ACM International Collegiate Programming Contest
2003 East Central Regional Contest
Ashland University
Carnegie Mellon University
Sheridan University
University of Cincinnati
November 8, 2003

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 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 PC^2 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 `EYDEMBLRTHANMEKTETOEEOTH`. 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
EYDEMBLRTHANMEKTETOEEOTH
HUMDING
EIAAHEBXOIFWEHRXONNAALRSUMNREDEXCTLFTVEXPEDARTAXNAARYIEX
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: EKG Sequence

The EKG sequence is a sequence of positive integers generated as follows: The first two numbers of the sequence are 1 and 2. Each successive entry is the smallest positive integer not already used that shares a factor with the preceding term. So, the third entry in the sequence is 4 (being the smallest even number not yet used). The next number is 6 and the next is 3. The first few numbers of this sequence are given below.

1, 2, 4, 6, 3, 9, 12, 8, 10, 5, 15, 18, 14, 7, 21, 24, 16, 20, 22, 11, 33, 27

The sequence gets its name from its rather erratic fluctuations. The sequence has a couple of interesting, but non-trivial, properties. One is that all positive integers will eventually appear in the sequence. Another is that all primes appear in increasing order. Your job here is to find the position in the sequence of a given integer.

Input

Input consists of a number of test cases. Each case will be a line containing a single integer n, $1 \leq n \leq 300000$. An input of 0 follows the last test case. Note that the portion of the EKG sequence that contains all integers $\leq$ 300,000 will not contain an integer >1,000,000.

Output

Each test case should produce one line of output of the form:

`The number` n `appears in location` p.

where n is the number given and p is the position of n in the EKG sequence. You are guaranteed that p will be no larger than 1,000,000.

Sample Input

```
12
21
2
33
100000
299977
0
```

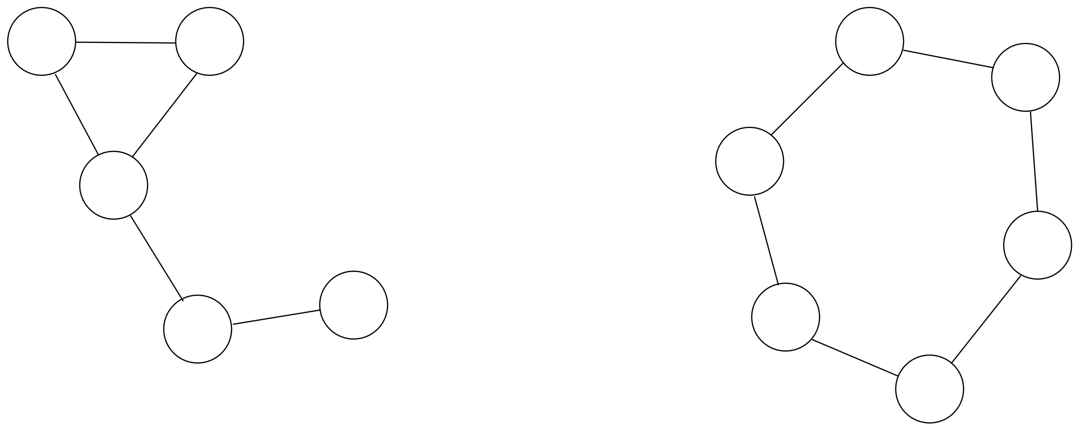
Sample Output

```
The number 12 appears in location 7.
The number 21 appears in location 15.
The number 2 appears in location 2.
The number 33 appears in location 21.
The number 100000 appears in location 97110.
The number 299977 appears in location 584871.
```

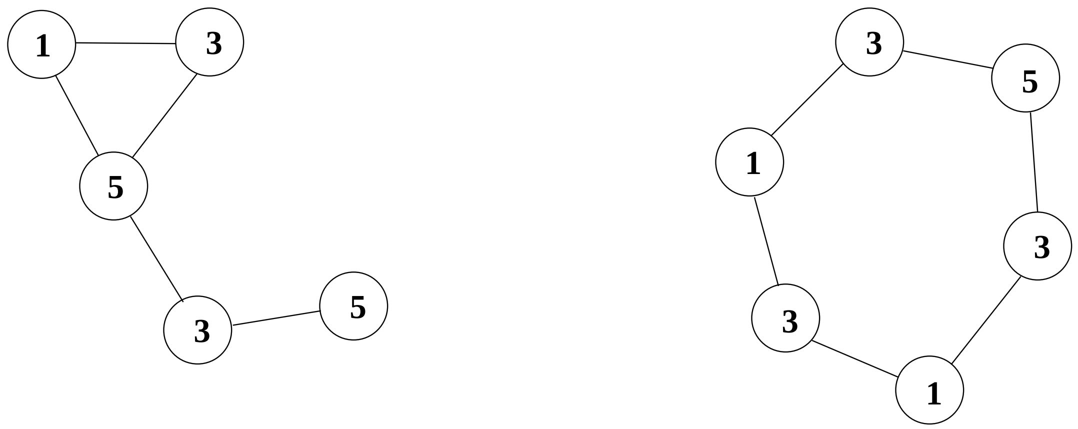
Problem D: Phone Home

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 ... 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 E: Polly Nomials

The Avian Computation Mission of the International Ornithologists Union is dedicated to the study of intelligence in birds, and specifically the study of computational ability. One of the most promising projects so far is the "Polly Nomial" project on parrot intelligence, run by Dr. Albert B. Tross and his assistants, Clifford Swallow and Perry Keet. In the ACM, parrots are trained to carry out simple polynomial computations involving integers, variables, and simple arithmetic operators.

When shown a formula consisting of a polynomial with non-negative integer coefficients and one variable x, each parrot uses a special beak-operated PDA, or "Parrot Digital Assistant," to tap out a sequence of operations for computing the polynomial. The PDA operates much like a calculator. It has keys marked with the following symbols: the digits from 0 through 9, the symbol 'x', and the operators '$+$', '$\times$', and '$=$'. (The x key is internally associated with an integer constant by Al B. Tross for testing purposes, but the parrot sees only the 'x'.)

For instance, if the parrot were presented with the polynomial

$$x^3 + x + 11$$

the parrot might tap the following sequence of symbols:

$$x,\ \times,\ x,\ \times,\ x,\ +,\ x,\ +,\ 1,\ 1,\ =$$

The PDA has no extra memory, so each $\times$ or $+$ operation is applied to the previous contents of the display and whatever succeeding operand is entered. If the polynomial had been

$$x^3 + 2x^2 + 11$$

then the parrot would not have been able to "save" the value of x^3 while calculating the value of $2x^2$. Instead, a different order of operations would be needed, for instance:

$$x,\ +,\ 2,\ \times,\ x,\ \times,\ x,\ +,\ 1,\ 1,\ =$$

The cost of a calculation is the number of key presses. The cost of computing x^3+x+11 in the example above is 11 (four presses of the x key, two presses of '$\times$', two presses of '$+$', two presses of the digit '1', and the '$=$' key). It so happens that this is the minimal cost for this particular expression using the PDA.

You are to write a program that finds the least costly way for a parrot to compute a number of polynomial expressions. Because parrots are, after all, just bird-brains, they are intimidated by polynomials whose high-order coefficient is any value except 1, so this condition is always imposed.

Input

Input consists of a sequence of lines, each containing a polynomial and an x value. Each polynomial $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$ is represented by its degree followed by the non-negative coefficients $a_n, \ldots, a_0$ of decreasing powers of x, where a_n is always 1. Degrees are between 1 and 100. The coefficients are followed on the same line by an integer value for the variable x, which is always either 1 or -1. The input is terminated by a single line containing the values 0 0.

Output

For each polynomial, print the polynomial number followed by the value of the polynomial at the given integer value x and the minimum cost of computing the polynomial; imitate the formatting in the sample output.

Sample Input

```
3 1 0 1 11 1
3 1 0 2 11 -1
0 0
```

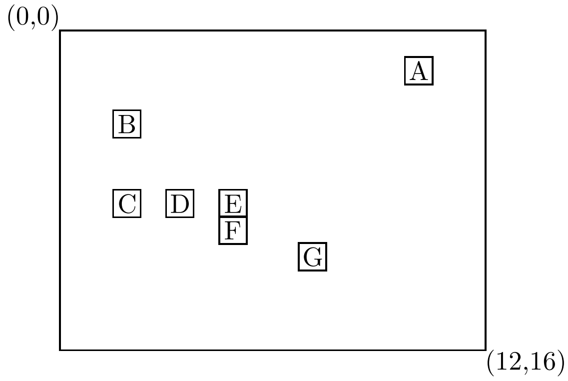
Sample Output

```
Polynomial 1: 13 11
Polynomial 2: 8 11
```

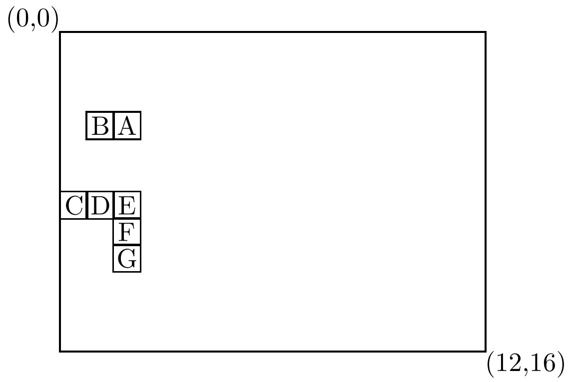
Problem F: Pushing Boxes

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) $\ldots$ (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

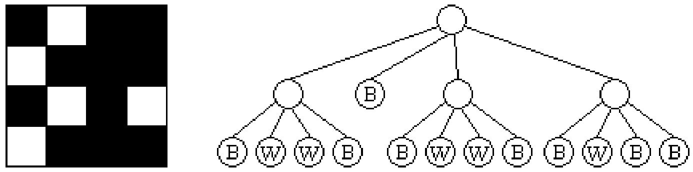
```
12 16
7 1 13 3 2 6 2 6 4 6 6 7 6 8 9
down 3
left 14
done
4 4
3 1 0 2 1 2 3
right 3
up 2
left 1
done
0 0
```

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).
```

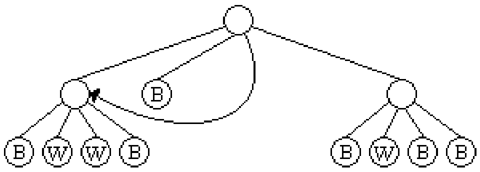
Problem G: Squadtrees

Quadtrees are data structures used to store digital images. For our purposes, the images will be simple bitmaps, where every pixel is either a 1 (black) or a 0 (white). A quadtree representation of a bitmap is obtained as follows: first, associate the root node with the entire image. If the entire image is either all 1's or all 0's, then store that value in the node and you're done. Otherwise divide the region into four equal size quadrants, add four children to the root, and assign each child one of the four regions in the following order: the first child gets the upper left quadrant, the second the upper right, the third the lower left and the fourth the lower right. Then recursively apply the above rules to each of the children. For example, the 4x4 image on the left would be represented by the quadtree on the right:



Note that this procedure only works as stated if the image is a square and has a side length equal to a power of 2. For those images which do not meet those requirements, we pad the rows and columns with 0's (on the right and on the bottom, respectively) until we have a bitmap of the appropriate size. For example, a 5x13 image would be converted to a 16x16 bitmap (with the original image residing in the upper left portion, and the remainder of the image filled with 0's).

While quadtrees can result in a significant savings in space over the original image, even more savings can be achieved if we identify repeated subtrees. For example, in the tree above, the first and third subtrees of the root are identical, so we could replace the root of the third subtree with a reference to the first subtree, obtaining something that symbolically looks like the following:



We will call these compressed quadtrees *super quadtrees*, or *squadtrees*. For our purposes the use of a reference saves space only when the tree it replaces has height of at least 1. Thus, while we could replace 5 of the nodes which contain a B with references to the first node with a B, this would not in practice save any space in the compression. Using this rule, our squadtree contains only 12 nodes, as opposed to 17 in the original quadtree. Your job for this problem is to take a set of images and determine the number of nodes in both the resulting quadtrees and squadtrees.

Input

Input will consist of multiple problem instances. Each instance will start with a single line containing two integers n and m, indicating the number of rows and columns in the image. The maximum values for these integers is 128. The next n lines will each contain m characters representing the image to process. A black pixel will be represented by a '1', and a white pixel will be represented by a '0'. The input line 0 0 will terminate input and should not be processed.

Output

For each problem instance, output two integers separated by a single space. The first value is the number of nodes in the quadtree for the problem instance, and the second is the number of nodes in the squadtree.

Sample Input

```
4 4
1011
0111
1010
0111
6 7
1110111
1010101
0000000
0100010
1011101
1010101
0 0
```

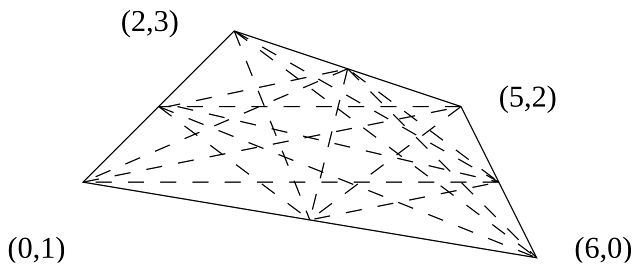
Sample Output

```
17 12
61 24
```

Problem H: 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
```