CALIFORNIA STATE UNIVERSITY, LOS ANGELES
PROGFEST 2012

Problem 1
Image Segmentation using Clustering

It is the year 2112. The earth has run out of natural resources, and the fate of humanity depends on finding a suitable earth-like replacement.

Years ago, NASA had predicted this would occur, and had sent several probes to investigate potential candidates. For the past 50 years, the probes have been returning imagery of each planet they encountered. To aid scientists in their search, you have been chosen to narrow down the list of potential candidates. One simple criteria would be to identify planets that have an appropriate amount of water, land, and vegetation. For example, the earth is about 70% water, 20% land, and 10% vegetation.

For this problem, you will implement the k-means algorithm to partition the image pixels into 3 groups (a.k.a. clusters). Each pixel is represented by a vector of its red, green, and blue components (collectively referred to as RGB values). The algorithm works by assigning pixels to a group, and iteratively refining those groups until they best represent their constituent pixels. More specifically, it can be described as follows:

1. Initialize the cluster centers to a random vector of pixels
2. Using euclidean distance, assign each pixel to a group based on its closest cluster. The euclidean distance between the cluster center vector $\mathbf{c}$ and pixel vector $\mathbf{p}$ is defined as:

$$d(\mathbf{c}, \mathbf{p}) = \sqrt{(c_{red} - p_{red})^2 + (c_{green} - p_{green})^2 + (c_{blue} - p_{blue})^2}$$

3. Compute a new cluster center by computing the average of all pixels in a group
4. Repeat from (2) until convergence (the pixels do not change groups)

The input to your program will be given in a text file, to be specified as a command line argument. To facilitate testing, we will provide you the initialization vector in the input text file. Thus, the format for the input will be as follows:

- Line 1: Initial vector of cluster centers given as comma-separated RGB values.
  Eg. $cluster1_{red}, cluster1_{green}, cluster1_{blue}, \ldots, cluster3_{red}, cluster3_{green}, cluster3_{blue}$
- Lines 2 to EOF: A pixel of satellite imagery, represented by a comma-separated RGB values

Your program must output the fraction of pixels that belong to each of the 3 classes, each on a new line, rounded to 3 decimal digits. Ideally, these would correspond to the fraction of the planet consisting of water, land, and forest.

SAMPLE INPUT:

```
255.0, 0.0, 0.0, 0.0, 255.0, 0.0, 0.0, 0.0, 255.0
183.0, 180.0, 81.0
46.0, 108.0, 147.0
67.0, 71.0, 207.0
```

```
152.0, 96.0, 99.0
41.0, 200.0, 183.0
173.0, 33.0, 215.0
16.0, 57.0, 87.0
69.0, 187.0, 40.0
159.0, 84.0, 154.0
224.0, 250.0, 125.0
```

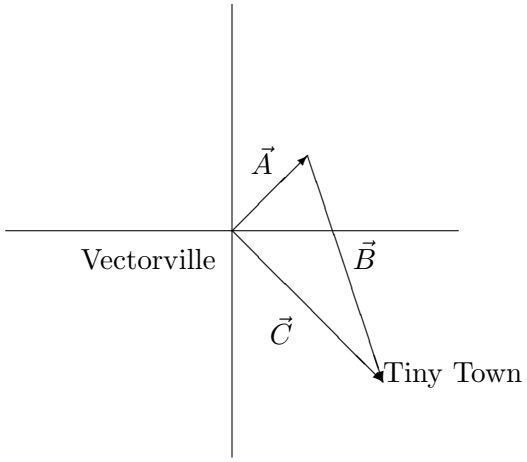
SAMPLE OUTPUT:

```
0.300
0.300
0.400
```

CALIFORNIA STATE UNIVERSITY, LOS ANGELES
PROGFEST 2012

Problem 2
Nautical Navigation

The Navy of Vectorville is looking to change the way it handles navigation. Due to recent treaties with their neighbor Polarpolis, new waterways have opened up that could significantly reduce the time it takes to move from one point to another. For example, to reach Tiny Town, Vectorville's Navy previously had to use Route $\vec{A}$ followed by Route $\vec{B}$. Now, they can directly use Route $\vec{C}$ (see below).



Each route is represented as a series of vectors, specified by its x and y components of shift. For example, Route $\vec{A}$ above is $< 1, 1 >$ and Route $\vec{B}$ is $< 1, -3 >$. Vectorville is interested in determining the new, shorter Route $\vec{C}$ and the amount of distance saved for using this new route.

However, in return for access to the waterways, Polarpolis has demanded Vectorville to use their standard of navigation. Thus, instead of representing Route $\vec{C}$ and other new routes through their x and y components, they must now be specified by angle (from the positive x-axis) and magnitude.

Your input to the program will come from a text file specified as a command line argument. Each line will consist of a vector (whitespace separated x, y pair) representing part of the path. A "0" will be used to separate one path from another, and the EOF will signify the end of input.

For each path given, your program must output the shortest route to the final coordinate, using the Polarpolis convention, as well as the distance saved on a new line. These values must be separated by a single whitespace, with the angle in radians from 0 to 2π coming first, followed by the magnitude, and finally the distance saved. Each value must be rounded to 4 decimal digits. The sample input below is for the path above, as well as one more.

SAMPLE INPUT:

```
1 1
1 -3
0
-3 4
```

```
6 0
```

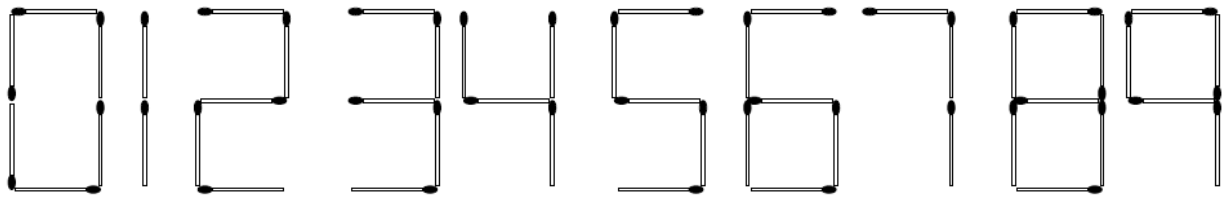
SAMPLE OUTPUT:

```
5.4978 2.8284 1.7481
0.9273 5.0000 6.0000
```

CALIFORNIA STATE UNIVERSITY, LOS ANGELES
PROGFEST 2012

Problem 3
Matchstick Mayhem

Problem: Matchsticks can be used to create "digital" numbers as shown:



You are given N matches, and you must find the number of different numbers that can be represented using the N matches. All of the numbers will be greater than or equal to 0, so no negative signs should be taken into account. For example: if you are given 3 matches, then you can only make the numbers 1 or 7. If you are given 4 matches, then you can make the numbers 1, 4, 7, or 11! Do not take into consideration any leading zeros (i.e. 001, 042, etc), only 0 can start with a 0.

The input must be a sequence of positive integers in free format. For each N, $1 \leq N \leq 80$, output the number of different (non-negative) numbers representable with $\leq N$ matches. Your answers will all be $< 2^{62}$

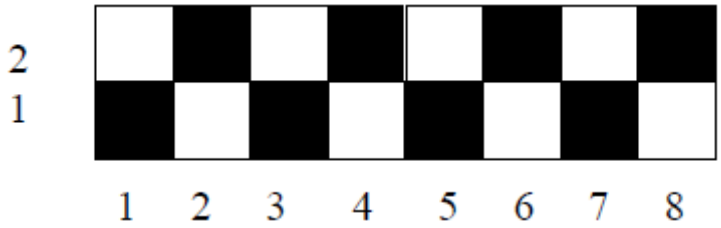
Input: A sequence of positive integers i, $1 \leq i \leq 80$, indicating how many matches you have.

Output: A sequence of positive integers j, $0 \leq j < 2^{62}$ indicating the count of how many numbers can be created using the number of matches from the input.

Sample Input	Sample Output
3 4 2	2 4 1

CALIFORNIA STATE UNIVERSITY, LOS ANGELES
PROGFEST 2012

Problem 4
Jumping Checkers



Problem: Checkers is always played on an 8 x 8 grid. The checker pieces are placed on the black squares of the grid only. Each piece moves towards the other end of the grid by moving to an unoccupied, adjacent black square. For example, a checker piece at location (1, 1) would move to location (2, 2) if it were unoccupied. Pieces may "jump" over and capture an opponent's checker piece if that piece is in an adjacent black square and the landing black square is unoccupied. I.E. a checker at location (1, 1) could "jump" over an opponent's checker at location (2, 2) if location (3, 3) was unoccupied. For this game, your "home row" is row 1 and your opponent's home row is row 8. If your checker lands in your opponent's home row, the checker becomes a "king" and can then jump both forwards and backwards.

Input: The number and location of your checkers, followed by the number of your opponent's checkers and their locations. Locations are given in ordered pair format (row, column).
Sample In put line #1 below indicates that you have 1 checker at location (1, 5), and your opponent has 2 checkers at locations (2,6) and (4,6).

Output: Given the board setup, it will be your move. You must make the move that gives the most jumps. Your output will be the greatest number of jumps possible for one move.

Sample Input	Sample Output
1,1,5,2,2,6,4,6	2
1,6,2,3,7,3,7,5,5,7	3
1,1,5,3,2,4,2,6,4,6	2
2,1,3,1,5,6,2,4,2,6,4,2,4,6,6,6,4,4	3

CALIFORNIA STATE UNIVERSITY, LOS ANGELES
PROGFEST 2012

Problem 5
Great Wall Game

Problem: This game is played with *n* stones on an *n x n* grid. The stones are placed at random in the squares of the grid, at most one stone per square. In a single move, any single stone can move into an unoccupied location one unit horizontally or vertically in the grid. The goal of the puzzle is to create a "wall," i.e., to line up all *n* stones in a straight line either horizontally, vertically, or diagonally using the fewest number of moves. An example for the case $n = 5$ is shown in Figure 1(a). In Figure 1(b) it is shown that with six moves we can line all the stones up diagonally. No smaller number of moves suffices to create a line of five stones. (However, there are other solutions using six moves, e.g., we can line up all five stones in the third column using six moves.)

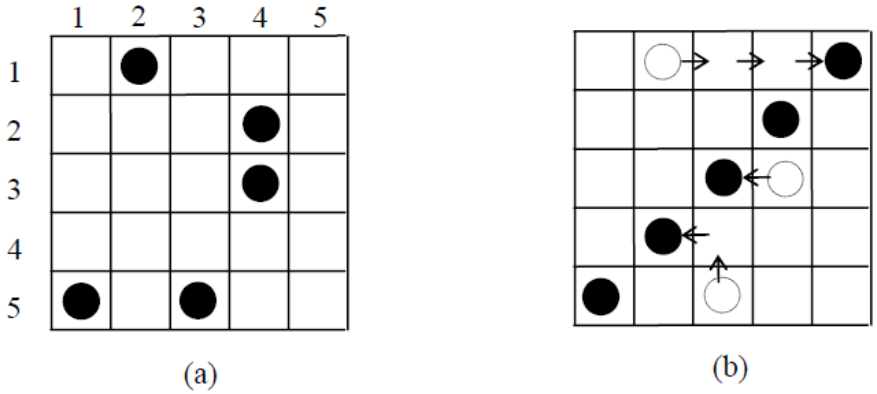


Figure 1. Starting board (a) and a 6-move solution (b) for $n = 5$

There is just one problem! We have no idea what the minimum number of moves is for any given starting board. We would like you to write a program that can take any starting configuration and determine the fewest number of moves needed to create a wall.

Input: The input consists of multiple cases. Each case begins with a line containing an integer n, $1 \le n \le 15$. The next line contains the row and column numbers of the first stone, followed by the row and column numbers of the second stone, and so on. Rows and columns are numbered as in the above diagram. The input data for the last case will be followed by a line containing a single zero.

Output: For each input case, display the case number (1, 2, ...) followed by the minimum number of moves needed to line up the *n* stones into a straight-line wall. Follow the format shown in the sample output.

Sample Input	Sample Output
5 1 2 2 4 3 4 5 1 5 3 2 1 1 1 2 3 3 1 1 2 2 2 0	Board 1: 6 moves required. Board 2: 0 moves required. Board 3: 1 moves required.

CALIFORNIA STATE UNIVERSITY, LOS ANGELES
PROGFEST 2012

Problem 6
Cracking the Code

Problem: A plain text message p of length n is to be transmitted over a secure channel. The sender chooses an integer $m \geq 2n$, and integers s, t, i, and j, where $0 \leq s, t, i, j < m$ and $i < j$. The scheme works as follows: m is the length of the transmitted ciphertext string, c. Initially, c contains m empty slots. The first letter of p is placed in position s of c. The kth letter, $k \geq 2$, is placed by skipping over i empty slots in c after the (k-1)st letter, wrapping around to the beginning of c if necessary. Slots already containing letters are not counted as empty. For instance, if the message is PRAGUE, if s = 1, i = 6, and m = 15, then the letters are placed in c as follows:

```
A P _ U _ _ _ _ R G _  _  E  _  _
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

Starting with the first empty slot in or after position t in string c, the plain text message is entered again, but this time skipping j empty slots between letters. For instance, if t = 0 and j = 8, the second copy of p is entered as follows (beginning in position 2, the first empty slot starting from t = 0):

```
A P P U R _ A U R G E  G  E  _  _
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

Finally, any remaining unfilled slots in c are filled in with randomly chosen letters:

```
A P P U R A A U R G E  G  E  W  E
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

Supposedly, the duplication of the message, combined with the use of random letters, will confuse decryption schemes based upon letter frequencies and that, without knowledge of s and i, no one can figure out what the original message is. Your job is to try to prove this idea wrong. Given a number of ciphertext strings (and no additional information), you will determine the longest possible message that could have been encoded using the above.

Input: A number of ciphertext strings, one per line. Each string will consist only of upper case alphabetic letters, with no leading or trailing blanks; each will have length between 2 and 40. Input for the last test case is followed by a line consisting of the letter X.

Output: For each input ciphertext string, print the longest string that could be encrypted in the ciphertext. If more than one string has the longest

Sample Input	Sample Output
APPURAAURGEGEWE ABABABAB THEACMPROGRAMMINGCONTEST X	Code 1: PRAGUE Code 2: Codeword not unique Code 3: Codeword not unique

CALIFORNIA STATE UNIVERSITY, LOS ANGELES
PROGFEST 2012

Problem 7
Perfectly Produced Palindromes

Problem: A positive integer is said to be a palindrome with respect to base b, if its representation in base b reads the same from left to right as from right to left. Palindromes are formed as follows:
Given a number, reverse its digits and add the resulting number to the original number. If the result isn't a palindrome, repeat the process. For example, start with 87 base 10. Applying this process, we obtain:

```
87 + 78 = 165
          165 + 561 = 726
                      726 + 627 = 1353
                                  1353 + 3531 = 4884, a palindrome
```

Whether all numbers eventually become palindromes under this process is unproved, but all base 10 numbers less than 10,000 have been tested. Every one becomes a palindrome in a relatively small number of steps (of the 900 3-digit numbers, 90 are palindromes to start with and 735 of the remainder take fewer than 5 reversals and additions to yield a palindrome). Except, that is, for 196. Although no proof exists that it will not produce a palindrome, this number has been carried through to produce a 2 million-digit number without producing a palindrome.

Input: 5 sets of data. Each set will consist of a positive integer and its base. Bases will be in the range 10 – 16.

Output: Print the palindrome produced. If no palindrome is produced after 10 additions, print the word "none" and the last sum.

Sample Input	Sample Output
A23, 16	D4D
A345, 12	9B4B9
196, 10	NONE, 18211171

CALIFORNIA STATE UNIVERSITY, LOS ANGELES
PROGFEST 2012

Problem 8
Tunneling Terrors

Problem: Curses! A handsome spy has somehow escaped from your elaborate deathtrap, overpowered your guards, and stolen your secret world domination plans. Now he is running loose in your volcano base, risking your entire evil operation. He must be stopped before he escapes!

Fortunately, you are watching the spy's progress from your secret control room, and you have planned for just such an eventuality. Your base consists of a complicated network of rooms connected by non-intersecting tunnels. Every room has a closed-circuit camera in it (allowing you to track the spy wherever he goes), and every tunnel has a small explosive charge in it, powerful enough to permanently collapse it. The spy is too quick to be caught in a collapse, so you'll have to strategically collapse tunnels to prevent him from traveling from his initial room to the outside of your base.

Damage to your base will be expensive to repair, so you'd like to ruin as few tunnels as possible. Find a strategy that minimizes the number of tunnels you'll need to collapse, no matter how clever the spy is. To be safe, you'll have to assume that the spy knows all about your tunnel system. Your main advantage is the fact that you can collapse tunnels whenever you like, based on your observations as the spy moves through the tunnels.

Input: The input consists of several test cases. Each test case begins with a line containing integers R ($1 \le R \le 50$) and T ($1 \le T \le 1000$), which are the number of rooms and tunnels in your base respectively. Rooms are numbered from 1 to R. T lines follow, each with two integers x, y ($0 \le x,y \le R$), which are the room numbers on either end of a tunnel; a 0 indicates that the tunnel connects to the outside. More than one tunnel may connect a pair of rooms. The spy always starts out in room 1. Input is terminated by a line containing two zeros.

Output: For each test case, print a line containing the test case number (beginning with 1) followed by the minimum number of tunnels that must be collapsed, in the worst case. Use the sample output format and print a blank line after each test case.

Sample Input	Sample Output
4 6 1 2 1 3 2 4 3 4 4 0 4 0 4 6 1 2 1 3 1 4 2 0 3 0 4 0 0 0	Case 1: 2 Case 2: 2

CALIFORNIA STATE UNIVERSITY, LOS ANGELES
PROGFEST 2012

Problem 9
Pilgrimage Problems

Problem: Jack is making a long distance walk with some friends along the old pilgrim road from Vézelay to Santiago de Compostela. Jack administers money for the group. His administration is quite simple. Whenever an amount (€ 60, say) has to be paid for the common good he will pay it, and write in his booklet: `PAY 60.`

When needed, Jack will ask every member of the group, including himself, to pay an amount (€ 50, say) to the collective purse, and write in his booklet: `COLLECT 50.` If the group size is 7, he collects € 350 in total.

Unfortunately some of the group members cannot participate in the full walk. So sometimes the group will grow, sometimes it will shrink. How does Jack handle these comings and goings of group members in terms of collective money? Suppose, for example, the group size is 7, and that Jack has € 140 in cash, which is € 20 for every group member. If two group members leave, each will receive € 20, and Jack will write in his booklet: `OUT 2.` If under the same circumstances three new group members arrive, they will each have to pay € 20, and Jack will write: `IN 3.`

In these cases the amount in cash could easily be divided, without fractions. As a strange coincidence, this happened during the whole trip. Jack never had to make calculations with fractional numbers of euros.

Near the end of the trip, Jack was joined by all his fellow travelers. Nobody was willing to miss the glorious finale of the trip. It was then that Jack tried to remember what the group size had been during each part of the trip. He could not remember.

Given a page of Jack's booklet, could you figure out the size of the group at the beginning of that page?

Input: The input file contains several test cases. Each test case is a sequence of lines in Jack's booklet. The first line of each test case will give the number N ($0 < N \leq 50$) of lines to follow. The next N lines have the format:

<keyword> <num>, where *<keyword>* = PAY | COLLECT | IN | OUT and *<num>* is a positive integer, with the following restrictions:

- IN *k* $k \leq 20$
- OUT *k* $k \leq 20$
- COLLECT *k* $k \leq 200$
- PAY *k* $k \leq 2000$

The last case is followed by a line containing a single zero.

Output: For each test case, print a single line describing the size of the group at the beginning of the part of the trip described in the test case. This line contains:

- The word `IMPOSSIBLE`, if the data are inconsistent.

- A single number giving the size of the group just prior to the sequence of lines in Jack's booklet, if this size is uniquely determined by the data.
- Several numbers, in increasing order, separated by spaces, giving the possible sizes of the group, in case the number of solutions is finite, but the solution is not unique.
- A statement in the format: `SIZE >=` *N*, giving a lower bound for the size of the group, in case the number of solutions is infinite. Observe that the inequality `SIZE >= 1` always applies, since at least Jack himself did the whole trip.

Sample Input	Sample Output
5 IN 1 PAY 7 IN 1 PAY 7 IN 1 7 IN 1 COLLECT 20 PAY 30 PAY 12 IN 2 PAY 30 OUT 3 3 IN 1 PAY 8 OUT 3 1 OUT 5 0	IMPOSSIBLE 2 3 7 SIZE >= 6