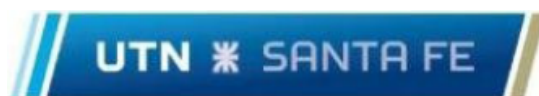


Encuentro Provincial Santafesino de Programación Competitiva



Problem Set

26 de Septiembre de 2009

1	Yupana	1
2	Fair division	3
3	Eights	4
4	Fantastic lottery	5
5	Collider	7
6	Primo Krypta	9
7	Flatland	11
8	Plane division	13

PROBLEM 1

Yupana

In 1588, José de Acosta wrote about Inca mathematicians that “... watching them in another type of quipus, where corn grains are used to count, is a fascinating experience, because these Indians can do the same accounts that a very expert accountant is to do by means of a pen, some ink and a lot of complicated operations to know, for example, the true amount of a tribute he is due. With the help of their grains, they put one seed here, three seeds there and eight seeds I don’t know where and, after moving one of them from here and changing down three of them, they succeed in doing their accounts without the smallest error (...)”.

The device they used for performing calculations is called *yupana*. The working principle of a yupana is unknown, but scholars formulated several interpretations which are compatible with historical records.

A yupana is composed by 20 cells, distributed in four columns and five rows. Cells in each column have a fixed number of holes where seed or corn grains may be deposited. From left to right there are 1, 2, 3 and 5 holes (these are the first numbers of the Fibonacci serie!).





















The number of grains (or seeds) in a cell located in the i -th row (from top to bottom) is multiplied by i , and each column represent a power of ten, from left to right: 1, 10, 100 and 10.000. Thus, two grains in the cell located in the 4th row and the 3rd column represents the value $2 \times 4 \times 100 = 800$. The total value represented in the yupana is the sum of the values represented by each cell. A cell without grains represents a value of zero. Note that some numbers cannot be represented by following this procedure (it is believed that Incas used grains with different shapes and unit-values for overcoming this issue, but we will ignore this fact).

A number is always represented using the minor possible amount of grains. For instance, 40 is represented by a single grain in the 40-cell (4th row, 2nd column) instead of placing two grains in the 20-cell (2nd row, 2nd column).

Furthermore, if there are several ways of representing a number using the minimal amount of grains, the chosen representation favors placing grains in higher-value positions. For instance, 70 is represented as $20+50$ instead of $30+40$, because it involves the 50-cell (5th row, 2nd column), and 777 is represented as $2+5+20+50+200+500$.

After this short explanation, the reader should find easy to represent a given integer number in a yupana.

(You are the reader.)

	x1	x10	x100	x10000	
1					x1
2					x2
3					x3
4					x4
5					x5

The value of the cell-multipliers of a yupana (as pictured in Felipe Guaman Poma de Ayala’s chronicle, circa 1615).

INPUT

The input file will consist of multiple test cases. Each test case will be given as a single integer (in decimal notation) between 1 and 1.000.000. There will be only one test case per line, and a value of $n = 0$ will indicate the end of input.

OUTPUT

For each input value, the output should be written as five 4-character lines, where each character (0, 1, 2, 3, 4 or 5) represents the number of grains to be deposited in a cell of the yupana. If the input cannot be represented in the yupana, each position will contain the uppercase letter N.

There will be an empty line *after each* yupana.

SAMPLE

INPUT	OUTPUT
40	0000
70	0000
777	0000
286	0100
4816	0000
1000	
0	0000
	0100
	0000
	0000
	0100
	0000
	1110
	0000
	0000
	1110
	1000
	0010
	0100
	0000
	1100
	NNNN
	NNNN
	NNNN
	NNNN
	NNNN
	0000
	0000
	0000
	0000
	0020

PROBLEM 2

Fair division

Each time you play in “Arcade Classic Machines” (ACM), you receive a ticket for k ($1 \leq k \leq 6$) points. You can accumulate your tickets and exchange them for different stuffs, like pencils (50 points), candies (100 points), pendrives (500 points), pacemakers (10.000 points), etc.

Daniel, Ezequiel and Javier (the silver triplet), had played from Monday to Friday, and their mother, Silvia, has keep all the tickets they won. On Saturday, the kids decide to exchange all tickets they have, and you must help Silvia to decide if she can distribute all the tickets with the following constraint: each kid must have exactly the same quantity of points.

INPUT

The input contains several test cases. The first line of a test case contains one integer N indicating the number of tickets accumulated by Silvia ($1 \leq N \leq 50$). The second line contains N integers X_i , separated by single spaces ($1 \leq X_i \leq 6$, for $0 \leq i \leq N - 1$). Each X_i represents the points given by the i -th ticket .

The end of input is indicated by a line containing only a zero.

OUTPUT

For each test case in the input, your program must print a single line, containing an integer P representing the total points each child will receive. If it is not possible to distribute the tickets, you must print the value -1.

SAMPLE

INPUT	OUTPUT
3	-1
1 1 2	9
8	-1
4 3 6 2 5 1 4 2	
3	
3 4 5	
0	

PROBLEM 3

Eights

Pattern Matchers have been designed for various sorts of patterns. Mr. HKP likes to observe patterns in numbers. After completing his extensive research on the squares of numbers, he has moved on to cubes. Now he wants to know all numbers whose cube ends in 888.

Given a number k , help Mr. HKP find the k -th number (indexed from 1) whose cube ends in 888.

INPUT

The first line of the input contains an integer t , the number of test cases. t test cases follow.

Each test case consists of a single line containing a single integer k ($1 \leq k \leq 2000000000000$).

OUTPUT

For each test case, output a single integer which denotes the k th number whose cube ends in 888. The result will be less than 2^{63} .

PROBLEM 4

Fantastic lottery

The *fantastic lottery* choose every week three distinct numbers between 1 and 36. The prize is calculated as follows: one dollar for each lottery ticket played, plus the accumulated jackpot from previous weeks, if any. If there are no winners, the prize is accumulated for the next week.

Although this lottery is very popular, since the first drawing has occurred a particular situation: in all of them there were at most one winner.

As Government Auditor, you have information about every drawing, and you must write a program to identify, for each drawing, the number of the winning ticket, if any, and the money his owner had won.

INPUT

The input is composed of the following information.

- A line containing one integer N , representing the number of drawings ($1 \leq N \leq 100$).
- N drawing descriptions, each one composed of:
 - A line containing a number M ($1 \leq M \leq 2000$) of tickets participating
 - M lines, containing four integers P, A, B and C ($1 \leq P \leq 10^6, 1 \leq A, B, C \leq 36$), separated by single spaces, where P is the ticket number, and A, B, C are the numbers chosen in that ticket.
 - A line composed of three numbers X, Y and Z ($1 \leq X, Y, Z \leq 36$), representing the winning numbers for this drawing.

OUTPUT

For each drawing, your program must print a line containing two integers W, P separated by a single space. If there are no winners, $W = P = 0$, otherwise, W is the winner ticket number, and P is the prize given to his owner.

SAMPLE

This sample input describes a situation with three drawings. The first drawing has two tickets, the second drawing has four tickets, and the third one has three tickets. In drawing #1 the ticket number 200 has won 2 dollars. In drawing #2 there were no winners, and the prize (4 dollars) has been accumulated for drawing #3. Then, the winner of drawing #3 has won 7 dollars (3 dollars plus the accumulated prize).

INPUT	OUTPUT
3	200 2
2	0 0
200 12 23 6	45608 7
100 30 15 12	
23 6 12	
4	
666 4 11 8	
777 11 28 3	
888 1 2 3	
999 30 20 10	
2 3 7	
3	
32100 4 6 8	
4321 5 3 7	
45608 11 22 33	
33 11 22	

PROBLEM 5

Collider

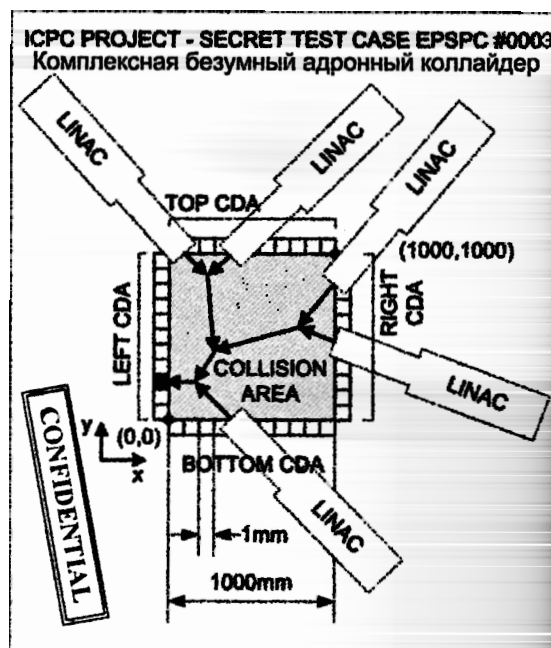
In particle physics one gains knowledge about elementary particles by accelerating particles to very high kinetic energy and letting them impact on other particles. A *collider* is a type of particle accelerator, a device which accelerates beams of particles directed against each other, so that the particles collide. Usually, only two beams of particles are accelerated, and they collide while flying in opposite directions.

The Union of Neutron Radiographers (UNR), in association with the University of Theoretical Novelty (UTN), the University of Never-Land (UNL) and ██████████, is developing the Integrated Crazy Particle Collider (ICPC) an innovative collider setup where *several* particle beams are allowed to fly in *arbitrary* directions.

The ICPC, located at ██████████, is composed by several Linear Accelerators (LINACs) which inject particle beams into the border of the collision area, which is a square of side 1000mm. On each side of the collision area there is a collision-detector array (CDA) that senses when a beam is received.

Of course, this amazing setup require strict and painful calculations in order to predict how the beams will collide. The research team has modelled the behaviour of the ICPC in the following way:

- Beams always enter the collision area from its border.
- The initial direction of a beam does always point to the interior of the collision area.
- After a beam excites the CDA, it leaves the collision area and doesn't further interact with other beams.
- There are no colineal beams.
- Beams always collide in pairs, i.e. at most two beams collide in a single point.
- When two beams collide, a reaction happens that transforms the particles into other particles, resulting a new beam moving in a direction which is the average of the directions of the colliding beams.
- At the beggining of the operation, some beams will collide first than others. This behavior is not of interest. *You may assume that after some time the beams will stabilize.*
- There will be no beam leaving the collision area near the corners of the CDAs (e.g., at $x = 0, y = 0$).



Schematic of the ICPC with 5 LINACs. A single beam leaves the collision area after exciting the left CDA.

INPUT

The input file will consist of multiple test cases. The first line of the test-case will contain a single integer N ($1 \leq N \leq 20$), indicating the number of LINACs. Each of the next N lines will contain 4 integers: X, Y ($0 \leq X, Y \leq 1000$) that represent the coordinates of the injection point (measured in millimeters; the bottom left corner is at $(0, 0)$) and d_x, d_y ($-1000 \leq d_x, d_y \leq 1000$) which represent the direction $d_x\vec{i} + d_y\vec{j}$ of the beam injected by that LINAC (i.e. the slope d_y/d_x). Either d_x or d_y may be zero, but they are not simultaneously 0.

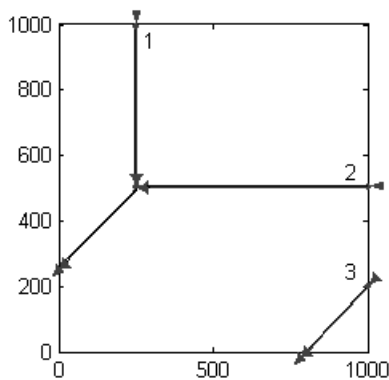
A value of $N = 0$ will indicate the end of input.

OUTPUT

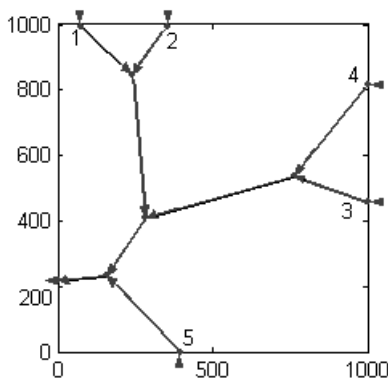
For each test case, the output will consist of a line with four integers N_L, N_B, N_R, N_T , representing the number of beams that are detected by each CDA (after the beams have stabilized). These values will be separated by a whitespace character, and preceded by an uppercase letter identifying the CDA: L, B, R, T. Trailing whitespaces are not allowed.

SAMPLE

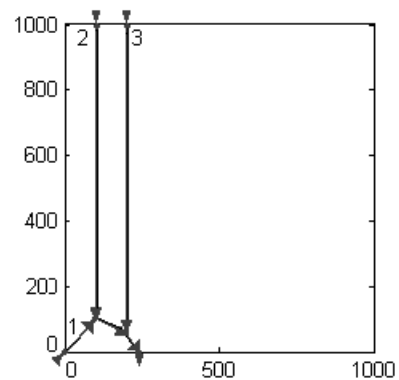
INPUT	OUTPUT
<pre> 1 0 500 1 0 3 250 1000 0 -1 1000 500 -1 0 1000 200 -2 -2 0 </pre>	<pre> L0 B0 R1 T0 L1 B1 R0 T0 </pre>
<pre> 5 72 1000 725 -659 353 1000 -404 -547 1000 456 -860 300 1000 815 -63 -73 398 0 -552 535 0 </pre>	<pre> L1 B0 R0 T0 </pre>
<pre> 3 0 0 1 1 100 1000 0 -2 200 1000 0 -2 0 </pre>	<pre> L0 B1 R0 T0 </pre>



(a) Second test case



(b) Third test case



(c) Fourth test case

PROBLEM 6

Primo Krypta

Peter is studying mathematics and he just discovered prime numbers, he also likes programming a lot, so he plans spent this weekend creating (and then coding) a new method to encrypt text messages... His main objective is to hide the messages he is receiving from his girlfriend to the curious eyes of his little brother...

His idea is the following: the program will receive a text message to encrypt (with a maximum length of 10000 characters). Then, the program will generate the output message where the input text will be hidden.

The message is encrypted, storing it —after reversed—, letter by letter, in the positions indexed by the n first prime numbers (where n is the length of the message). The remaining positions will be filled in this way:

- The first letter in the input message will be copied once in the first position unused at the output (from left to right).
- The second letter in the input message will be copied twice starting from the following unused position at the output.
- ...
- The i -th letter in the input will be copied i times starting from the next unused position (maybe the last letter used in the filling process be used lesser times)
- The blanks at the input message won't be used to this filling process (only letters and digits).
- The filling process will finish when all the unused positions before the rightmost prime position be taken.

Let's say this idea isn't brilliant..., but his little brother isn't very smart, so Peter is confident the approach will be enough.

INPUT

The input contains several test cases. The first line of a test case contains one integer N indicating the number of text messages to be encrypted ($1 \leq N \leq 50$). The following N lines contain one input message per line. The maximum length per message is 10000 characters.

You can assume there aren't spaces at the beginning or the end of the message, and there aren't two consecutive spaces.

OUTPUT

For each text message in the input, the output will contain one line with the encrypted message.

SAMPLE

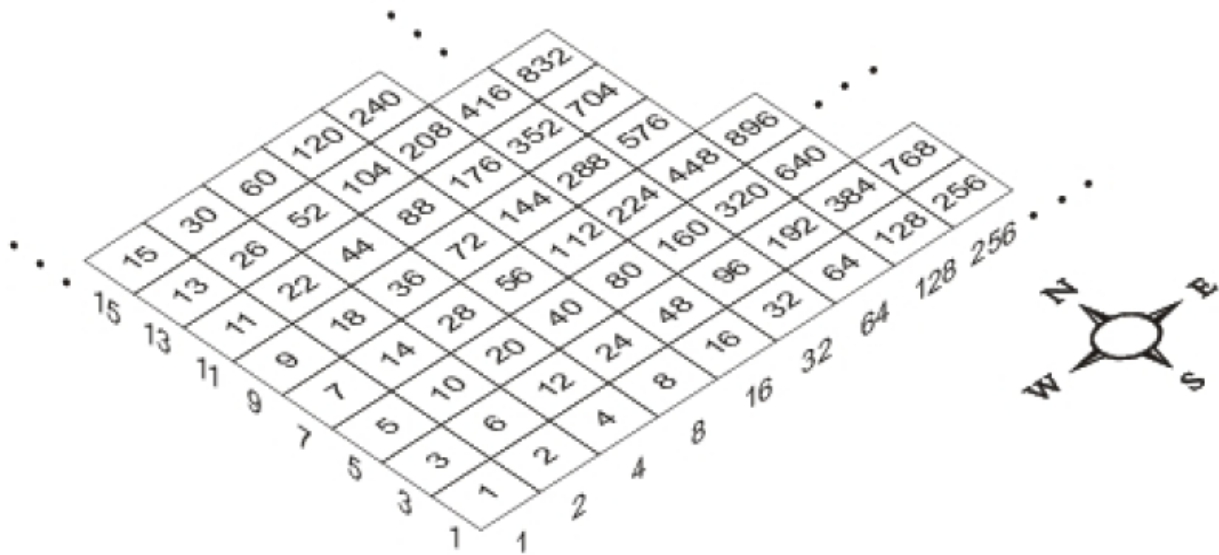
INPUT	OUTPUT
1 RUMBO A ACM	RMCUAU MMMAB BBBOOB000MOAAUAR
WE WILL BE IN TOP 3	W3 EPEOWWWTI IIINLILL LLLLLELBLBBBB BBBLELEEEIEEEEIWIIIII IEIINNWW

(Note that, for each test case, the output is a *single* line.)

PROBLEM 7

Flatland

In recognition to the number of famous mathematicians of its residents, the City of Flatland has decided to rename all its streets as numbers (positive integers to be more precise.) The streets of Flatland are organized as a grid. The city decided to number all its North-South streets using powers of two (1, 2, 4, 8,...) and all its East-West streets using odd numbers (1, 3, 5,...) . The city also decided to re-number all its buildings so that the number of each building is the result of multiplying the numbers of the two streets the building is on. For example, building #40 is at the intersection of streets 5 and 8.



The problem with this numbering scheme is that it is not easy for the residents to determine the distance between buildings. The distance between any two buildings is the number of buildings one needs to cross to go from one building to another. One can only move parallel to the streets (no diagonals or any other shortcuts.) For example, to go from building #6 to building #40, one has to travel one building north and two buildings east, so the distance is 3. Similarly, the distance from building #80 to building #88 is 4.

Help the residents of Flatland by writing a program that calculates the distance between any two given buildings

INPUT

The input is made of one or more pairs of building numbers. Each pair $\langle S, T \rangle$ appears on a single line with a single space between the two numbers. Note that $S, T < 1.000.000.000$. The end of the input is identified by the pair $\langle 0, 0 \rangle$ (which is not part of the test cases.)

OUTPUT

For each input pair $\langle S, T \rangle$, the output file should include a line of the form:

The distance between S and T is D .

SAMPLE

INPUT	OUTPUT
12 14	The distance between 12 and 14 is 3.
20 30	The distance between 20 and 30 is 6.
40 50	The distance between 40 and 50 is 12.
0 0	

PROBLEM 8

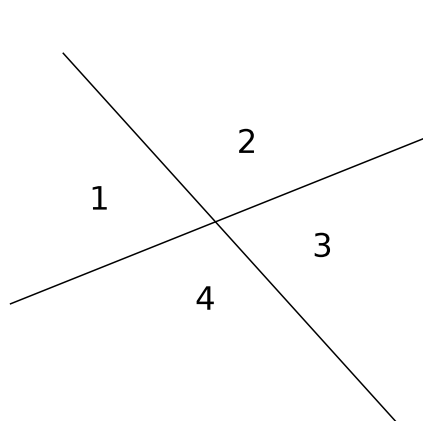
Plane division

The plane division problem has worried the *Association of Cumbersome Maths* since its creation in 1909.

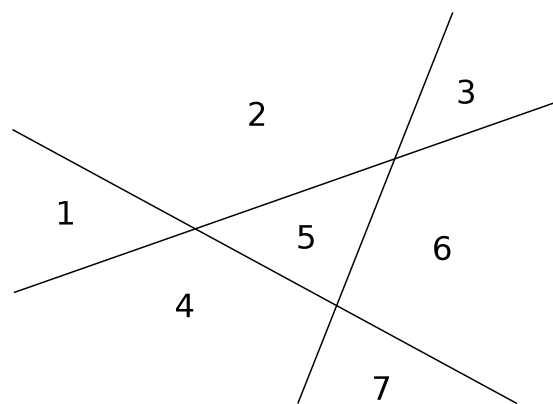
In the first association meeting, its leader presented the problem with memorable words:

“My friends, suppose you have $N + K$ lines in the plane, such that (1) there are exactly K parallel lines, (2) there are not, among the other N lines, two parallel lines and, (3) there are not, among the $N + K$ lines, three concurrent lines. I ask you, my friends, How many regions are determined by these lines?”

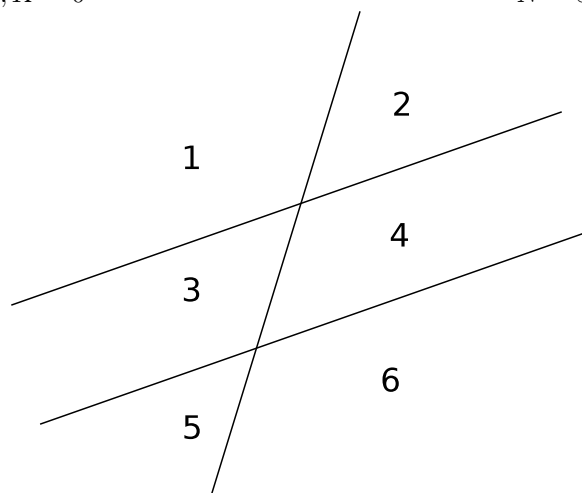
The problem started to be discussed, and some particular solutions were found (three of them are drawn below).



$N = 2, K = 0$



$N = 3, K = 0$



$N = 1, K = 2$

INPUT

The input contains several test cases. The first line of a test case contains an integer M indicating the number of lines for this case. The next M lines contains two integers A, B ($-1000 \leq A, B \leq 1000$), giving the information about each line. The number A represents the line slope, and B represents the y-intercept (the value of the y coordinate when the line intersects the vertical axis). The end of the input is indicated by a line containing only a zero.

OUTPUT

For each test case in the input, your program must print a single line, containing an integer R representing the number of regions determined by the M lines.

SAMPLE

INPUT	OUTPUT
2	4
-1 3	7
2 -4	6
3	
1 5	
3 1	
-2 5	
3	
2 -3	
2 4	
-1 1	
0	