Colombian Collegiate Programming League
CCPL 2013

Contest 1 -- February 23rd

Problems

This set contains 10 problems; pages 1 to 15.

(Borrowed from the *2012 ACM ICPC Southeast USA Regional Programming Contest -- Division II.*)

A - Candy Store

Source file name: `store.c`, `store.cpp` *or* `store.java`

You are walking with a friend, when you pass a candy store. You make a comment about how unhealthy their wares are. Your friend issues an interesting challenge: who can be the unhealthiest? Both of you will go into the store with the same amount of money. Whoever buys candy with the most total calories wins!
Since you're a smart computer scientist, and since you have access to the candy store's inventory, you decide not to take any chances. You will write a program to determine the most calories you can buy. The inventory tells you the price and calories of every item. It also tells you that there is so much in stock that you can buy as much of any kind of candy as you want. You can only buy whole pieces of candy.

Input

There will be multiple test cases in the input. Each test case will begin with a line with an integer n ($1 \leq n \leq 5,000$), and an amount of money m ($\$0.01 \leq m \leq \$100.00$), separated by a single space, where n is the number of different types of candy for sale, and m is the amount of money you have to spend. The monetary amount m will be expressed in dollars with exactly two decimal places, and with no leading zeros unless the amount is less than one dollar. There will be no dollar sign. Each of the next n lines will have an integer c ($1 \leq c \leq 5,000$) and an amount of money p ($\$0.01 \leq p \leq \$100.00$), separated by a single space, where c is the number of calories in a single piece of candy, and p is the price of a single piece of candy, in dollars and in the same format as m. The input will end with a line containing '00.00'.

The input must be read from standard input.

Output

For each test case, output a single integer, indicating the maximum amount of calories you can buy with up to m dollars. Output no spaces, and do not separate answers with blank lines.

The output must be written to standard output.

Sample input	**Sample output**
2 8.00 700 7.00 199 2.00 3 8.00 700 7.00 299 3.00 499 5.00 0 0.00	796 798

B - Collision Detection

Source file name: `detect.c`, `detect.cpp` *or* `detect.java`

As a preliminary step in developing an autonomous vehicle system, your team is seeking to prove that a central traffic controller can sound an alert when automobiles are likely to collide unless corrective actions are taken.
The test course consists of a number of straight tracks that intersect at a variety of angles. As cars pass sensors mounted on the tracks, their position and speed is recorded and sent to the central controller. The controller remembers its two most recent sets of readings for each car. We want the controller to sound the alert whenever two cars will, if they behave as predicted, pass 'dangerously close' to one another any time within the next 30 seconds (following the most recent of the sensor readings). For this purpose, consider that cars are dangerously close if they pass within 18ft. of one another. Cars are considered safe if their closest approach is at least 20ft apart. A passage within 18ft to 20ft is considered ambiguous.
Assume that

- The cars remain on their straight course
- The acceleration (change in speed per unit time) of each car remains constant over the time between observations and for the next 30 sec, with the two exceptions given below.
  - Exception 1: if the car is decelerating, it stops decelerating if its speed reaches zero (cars do not shift into reverse).
  - Exception 2: if the car is accelerating, it stops accelerating if its speed reaches 80 feet per second (about 55 MPH).

Given the two most recent sets of reading for each of two cars, determine if they will pass within 18ft of each other within 30 seconds of the last measurement.

Input

There will be multiple test cases in the input. Each test case consists of four observations, one observation per line. The first two observations are for the first car, the second two observations are for the second car.
Each observation consists of four floating point numbers t, x, y and s, separated by single spaces, where:

- t is the time of the observation in seconds ($0 \leq t \leq 120$)
- x and y give the position of the car at the time of the observation, in feet ($-5,000 \leq x, y \leq 5,000$)
- s is the speed in feet per second ($0 \leq s \leq 80$)

There will be no data sets in which the closest approach within the indicated timer interval falls in the ambiguous 18ft to 20ft. range. The two observations for a given car will always occur at distinct times, and the first time for each car will be before the second time for that car.
Input is terminated with a line with four negative numbers.

The input must be read from standard input.

Output

For each data set, print a single line consisting of either 1 if the cars will come within 18 feet of each other within 30 seconds following the maximum of the 4 input times, or 0 if not. Output no extra spaces, and do not separate answers with blank lines.

The output must be written to standard output.

Sample input	**Sample output**
10 0 0 10 11 7.42 7.42 11 11 41.0 106.0 16 12 56 106 14 0 0 0 50 0.5 21.7 12.5 50.1 0.25 39.0 22.5 50 0.75 60.7 35.0 50.1 -1 -1 -1 -1	1 0

C - Do It Wrong, Get It Right

Source file name: `doit.c`, `doit.cpp` *or* `doit.java`

In elementary school, students learn to subtract fractions by first getting a common denominator and then subtracting the numerators. However, sometimes a student will work the problem incorrectly and still arrive at the correct answer. For example, for the problem

$$\frac{5}{4} - \frac{9}{12}$$

one can subtract the numbers in the numerator and then subtract the numbers in the denominator, simplify and get the answer. i.e.

$$\frac{5}{4} - \frac{9}{12} = \frac{-4}{-8} = \frac{4}{8} = \frac{1}{2}.$$

For a given fraction b/n, your task is to find all of the values a and m, where $a \geq 0$ and $m > 0$, for which

$$\frac{a}{m} - \frac{b}{n} = \frac{a-b}{m-n}.$$

Input

There will be several test cases in the input. Each test case will consist of a single line with two integers, b and n ($1 \leq b, n \leq 10^6$) separated by a single space. The input will end with a line with two 0s.

The input must be read from standard input.

Output

For each case, output all of the requested fractions on a single line, sorted from smallest to largest. For equivalent fractions, print the one with the smaller numerator first. Output each fraction in the form a/m with no spaces immediately before or after the /. Output a single space between fractions. Output no extra spaces, and do not separate answers with blank lines.

The output must be written to standard output.

Sample input	**Sample output**
9 12 12 14 4 12 0 0	0/24 5/20 8/16 8/8 5/4 0/28 9/21 9/7 0/24 3/18 3/6

D - Dueling Philosophers

Source file name: `duel.c`, `duel.cpp` *or* `duel.java`

Following a sad and strange incident involving a room full of philosophers, several plates of spaghetti, and one too few forks, the faculty of the Department of Philosophy at ACM University have been going through the papers of a recently deceased colleague. The faculty members were amazed to find numerous unpublished essays. They believe that the essays, collected into one volume, may constitute a major work of scholarship that will give their department some much-needed positive publicity. Naturally, all of the faculty members began to vie for the honor (to say nothing of the fame) of serving as editor of the collection.
After much debate, the faculty members have narrowed the list to two candidates. Both applicants were asked to explain how they would arrange the essays within the final book. Both have noted that many of the essays define terminology and concepts that are explored in other essays. Both have agreed to the basic principle that an essay that uses a term must appear after the essay that defines that term. One of the candidates has presented what he claims is the only possible arrangement of the essays, under those constraints, and is arguing that he should be given the job simply because he has already done this major part of the work. The second candidate scoffs at this claim, insisting that there are many possible arrangements of the essays, and that an editor of true skill (himself) is needed to choose the optimal arrangement.
Write a program to determine if zero, one, or more than one arrangement of the essays is possible.

Input

There will be multiple test cases in the input. Each test case will begin with a line with two integers, n ($1 \leq n \leq 1,000$) and m ($1 \leq m \leq 500,000$), where n is the number of essays, and m is the number of relationships between essays caused by sharing terms. They will be separated by a single space. On each of the next m lines will be two integers, d followed by u ($1 \leq d, u \leq n, d \neq u$) which indicate that some term is defined in essay d and used in essay u. Integers d and u will be separated by a single space. The input will end with two 0s on their own line.

The input must be read from standard input.

Output

For each test case, output a 0 if no arrangement is possible, a 1 if exactly one arrangement is possible, or a 2 if multiple arrangements are possible (output 2 no matter how many arrangements there are). Output no extra spaces, and do not separate answers with blank lines.

The output must be written to standard output.

Sample input	**Sample output**
5 4 1 5 5 2 3 2 4 3 5 4 3 1 4 2 1 5 5 4 2 2 1 2 2 1 0 0	2 1 0

E - Paint Me

Source file name: `paintme.c`, `paintme.cpp` *or* `paintme.java`

A contractor is planning to bid on interior painting for an apartment building. These apartments are for student housing, so they are to be single-room efficiencies, and have basic drywall walls and ceilings, with no particular architectural features like crown molding. He would like to find a quicker way to estimate how much paint it will take to paint the walls and ceilings for each job. The plan for these buildings is to paint the four walls and the ceiling of the main room. The closets and bathrooms will not be painted. Of course, no paint is needed for window and door openings. All rooms, windows and doors are rectangular. All rooms will be painted the same color.
The contractor will provide you with information about the dimensions of the rooms, the windows and doors for each floor plan, and the number of apartments. Your team is to write a program that will tell him how many cans of paint he should include in his bid.

Input

There will be several test cases in the input. Each test case begins with a line with 6 integers:

$$n \; width \; length \; height \; area \; m$$

Where n ($1 \leq n \leq 100$) is the number of apartments, *width* ($8 \leq width \leq 100$) is the width of each room, *length* ($10 \leq length \leq 100$) is the length of each room, *height* ($8 \leq height \leq 30$) is the height of each room, *area* ($100 \leq area \leq 1,000$) is the area in square feet that can be covered by each can of paint, and m ($0 \leq m \leq 10$) is the number of windows and doors. On each of the next m lines will be two integers, *width* and *height*, describing a door or window. No window or door will be larger than the largest wall, and both *width* and *height* will be positive. All linear measures will be expressed in feet. The input will end with a line with six 0s.

The input must be read from standard input.

Output

For each test case, output a single integer on its own line, indicating the number of cans of paint needed to paint all of the walls and ceilings of all of the apartments. Output no extra spaces, and do not separate answers with blank lines.

The output must be written to standard output.

Sample input	**Sample output**
50 8 20 8 350 2 6 3 3 3 50 8 20 8 300 3 6 3 5 3 3 3 0 0 0 0 0 0	83 95

F - Party Games

Source file name: `party.c`, `party.cpp` *or* `party.java`

You've been invited to a party. The host wants to divide the guests into 2 teams for party games, with exactly the same number of guests on each team. She wants to be able to tell which guest is on which team as she greets them as they arrive, and as easily as possible, without having to take the time to look up each guest's name on a list. Being a good computer scientist, you have an idea: give her a single string, and all she has to do is determine whether the guest's name is alphabetically less than or greater than that string.
Given the unique names of n party guests (n is even), find the shortest possible string S such that exactly half the names are less than or equal to S, and exactly half are greater than S. If there's more than one string of the shortest length, output the one that comes first alphabetically.

Input

There will be multiple test cases in the input. Each test case will begin with an even integer n ($2 \leq n \leq 1,000$) on its own line. On the next n lines will be names, one per line. Each name will be a single word consisting only of capital letters, and will be at least one letter and no longer than 30 letters. All of the names in a test case will be unique. The input will end with a 0 on its own line.

The input must be read from standard input.

Output

For each case, output the alphabetically first of all of the shortest possible strings that your host could use to separate her guests. Output this string using all upper case letters. Do not output any spaces. Do not put a blank line between outputs.

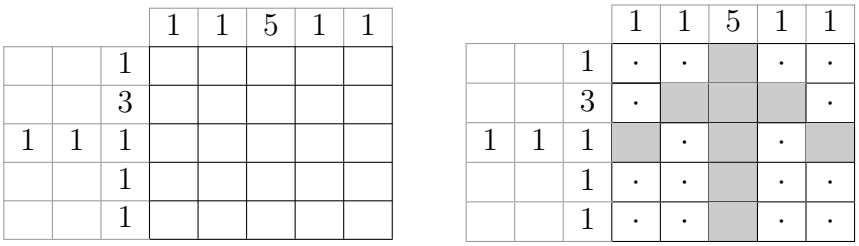
The output must be written to standard output.

Sample input	**Sample output**
4 FRED SAM JOE MARGARET 2 FRED FREDDIE 2 JOSEPHINE JERRY 0	K FRED JF

G - Reverse Nonogram

Source file name: `nonogram.c`, `nonogram.cpp` *or* `nonogram.java`

A Nonogram is a pencil puzzle played on a grid. The grid is initially blank. There are numbers on the side and top of the grid, which indicate how the grid squares should be filled in. The numbers measure how many unbroken lines of filled-in squares there are in any given row or column. For example, a clue of "4 8 3" would mean there are sets of four, eight, and three filled squares, in that order, with at least one blank square between successive groups. Here is a small example, with its solution.



You are going to work backwards. Given a Nonogram solution, produce the numbers which should be at the side and top of the grid.

Input

There will be several test cases in the input. Each test case will begin with an integer n ($2 \leq n \leq 100$) indicating the size of the grid. Each of the next n lines will have exactly n characters, consisting of either '.' for a blank square, or 'X' for a square which has been filled in. The input will end with a line with a single 0.

The input must be read from standard input.

Output

For each test case, print $2n$ lines of output. The first n lines represent the numbers for the rows, from top to bottom. The next n lines represent the numbers across the top, from left to right. If any row or column has no squares filled in, output a 0. Put a single space between numbers on the same line. Do not output any lines with leading or trailing blanks. Do not output blank lines between any lines of output.

The output must be written to standard output.

Sample input	**Sample output**
3 XXX .XX .X. 3 X.X ..X X.. 5 ..X.. .XXX. X.X.X ..X.. ..X.. 0	3 2 1 1 3 2 1 1 1 1 1 1 0 2 1 3 1 1 1 1 1 1 1 5 1 1

H - Tsunami

Source file name: `tsunami.c`, `tsunami.cpp` *or* `tsunami.java`

The country of Cartesia can be described simply by a Cartesian plane. The x-axis is a shoreline. The positive y half-plane is land, and the negative y half-plane is ocean. Several large cities dot the mainland. Their positions can be described by coordinates (x, y), with $y > 0$. Unfortunately, there are sometimes tsunamis in the ocean near Cartesia. When this happens, the entire country can flood. The waters will start at $y = 0$ and advance uniformly in the positive y direction.
Cartesia is trying to develop a tsunami warning system. The warning system consists of two components: a single meteorological center which can detect a tsunami miles out, and wired connections which can carry the warning from city to city in straight lines (No wireless communication!!). Despite the lack of wireless communications, the wired connections can carry the warning virtually instantaneously.
A city is considered safe if it either has the meteorological center, or if it has a direct wire connection to another safe city (i.e. if it has a multi-hop cable path to the single meteorological center).
Unfortunately, a simple engineering problem is made more complicated by politics! If a city A receives the warning from city B, and city B is further away from the shore than city A, then city A's leaders will complain! ‘‘‘We're closer to the ocean than city B, so the warning should have gone through us!’’ With a sigh, you agree to find a solution where no city will get the warning from a city that's further from the shore. (This means that the meteorological center must be in a city that's closest to the shore.)
Given a description of Cartesia, find the least amount of cable necessary to build a tsunami warning system where every city is *safe*, and no city will receive the warning from another city that is further from the shore.

Input

There will be several test cases in the input. Each test case will begin an integer n ($1 \leq n \leq 1,000$) on its own line, indicating the number of cities. On each of the next n lines will be a pair of integers x and y ($-1,000 \leq x \leq 1,000, 0 < y \leq 1,000$), each of which is the (x, y) location of a city. These integers will be separated by a single space. Within a test case, all (x, y) pairs will be unique. The input will end with a line containing a single 0.

The input must be read from standard input.

Output

For each test case, output a single real number on its own line, which is the minimum amount of cable which must be used to build the tsunami warning system. Output this number with exactly two decimal places. Output no extra spaces, and do not separate answers with blank lines.

The output must be written to standard output.

Sample input	**Sample output**
3 100 10 300 10 200 110 4 100 10 300 10 200 110 200 60 0	341.42 361.80

I - Unhappy Numbers

Source file name: `unhappy.c`, `unhappy.cpp` *or* `unhappy.java`

Numbers have feelings too! For any positive integer, take the sum of the squares of each of its digits, and add them together. Take the result, and do it again. A number is *happy* if, after repeating this process a finite number of times, the sum is 1. Some happy numbers take more iterations of this process to get to 1 than others, and that would be referred to as its distance from happiness. 1's distance from happiness is 0. 23's distance from happiness is 3, since $2^2 + 3^2 = 13$,$1^2 + 3^2 = 10$, and $1^2 + 0^2 = 1$. Numbers are *unhappy* if they are infinitely far away from happiness because they get stuck in a loop. Given the lower end and upper end of a range of integers, determine how many Unhappy numbers are in that range (inclusive).

Input

There will be several test cases in the input. Each test case will consist of two positive integers, *lo* and *hi* ($0 < lo \leq hi \leq 10^{18}$) on a single line, with a single space between them. Input will terminate with two 0s.

The input must be read from standard input.

Output

For each test case, output a single integer on its own line, indicating the count of *unhappy numbers* between *lo* and *hi* (inclusive). Output no extra spaces, and do not separate answers with blank lines.

The output must be written to standard output.

Sample input	**Sample output**
1 10 1 100 0 0	7 80

J - Walls

Source file name: `walls.c`, `walls.cpp` *or* `walls.java`

There are a number of research stations on a featureless patch of desert, which can be modeled as a Cartesian plane. Each station is located at some point (x, y) where x and y are even integers. For security reasons, sufficiently long and high walls are to be constructed to separate the stations so that no station is visible from any of the other stations. A wall may only be constructed along a North-South or East-West line. A vertical wall may be built at an odd x-coordinate, and a horizontal wall may be built at an odd y-coordinate. Since the stations are located at even valued coordinates, and the walls are built along odd valued coordinates, no wall can ever touch a station. The walls are always long enough to completely separate stations on one side from stations on the other side. Given a list of stations, you must determine the smallest number of walls that need to be constructed.

Input

There will be several test cases in the input. Each test case will begin with an integer n ($2 \leq n \leq 100$), which is the number of stations. The next n lines each will contain two integers x and y ($0 \leq x, y \leq 36$), separated by a single space, indicating the (x, y) location of a station. The x and y values are guaranteed to be even. Within a test case, all (x, y) locations will be unique. The last test case will be followed by a line with a single 0.

The input must be read from standard input.

Output

For each test case, output a single integer, indicating the smallest number of walls that can prevent the given n stations from seeing each other. That is, a straight line-segment joining any two stations must be intersected by at least one wall. Output no extra spaces, and do not separate answers with blank lines.

The output must be written to standard output.

Sample input	**Sample output**
4 12 12 4 8 8 6 2 4 4 0 0 4 4 10 8 14 6 0	2 3