ACM International Collegiate Programming Contest
2002 East Central Regional Contest

Ashland University
University of Cincinnati
Western Michigan University
Sheridan University
November 9, 2002

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: Ball Toss

Classmates stand in a circle facing inward, each with the direction *left* or *right* in mind. One of the students has a ball and begins by tossing it to another student. (It doesn't really matter which one.) When one catches the ball and is thinking *left*, she throws it back across the circle one place to the left (from her perspective) of the person who threw her the ball. Then she switches from thinking *left* to thinking *right*. Similarly, if she is thinking *right*, she throws the ball to the right of the person who threw it to her and then switches from thinking *right* to thinking *left*.

There are two exceptions to this rule: If one catches the ball from the classmate to her immediate left and is also thinking *left*, she passes the ball to the classmate to her immediate right, and then switches to thinking *right*. Similarly, if she gets the ball from the classmate to her immediate right and is thinking *right*, she passes the ball to the classmate to her immediate left, and then switches to thinking *left*. (Note that these rules are given to avoid the problem of tossing the ball to oneself.)

No matter what the initial pattern of left and right thinking is and who first gets tossed the ball, everyone will get tossed the ball eventually! In this problem, you will figure out how long it takes.

You'll be given the initial directions of n classmates (numbered clockwise), and the classmate to whom classmate 1 initially tosses the ball. (Classmate 1 will always have the ball initially.)

Input

There will be multiple problem instances. Each problem instance will be of the form

$n\ k\ t_1\ t_2\ t_3\ \dots\ t_n$

where n ($2 \le n \le 30$) is the number of classmates, numbered 1 through n clockwise around the circle, k (> 1) is the classmate to whom classmate 1 initially tosses the ball, and t_i ($i = 1, 2, \dots, n$) are each either `L` or `R`, indicating the initial direction thought by classmate i. ($n = 0$ indicates end of input.)

Output

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

`Classmate` m `got the ball last after` t `tosses.`

where m and t are for you to determine. You may assume that t will be no larger than 100,000.

Note that classmate number 1 initially has the ball and tosses it to classmate k. Thus, number 1 has not yet been tossed the ball and so does not switch the direction he is thinking.

Sample Input

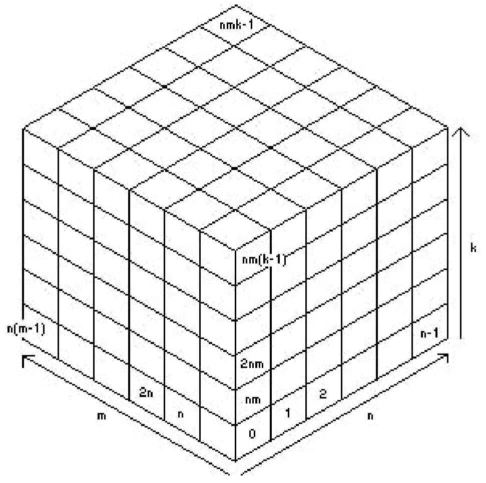
```
4 2 L L L L
4 3 R L L R
10 4 R R L R L L R R L R
0
```

Sample Output

```
Classmate 3 got the ball last after 4 tosses.
Classmate 2 got the ball last after 4 tosses.
Classmate 9 got the ball last after 69 tosses.
```

Problem B: Galactic Breakup

After ruling a large chunk of the Milky Way for millennia, the Cosmic OBsolescent OLigarchy is finally breaking up into a collection of independent monarchies. COBOL is a very organized empire and takes the shape of a gigantic cube with dimensions n by m by k parsecs. (COBOL is also very secretive, so only a few know the exact values of n, m and k.) To facilitate the control of the empire it is partitioned into nmk smaller dominions, each 1 cubic parsec in size. These dominions are numbered as follows:



Each independent monarchy is a connected collection of one or more dominions (a dominion is connected to another if they share a face) and over a period of several imperial months, one monarchy per month will secede from the empire. Each secession begins at the first day of the month. One concern of COBOL is that during the breakup, various parts of the remaining empire may become disconnected from one another, which could hamper the administration of what's left of the empire. Your job is to determine the number of months of the breakup during which the empire is disconnected.

Input

Input will consist of multiple problem instances. The first line will contain a positive integer indicating the number of problem instances to follow. The first line of each problem instance will contain four integers: $n\ m\ k\ l$, where n, m and k are as described above, with $1 \le n, m, k \le 30$, and l is the number of independent monarchies which the empire is being divided into. Following this will be l lines defining the monarchies. Each will have the form $p\ d_1\ d_2\ d_3\ \ldots\ d_p$, where p is the number of dominions making up the monarchy ($1 \le p \le 20$), and $d_1, \ldots, d_p$ are the dominions. The monarchies are listed in the order in which they will secede from the empire.

Output

Output for each problem instance should consist of a single integer on a line, indicating the number of months which the empire was disconnected.

Sample Input

```
2
2 2 3 9
2 4 5
3 6 8 10
1 7
1 2
1 11
1 9
1 1
1 0
1 3
2 2 3 3
4 0 1 2 3
4 4 5 6 7
4 8 9 10 11
```

Sample Output

```
4
0
```

Problem C: Increasing Sequences

Given a string of digits, insert commas to create a sequence of strictly increasing numbers so as to minimize the magnitude of the last number. For this problem, leading zeros are allowed in front of a number.

Input

Input will consist of multiple test cases. Each case will consist of one line, containing a string of digits of maximum length 80. A line consisting of a single 0 terminates input.

Output

For each instance, output the comma separated strictly increasing sequence, with no spaces between commas or numbers. If there are several such sequences, pick the one which has the largest first value; if there's a tie, the largest second number, etc.

Sample Input

```
3456
3546
3526
0001
100000101
0
```

Sample Output

```
3,4,5,6
35,46
3,5,26
0001
100,000101
```

Problem D: Pre-Post-erous!

We are all familiar with pre-order, in-order and post-order traversals of binary trees. A common problem in data structure classes is to find the pre-order traversal of a binary tree when given the in-order and post-order traversals. Alternatively, you can find the post-order traversal when given the in-order and pre-order. However, in general you cannot determine the in-order traversal of a tree when given its pre-order and post-order traversals. Consider the four binary trees below:

```
    a     a    a     a
   /     /      \     \
  b     b        b     b
 /       \      /       \
c         c    c         c
```

All of these trees have the same pre-order and post-order traversals. This phenomenon is not restricted to binary trees, but holds for general m-ary trees as well.

Input

Input will consist of multiple problem instances. Each instance will consist of a line of the form

m $s1$ $s2$

indicating that the trees are m-ary trees, $s1$ is the pre-order traversal and $s2$ is the post-order traversal. All traversal strings will consist of lowercase alphabetic characters. For all input instances, $1 \leq m \leq 20$ and the length of $s1$ and $s2$ will be between 1 and 26 inclusive. If the length of $s1$ is k (which is the same as the length of $s2$, of course), the first k letters of the alphabet will be used in the strings. An input line of 0 will terminate the input.

Output

For each problem instance, you should output one line containing the number of possible trees which would result in the pre-order and post-order traversals for the instance. All output values will be within the range of a 32-bit signed integer. For each problem instance, you are guaranteed that there is at least one tree with the given pre-order and post-order traversals.

Sample Input

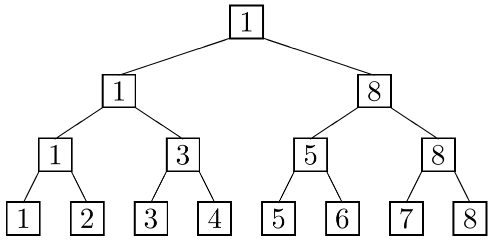
```
2 abc cba
2 abc bca
10 abc bca
13 abejkcfghid jkebfghicda
0
```

Sample Output

```
4
1
45
207352860
```

Problem E: Knockout Tournament

In a knockout tournament there are 2^n players. One loss and a player is out of the tournament. Winners then play each other with the new winners advancing until there is only one winner left. If we number the players $1, 2, 3, \ldots, 2^n$, with the first round pairings $2k-1$ vs $2k$, for $k = 1, 2, \ldots, 2^{n-1}$, then we could give the results of the tournament in a complete binary tree. The winners are indicated in the interior nodes of the tree. Below is an example of a tournament with $n = 3$.



After the tournament, some reporters were arguing about the relative ranking of the players, as determined by the tournament results. It's assumed that if player A beats player B who in turn beats player C, that player A will also beat player C; that is, winning is transitive. Now there is no doubt who the best player is. The question is what is the highest ranking a player can reasonably claim as a result of the tournament and what is the worst ranking a player can have, as a result of the tournament? For example, in the above tournament player 2, having lost to the eventual winner, could claim to be the 2nd best player in the field, but could well be the worst (ranked 8th). Player 5 could claim to be as high as 3rd (having lost to someone who could be 2nd) but no worse than 7th (having beaten one player in the 1st round).

You are to determine the highest and lowest possible rankings of a set of players in the field, given the results of the tournament.

Input

There will be multiple input instances. The input for each instance consists of three lines. The first line will contain a positive integer $n < 8$, indicating there are 2^n players in the tournament, numbered 1 through 2^n, paired in the manner indicated above. A value of $n = 0$ indicates end of input. The next line will contain the results of each round of the tournament (listed left-to-right) starting with the 1st round. For example, the tournament above would be given by

```
1 3 5 8 1 8 1
```

The final line of input for each instance will be a positive integer m followed by integers $k_1, ..., k_m$, where each k_i is a player in the field.

Output

For each k_i, issue one line of output of the form:

`Player` k_i `can be ranked as high as` h `or as low as` l`.`

where you supply the appropriate numbers. These lines should appear in the same order as the k_i did in the input. Output for problem instances should be separated with a blank line.

Sample Input

```
3
1 3 5 8 1 8 1
2 2 5
4
2 3 6 7 9 11 14 15 3 6 9 15 6 9 6
4 1 15 7 6
0
```

Sample Output

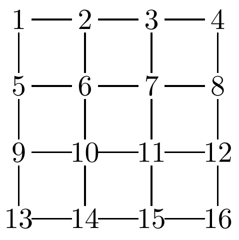
```
Player 2 can be ranked as high as 2 or as low as 8.
Player 5 can be ranked as high as 3 or as low as 7.

Player 1 can be ranked as high as 4 or as low as 16.
Player 15 can be ranked as high as 3 or as low as 13.
Player 7 can be ranked as high as 2 or as low as 15.
Player 6 can be ranked as high as 1 or as low as 1.
```

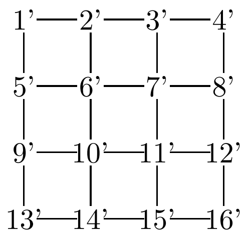
Problem F: Plugged In

The designers of the new NentindoBoxStation game system want to provide interactive input from many different sources. Using a special sensor-lined "electronic cocoon" users should be able to do things such as control simulated laser cannons by moving their eyebrows, accelerate/decelerate by wiggling their ears, steer in three-dimensional space by rotating their ankles ... the possibilities are endless.

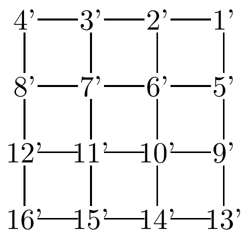
The connections between the sensors in the cocoon and the simulated actions in the computer are to be made using a special square plug with n^2 pins (the value of n has not yet been determined). Each pin can carry the output from one sensor, although for some applications not all pins will be active. The plug fits into a square socket containing n^2 holes that is attached to the various game inputs; again, for some games, not all input holes will be used. The socket can be flipped over and rotated to achieve different matchings between sensor pins and game inputs. Pins and socket holes are numbered consecutively in row major order (as shown below for the value $n = 4$).



Plug pins



Socket holes (front of plug)



Socket holes (back of plug)

Clearly pin 1 can be connected only to holes 1', 4', 13', or 16' (depending on how the plug is rotated with respect to the socket). Pin 2 can be connected only to holes 2', 3', 5', 8', 9', 12', 14', or 15' (if we consider all rotations and connections in both the back and front of the plug).

Most games require extra wiring to achieve connections because there is no way to match pins directly to their corresponding sockets (for instance, connecting pin 1 to hole 11' in the figure). This wiring will be achieved with a special game-specific "wiring block" that will be placed between the plug and the socket. The lengths of these wires will depend on the orientation of the socket with respect to the plug. Given a list of connections that must be made, you are going to help the designers determine the *minimum average wire length* that is needed for the connections in the wiring block. Wires always run parallel to the grid lines, so the amount of wire between a pin in the plug and a hole in the socket is 1 plus the length of a shortest grid path between the nodes (the extra "1" is due to the thickness of the wiring block itself). Thus, one unit of wire is the minimum required (when a pin is positioned directly over the hole it is supposed to connect to).

For instance, if we are given the set (1,3'), (5,7'), (2,6') for the plug and socket above, the average distance for this set of pairs is 2.6667 if we put the plug into the front of the socket without rotating the socket, but is only 2.3333 if we rotate the socket 180 degrees and then flip it horizontally, placing the plug in the back of the socket.

Input

Input will consist of a set of scenarios. Each scenario consists of a positive integer n, the side length of the plug and socket (less than or equal to 100) on a line by itself, followed by a positive integer m (less than or equal to n^2) on a line by itself, followed by m lines, each containing a pair of positive integers in the range $1, \ldots, n^2$. You may assume that no two pairs will have either a common first element or a common second element. The first integer represents a pin position in the plug, the second is a hole position in the socket. The final scenario is followed by 0 on a line by itself.

Output

For each scenario, output the scenario number (starting with 1), followed by the smallest average distance achievable between the m pin/socket pairs after rotations and reflections are considered (assuming an appropriate routing box is used), in a line of the form:

`Scenerio` n`: smallest average =` *avg*

where *avg* is the average is rounded, and displayed, to four decimal places. Separate lines of output by a single blank line.

Sample Input

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

Sample Output

```
Scenario 1: smallest average = 2.3333

Scenario 2: smallest average = 1.0000
```

Problem G: One Person "The Price is Right"

In the game show "The Price is Right", a number of players (typically 4) compete to get on stage by guessing the price of an item. The winner is the person whose guess is the closest one not exceeding the actual price. Because of the popularity of the one-person game show "Who Wants to be a Millionaire", the American Contest Management (ACM) would like to introduce a one-person version of the "The Price is Right". In this version, each contestant is allowed G ($1 \leq G \leq 30$) guesses and L ($0 \leq L \leq 30$) lifelines. The contestant makes a number of guesses for the actual price. After each guess, the contestant is told whether it is correct, too low, or too high. If the guess is correct, the contestant wins. Otherwise, he uses up a guess. Additionally, if his guess is too high, a lifeline is also lost. The contestant loses when all his guesses are used up or if his guess is too high and he has no lifelines left. All prices are positive integers.

It turns out that for a particular pair of values for G and L, it is possible to obtain a guessing strategy such that if the price is between 1 and N (inclusive) for some N, then the player can guarantee a win. The ACM does not want every contestant to win, so it must ensure that the actual price exceeds N. At the same time, it does not want the game to be too difficult or there will not be enough winners to attract audience. Thus, it wishes to adjust the values of G and L depending on the actual price. To help them decide the correct values of G and L, the ACM has asked you to solve the following problem. Given G and L, what is the largest value of N such that there is a strategy to win as long as the price is between 1 and N (inclusive)?

Input

The input consists of a number of cases. Each case is specified by one line containing two integers G and L, separated by one space. The end of input is specified by a line in which $G = L = 0$.

Output

For each case, print a line of the form:

`Case` c`:` N

where c is the case number (starting from 1) and N is the number computed.

Sample Input

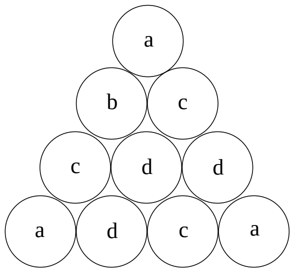
```
3 0
3 1
10 5
7 7
0 0
```

Sample Output

```
Case 1: 3
Case 2: 6
Case 3: 847
Case 4: 127
```

Problem H: Slots of Fun

The International Betting Machine company has just issued a new type of slot machine. The machine display consists of a set of identical circles placed in a triangular shape. An example with four rows is shown below. When the player pulls the lever, the machine places a random letter in the center of each circle. The machine pays off whenever any set of identical letters form the vertices of an equilateral triangle. In the example below, the letters 'a' and 'c' satisfy this condition.



In order to prevent too many payoffs, the electronics in the machine ensures that no more than 3 of any letter will appear in any display configuration.

IBM is manufacturing several models of this machine, with varying number of rows in the display, and they are having trouble writing code to identify winning configurations. Your job is to write that code.

Input

Input will consist of multiple problem instances. Each instance will start with an integer n indicating the number of rows in the display. The next line will contain $n(n+1)/2$ letters of the alphabet (all lowercase) which are to be stored in the display row-wise, starting from the top. For example, the display above would be specified as

```
4
abccddadca
```

The value of n will be between 1 and 12, inclusive. A line with a single 0 will terminate input.

Output

For each problem instance, output all letters which form equilateral triangles on a single line, in alphabetical order. If no such letters exist, output "LOOOOOOOOSER!".

Sample Input

```
4
abccddadca
6
azdefccrhijrrmznzocpq
2
abc
0
```

Sample Output

```
ac
crz
LOOOOOOOOSER!
```