

**UM Team Practice Programming Contest  
September 27, 2011**

**Sponsored by:  
University of Michigan**

Rules:

1. There are a five to eight questions to be solved at any time during the contest.
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 contestant 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 PC2 environment.
8. Judges' decisions are to be considered final. No cheating will be tolerated.

## Problem A: Crypto Columns

The columnar encryption scheme scrambles the letters in a message (or *plaintext*) using a keyword as illustrated in the following example: Suppose **BATBOY** is the keyword and our message is **MEET ME BY THE OLD OAK TREE**. Since the keyword has 6 letters, we write the message (ignoring spacing and punctuation) in a grid with 6 columns, padding with random extra letters as needed:

```
MEETME
BYTHEO
LDOAKT
REENTH
```

Here, we've padded the message with **NTH**. Now the message is printed out by columns, but the columns are printed in the order determined by the letters in the keyword. Since **A** is the letter of the keyword that comes first in the alphabet, column 2 is printed first. The next letter, **B**, occurs twice. In the case of a tie like this we print the columns leftmost first, so we print column 1, then column 4. This continues, printing the remaining columns in order 5, 3 and finally 6. So, the order the columns of the grid are printed would be 2, 1, 4, 5, 3, 6, in this case. This output is called the *ciphertext*, which in this example would be **EYDEMBLRTHANMEKTETOOEEOTH**. Your job will be to recover the plaintext when given the keyword and the ciphertext.

### Input

There will be multiple input sets. Each set will be 2 input lines. The first input line will hold the keyword, which will be no longer than 10 characters and will consist of all uppercase letters. The second line will be the ciphertext, which will be no longer than 100 characters and will consist of all uppercase letters. The keyword **THEEND** indicates end of input, in which case there will be no ciphertext to follow.

### Output

For each input set, output one line that contains the plaintext (with any characters that were added for padding). This line should contain no spacing and should be all uppercase letters.

### Sample Input

```
BATBOY
EYDEMBLRTHANMEKTETOOEEOTH
HUMDING
EIAAHEBXOIFWEHRXONNAALRSUMNREDEXCTLFVEXPEDARTAXNAARYIEX
THEEND
```

### Sample Output

```
MEETMEBYTHEOLDOAKTREENTH
ONCEUPONATIMEINALANDFARFARAWAYTHERELIVEDTHREEBEARSXXXXXX
```

## Problem B: Decorations

The Sultan of Sylvania loves throwing parties, because that gives him a reason to decorate the palace. He particularly likes decorations called *streamers* made up of different beads strung together on a string and hung from the ceiling. Now, like most Sultans, he is very particular about everything, including these strung decorations. Specifically, he only likes certain combinations of beads to be used on the streamers. For example, if there are four different types of beads – A, B, C and D – the Sultan might say “It pleases his highness that only the combinations ABB, BCA, BCD, CAB, CDD and DDA appear in the streamers at tonight’s party”. This, needless to say, puts a severe limit on the number of different streamers possible. For example, if the length of the streamers was 5, then the only possible streams of beads would be BCABB and BCDDA (strings such as ABBCA could not be used because BBC is not an approved combination). Since the Sultan likes variety, it is important to know the total number of streamers possible, given a length and the current bead combinations which tickle the Sultan’s fancy.

### Input

Input will consist of multiple test cases. Each case will consist of two lines. The first line will contain three positive integers  $n$ ,  $l$  and  $m$ , where  $n$  indicates the number of bead types,  $l$  is the length of the streamers and  $m$  indicates the number of bead combinations which the Sultan likes. The maximum values for  $n$ ,  $l$  and  $m$  will be 26, 100 and 600, respectively. The next line will contain the  $m$  combinations. Each combination will be of the same length (between 1 and 10) and will be separated using a single space. All combinations will make use of only the uppercase letters of the alphabet. An input line of 0 0 0 will terminate input and should not be processed.

### Output

For each test case, output a single line indicating the number of possible streamers. All answers will be within the range of a 32-bit integer.

### Sample Input

```
4 5 6
ABB BCA BCD CAB CDD DDA
5 4 5
E D C B A
4 8 3
AA BB CC
0 0 0
```

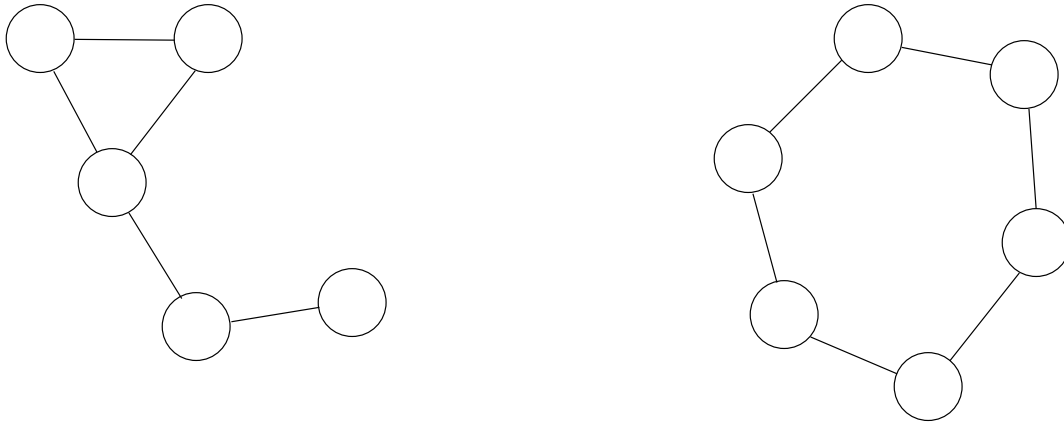
### Sample Output

```
2
625
3
```

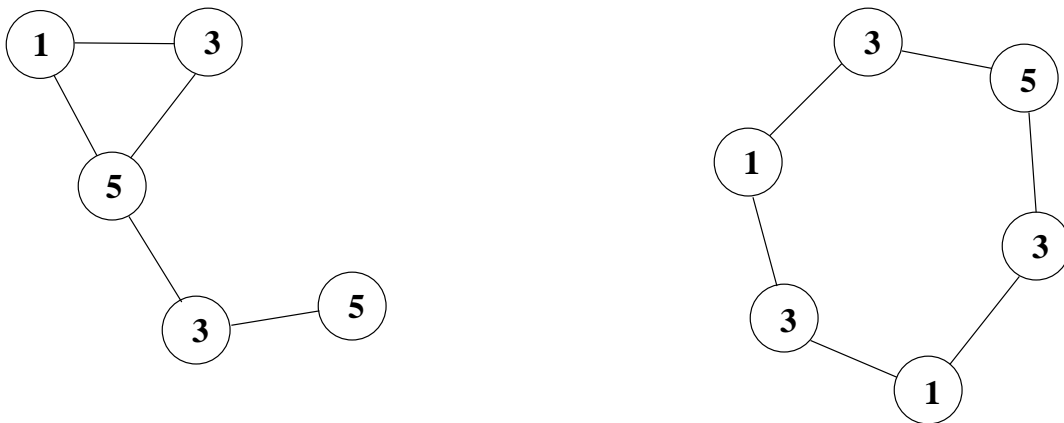
## Problem C: Phone Home

When relay towers for mobile telephones communicate with the mobile phones in their area, there is always the possibility of interference. So, when assigning the transmission frequency, the FCC makes sure that nearby towers have frequencies that aren't too close. On the other hand, the FCC does not want to assign too many different frequencies; they want to save as many as possible for other uses. Your job is to find an optimal assignment of frequencies.

In this problem, the frequencies will be integers. Nearby towers must be assigned frequencies that differ by at least 2. You'll find an assignment using as few frequencies as possible. For example, consider the following two arrangements of towers. Two towers near each other are indicated by the connecting line.



Note that the following are legal frequency assignments to these two tower configurations. However, the second arrangement does not use the fewest number of frequencies possible, since the tower with frequency 5 could have frequency 1.



### Input

There will be multiple test cases. Input for each test case will consist of two lines: the first line will contain the integer  $n$ , indicating the number of towers. The next line will be of the form  $x_1 y_1 x_2 y_2 \dots x_n y_n$  where  $x_i y_i$  are the coordinates of tower  $i$ . A pair of towers are considered "near" each other if the distance between them is no more than 20. There will be no more than 12 towers and no tower will have more than 4 towers near it. A value of  $n = 0$  indicates end of input.

## Output

For each test case, you should print one line in the format:

The towers in case  $n$  can be covered in  $f$  frequencies.

where you determine the value for  $f$ . The case numbers,  $n$ , will start at 1.

## Sample Input

```
5
0 0 5 7.5 1 -3 10.75 -20.1 12.01 -22
6
0 1 19 0 38 1 38 21 19 22 0 21
0
```

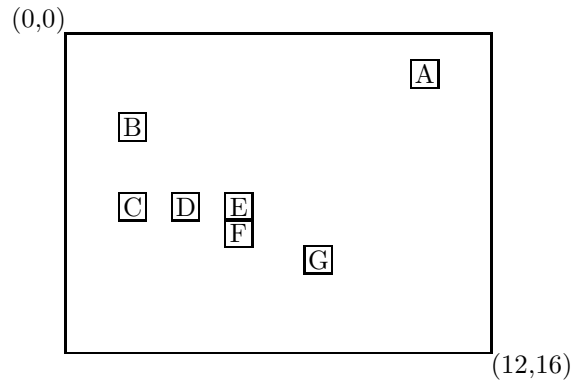
## Sample Output

```
The towers in case 1 can be covered in 3 frequencies.
The towers in case 2 can be covered in 2 frequencies.
```

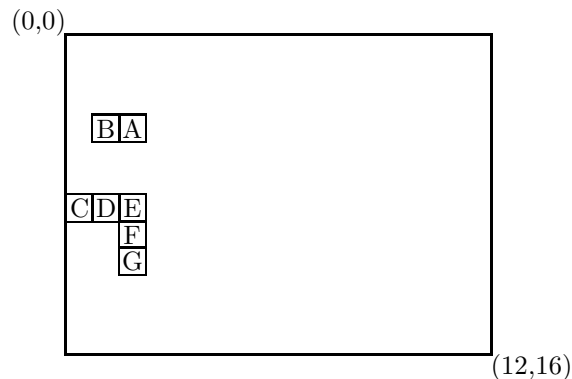
## Problem D: Pushing Boxes

Scrap cars in a junk yard are crushed in a device that pushes the car in from the sides, from the front and back, and from the top and bottom. The result is a compact little chunk of metal. In this problem you're going to model a device that works in a similar manner, but doesn't crush anything, only pushes boxes around in two dimensions. The boxes are all square with unit length on a side and are situated on the floor of a room. Each wall of the room can be programmed to move inward a certain amount, pushing any boxes it may bump into. Unlike the car-crusher, this device is sensitive and if it senses that boxes are stacked up against a wall and that it might crush them if pressed any farther, it will stop.

For example, suppose we have boxes arranged in a 12-by-16 room as shown below. The upper left-hand corners of the boxes (which is how we will locate them in this problem) are at coordinates  $(1,13)$  (box A below),  $(3,2)$ ,  $(6,2)$ ,  $(6,4)$ ,  $(6,6)$ ,  $(7,6)$  and  $(8,9)$  (box G), where the first coordinate indicates distance from the top wall and the second coordinate indicates distance from the left wall.



Suppose the top wall is programmed to move down 3 units (then retreats, as the walls always will) and then the right wall is programmed to move left 14 units. The first operation can be performed with no problem, but the second one can not be carried out without crushing some boxes. Therefore, the right wall will move only 13 units, the maximum distance it can move until boxes are packed tightly between it and the left wall. The boxes will then be in the configuration shown in the following figure. The locations of the boxes are  $(3,1)$ ,  $(3,2)$ ,  $(6,0)$ ,  $(6,1)$ ,  $(6,2)$ ,  $(7,2)$ ,  $(8,2)$ .



## Input

There will be multiple data sets for this problem. The first line of each data set will be two integers giving the height and width of the room. (We'll visualize the room as if on a piece of paper, as drawn above.) Each dimension will be no more than 20. The next line will contain an integer  $n$  ( $0 < n \leq 10$ ) followed by  $n$  pairs of integers, each pair giving the location of a box as the distances from the top and the left walls of the room. The following lines will be of the form **direction**  $m$ , where **direction** is either **down**, **left**, **up**, **right**, or **done** and  $m$  is a positive integer. For example, **left 2** would mean to try to move the right wall 2 spaces to the left. The "direction" **done** indicates that you are finished pushing this set of boxes around. There will be no integer  $m$  following the direction **done**, of course. The data sets are followed by a line containing 0 0.

## Output

For each data set you are to produce one line of output of the form:

Data set  $d$  ends with boxes at locations  $(r_1, c_1)$   $(r_2, c_2)$  ...  $(r_n, c_n)$ .

where the  $(r_i, c_i)$  are the locations of the boxes given from top-to-bottom, left-to-right, (separated by one space) and  $d$  is the data set number (starting at 1).

## Sample Input

```
12 16
7 1 13 3 2 6 2 6 4 6 6 7 6 8 9
down 3
left 14
done
4 4
3 1 0 2 1 2 3
right 3
up 2
left 1
done
0 0
```

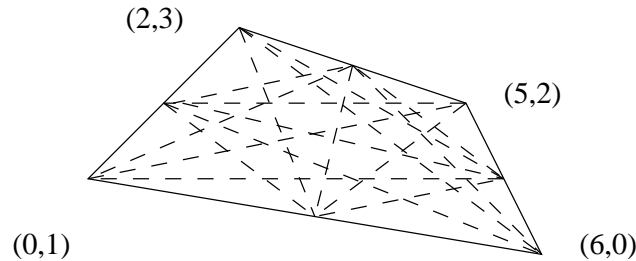
## Sample Output

```
Data set 1 ends with boxes at locations (3,1) (3,2) (6,0) (6,1) (6,2) (7,2) (8,2).
Data set 2 ends with boxes at locations (0,2) (1,1) (1,2).
```

## Problem E: This Takes the Cake

In the kingdom of Polygonia the royal family consists of the king, the queen, and the 10-year-old twins, Prince Obtuse and Prince Trisect. The twins are fiercely competitive, and on their birthday they always vie with each other for the biggest portion of the cake. The wise king and queen have devised the following way to prevent squabbles over the cake. One prince is allowed to cut the cake into two pieces, then the other prince gets to choose which of the two pieces he wants.

Cakes in Polygonia are always in the shape of a convex quadrilateral (a four-sided polygon with each internal angle less than 180 degrees). Furthermore, local custom dictates that all cake cutting must be done using a straight cut that joins two vertices, or two midpoints of the sides of the cake, or a vertex and a midpoint. For instance, the following figure shows all the possible legal cuts in a typical cake.



Your problem is to determine, for a number of different cakes, the best cut, i.e., the one that divides the cake into two pieces whose areas (we are disregarding the thickness of the cake) are as nearly equal as possible. For instance, given a cake whose vertices (when the cake is viewed from above) are located, in counterclockwise order, at the points  $(0, 1)$ ,  $(6, 0)$ ,  $(5, 2)$  and  $(2, 3)$ , the best possible cut would divide the cake into two pieces, one with area 4.375, the other with area 5.125; the cut joins the points  $(1, 2)$  and  $(5.5, 1)$  (the midpoints of two of the sides).

### Input

Input consists of a sequence of test cases, each consisting of four  $(x, y)$  values giving the counterclockwise traversal of the cake's vertices as viewed from directly above the cake; the final test case is followed by a line containing eight zeros. No three points will be collinear, all quadrilaterals are convex, and all coordinates will have absolute values of 10000 or less.

### Output

For each cake, the cake number followed by the two areas, smaller first, to three decimal places of precision.

### Sample Input

```
0 1 6 0 5 2 2 3
0 0 100 0 100 100 0 100
0 0 0 0 0 0 0 0
```

### Sample Output

```
Cake 1: 4.375 5.125
Cake 2: 5000.000 5000.000
```