancestors.
Case 3: Garden of Eden.
```

Problem D: Queens, Knights and Pawns

You all are familiar with the famous 8-queens problem which asks you to place 8 queens on a chess board so no two attack each other. In this problem, you will be given locations of queens and knights and pawns and asked to find how many of the unoccupied squares on the board are not under attack from either a queen or a knight (or both). We'll call such squares "safe" squares. Here, pawns will only serve as blockers and have no capturing ability. The board below has 6 safe squares. (The shaded squares are safe.)



Recall that a knight moves to any unoccupied square that is on the opposite corner of a 2x3 rectangle from its current position; a queen moves to any square that is visible in any of the eight horizontal, vertical, and diagonal directions from the current position. Note that the movement of a queen can be blocked by another piece, while a knight's movement can not.

Input

There will be multiple test cases. Each test case will consist of 4 lines. The first line will contain two integers n and m, indicating the dimensions of the board, giving rows and columns, respectively. Neither integer will exceed 1000. The next three lines will each be of the form

$k\ r_1\ c_1\ r_2\ c_2 \cdots r_k\ c_k$

indicating the location of the queens, knights and pawns, respectively. The numbering of the rows and columns will start at one. There will be no more than 100 of any one piece. Values of $n = m = 0$ indicate end of input.

Output

Each test case should generate one line of the form

`Board` b `has` s `safe squares.`

where b is the number of the board (starting at one) and you supply the correct value for s.

Sample Input

```
4 4
2 1 4 2 4
1 1 2
1 2 3
2 3
1 1 2
1 1 1
0
1000 1000
1 3 3
0
0
0 0
```

Sample Output

```
Board 1 has 6 safe squares.
Board 2 has 0 safe squares.
Board 3 has 996998 safe squares.
```

Problem E: Reliable Nets

You're in charge of designing a campus network between buildings and are very worried about its reliability and its cost. So, you've decided to build some redundancy into your network while keeping it as inexpensive as possible. Specifically, you want to build the cheapest network so that if any one line is broken, all buildings can still communicate. We'll call this a *minimal reliable net*.

Input

There will be multiple test cases for this problem. Each test case will start with a pair of integers n (≤ 15) and m (≤ 20) on a line indicating the number of buildings (numbered 1 through n) and the number of potential inter-building connections, respectively. (Values of $n = m = 0$ indicate the end of the problem.) The following m lines are of the form b_1 b_2 c (all positive integers) indicating that it costs c to connect building b_1 and b_2. All connections are bidirectional.

Output

For each test case you should print one line giving the cost of a minimal reliable net. If there is a minimal reliable net, the output line should be of the form:

`The minimal cost for test case` p `is` c`.`

where p is the number of the test case (starting at 1) and c is the cost. If there is no reliable net possible, output a line of the form:

`There is no reliable net possible for test case` p`.`

Sample Input

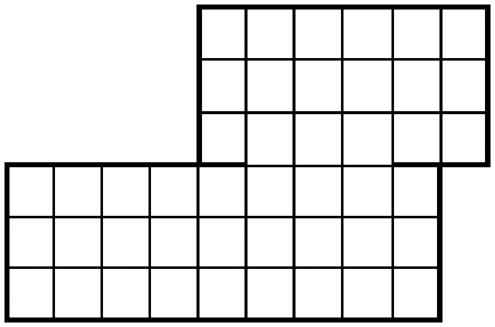
```
4 5
1 2 1
1 3 2
2 4 2
3 4 1
2 3 1
2 1
1 2 5
0 0
```

Sample Output

```
The minimal cost for test case 1 is 6.
There is no reliable net possible for test case 2.
```

Problem F: Square Count

Little Bobby Roberts, age 8, has been dragged to yet another museum by his parents. While they while away the hours studying Etruscan pottery and Warhol soup cans, Bobby must depend on himself for entertainment. Having a mathematical bent, he recently started counting all the square tiles on the floors of the museum. He soon realized that the tiles could be grouped into larger squares that needed to be added to the count. The problem became a bit more complicated when he started counting squares contained in multiple rooms, since some squares overlapped both rooms. For example, the two rooms shown below contain a total of 86 squares: 45 1×1 squares, 28 2×2 squares and 13 3×3 squares. (Note the opening between the two rooms is only 3 squares wide.)



While this helped kill several days' worth of museum visits, it soon became rather tedious, so Bobby is now looking for a program to automate the counting process for him.

Input

Input will consist of multiple test cases. The first line of each case will be a positive integer $n \leq 1000$ which will indicate the number of rooms in the museum. After this will be n lines, each containing a description of one room. Each room will be rectangular in shape and will be described by a line of the form

x_1 y_1 x_2 y_2

where (x_1, y_1) and (x_2, y_2) are opposing corner coordinates (integers) of the room. No two rooms will overlap, though they may share a side. If the shared side is of length $m > 2$, then a door of length $m - 2$ exists between the two rooms, centered along the shared length. No square of any size will overlap more than two rooms. All x and y values will be $\leq 1,000,000$. An input line of $n = 0$ terminates input and should not be processed.

Output

For each test case, output the total number of squares on a single line in the format shown below. All answers will fit within a 32-bit integer and cases are enumerated starting at 1.

Sample Input

```
2
0 0 9 3
10 6 4 3
3
11 20 15 24
11 17 15 20
15 16 20 24
0
```

Sample Output

```
Case 1: 86
Case 2: 152
```

Problem G: Swamp Things

Hugh F. Oh, in his never-ending quest to prove the existence of extraterrestrials, has gotten hold of a number of nighttime photographs taken by a research group that is examining glowing swamp gas. Hugh wants to see if any of the photos show, not swamp gas, but Little Grey Men in glowing suits. The photographs consist of bright dots appearing against a black background. Unfortunately, at the time the photos were taken, trains were travelling through the area (there is a train trestle over the swamp), and occasional lights from the train windows also appear in the photographs. Hugh, being a fastidious researcher, wants to eliminate these spots from the images. He can't tell from the photos exactly where the tracks are, or from what direction the photos were taken, but he knows that the tracks in that area are perfectly straight, so he's decided on the following approach: he will find the line with the maximum number of spots lying on it and, if there are four or more spots on the line, he will eliminate those points from his calculations, assuming that those are windows on the train. If two or more lines have the maximum number of points, Hugh will just randomly select one such set and delete it from the photo (he's not all that fastidious – after all, he believes in Little Grey Men). If there are fewer than four points lying along a common line, Hugh will assume that there is no train in the photograph and won't delete any points. Please write a program for him to process a set of photographs.

Input

There will be a series of test cases. Each test case is one photograph described by a line containing a positive integer n ($\leq$ 1000), the number of distinct spots in the photograph, followed by n lines containing the integer coordinates of the spots, one (x, y) pair per line. All coordinates are between 0 and 10000. The last photo description is followed by a line containing a zero, marking the end of the input. This line should not be processed.

Output

For each test case, output the photo number followed by the number of points eliminated from the photograph. Imitate the sample output below.

Sample Input

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

Sample Output

```
Photo 1: 4 points eliminated
Photo 2: 0 points eliminated
```

Problem H: Two Ends

In the two-player game “Two Ends”, an even number of cards is laid out in a row. On each card, face up, is written a positive integer. Players take turns removing a card from either end of the row and placing the card in their pile. The player whose cards add up to the highest number wins the game. Now one strategy is to simply pick the card at the end that is the largest — we’ll call this the greedy strategy. However, this is not always optimal, as the following example shows: (The first player would win if she would first pick the 3 instead of the 4.)

3 2 10 4

You are to determine exactly how bad the greedy strategy is for different games when the second player uses it but the first player is free to use any strategy she wishes.

Input

There will be multiple test cases. Each test case will be contained on one line. Each line will start with an even integer n followed by n positive integers. A value of $n = 0$ indicates end of input. You may assume that n is no more than 1000. Furthermore, you may assume that the sum of the numbers in the list does not exceed 1,000,000.

Output

For each test case you should print one line of output of the form:

`In game` m`, the greedy strategy might lose by as many as` p `points.`

where m is the number of the game (starting at game 1) and p is the maximum possible difference between the first player’s score and second player’s score when the second player uses the greedy strategy. When employing the greedy strategy, always take the larger end. If there is a tie, remove the left end.

Sample Input

```
4 3 2 10 4
8 1 2 3 4 5 6 7 8
8 2 2 1 5 3 8 7 3
0
```

Sample Output

```
In game 1, the greedy strategy might lose by as many as 7 points.
In game 2, the greedy strategy might lose by as many as 4 points.
In game 3, the greedy strategy might lose by as many as 5 points.
```