UW-Madison ACM ICPC Individual Contest

Set up

Before the contest begins, log in to your workstation and set up and launch the PC2 contest software using the following instructions. You will use this program to submit problem solutions, receive the judges' answers, and communicate clarification requests.

1. Download the custom PC2 package into a directory of your choosing from www.cs.wisc.edu/~dieter/ICPC/13-14/pc2.tar.gz
2. In a terminal window, cd to the directory where you downloaded the package and type `tar xzvf pc2.tar.gz`
3. Type `cd pc2` followed by the command `bin/pc2team` -- this brings up your PC2 terminal that will be your interface to the judges during the contest.
4. Log in using the login ID and password given to you by the judges when you arrive. They will be of the form teamX where X is an integer, and the password will be your uva online judge username. If you have not yet sent your username to the judges, you will not be given a login and password by default -- please see the judges to get one.

The Contest

Begin the contest by solving the problem on the next page, "**count"**. This is a warmup problem designed to get you used to submitting problems via PC2. Code your solution to the problem and submit it as follows:

1. Click on the "submit run" tab in your PC2 window.
2. In the dropdown menu labeled "Problem", choose "**count**". Choose the programming language you used from the "language" dropdown menu. Then select your source code file by clicking the "select" button in the "main file" section.
3. Submit your code by clicking "Submit" (note: clicking "Test" doesn't really do anything unless you've created your own test files, so don't expect it to automatically test your program against the sample input).
4. Wait -- you will receive a judgement from the judge shortly by way of a pop-up window. If your answer comes back something other than "accepted", try again.

The remaining problems are known to PC2 as **equilateral, tree, lattice, euro, conduit, and highway**, respective of their order in this packet. You can ignore any requirements stated in the problem for source name file, e.g., "equilateral.cpp": name your source files as you please.

All input comes from standard in, all output should be sent to standard out.

You may use any online Java or C++ documentation, as well as any notes or texts you have brought with you. You may use the printer at any time. Collaboration with others or searching the web for solutions to these problems is prohibited. Please turn off your cell phones.

You may submit problem clarifications via the PC2 program at any time, but please read the problems thoroughly before doing so.

After the contest, please fill out the questionnaire on the back of this sheet, then join us in room 1325 for pizza and soda.

Name: **CS Login:**

Student status (i.e. Junior, first year grad student):

Year that you started college:_____ **Year of birth:_____**

Have you participated in the ICPC before, and if so, how many regionals and world finals have you participated in?

What do you feel are your strengths with respect to the ICPC?

Which programming languages are you proficient in, among C, C++, and Java? Which do you prefer?

Is there anyone that you would prefer to be placed on a team with (if possible)?

How did you hear about the ICPC (e-mail, flier, word of mouth, etc.):

Can you commit to training for and participating in the World Finals on June 22-26 in Ekaterinburg, Russia if you qualify?

In particular, do you forsee any barriers to obtaining a Russian visa?

Is there anything else we should take into account about you while forming teams?

Warmup Problem: Count

Can you count from one up to any number N? Write a program to prove it!

Input

The input begins with a single number that describes the number of test cases. Each test case follows on its own line, and consists of a single positive integer N <= 1,000,000 that describes how high you should count for that test case.

Output

The output for each test case should be on its own line, and consist of the numbers 1 through N (inclusive), each separated by a space.

Sample Input

```
3
3
5
10
```

Sample Output

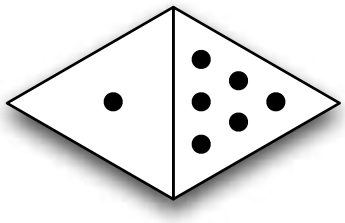
```
1 2 3
1 2 3 4 5
1 2 3 4 5 6 7 8 9 10
```

E Equilateral Dominoes (`equilateral.{c,cc,java}`)

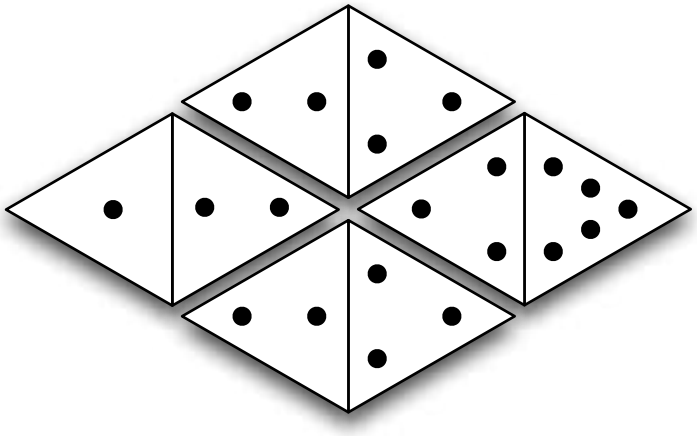
E.1 Description

We've all seen regular dominoes before—they look like rectangular tiles, twice as long as they are wide, with pips designating a number between 1 and 6 on each of their two ends. The object of a dominoes game involves some variation of tiling the dominoes so that the numbers on adjacent dominoes match up.

Now what if we were to make the dominoes out of equilateral triangles, like this:



An equilateral domino is shaped like a quadrilateral made up of two equilateral triangles. Again, the triangle at each end depicts a number from 1 to 6. Two equilateral dominoes can be tiled next to each other if the domino ends on either side of their shared edge have the same value, and if neither domino overlaps another. Once you have begun a tiling, additional dominoes may only placed adjacent to one or more of the dominoes already on the table. In other words, two more disjoint groups of dominoes does not constitute a valid tiling. For example, equilateral dominoes may be tiled like this:



Your task is to find a best tiling, in some sense, of some equilateral dominoes. If you score one point for every edge that is shared between two dominoes, what is the best way to tile a given set of equilateral dominoes to achieve the highest score?

E.2 Input

The input consists of multiple test cases. The first line of each test case contains an integer N, $1 \leq N \leq 6$, the number of equilateral dominoes in that set. This is followed by N lines with two integers each (values between 1 and 6 inclusive), with each line indicating the pip values on each of the dominoes in the set. Input is followed by a single line with $N = 0$, which should not be processed. For example:

```
4
1 2
2 3
3 2
4 3
2
5 6
2 1
4
3 2
3 4
1 5
1 6
0
```

E.3 Output

For each test case, output a single line containing the highest tiling score that can be achieved with the given set of equilateral dominoes. If there is no valid tiling of the dominoes, output a zero. For example:

```
4
0
1
```

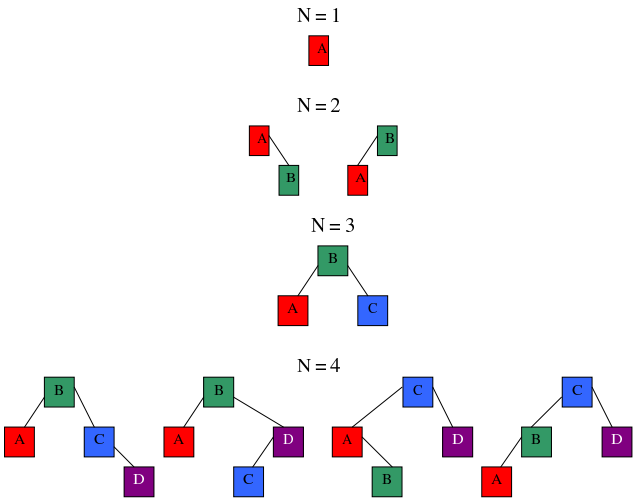
Problem K — limit 5 seconds

Tree Count

Description

Our superhero Carry Adder has uncovered the secret method that the evil Head Crash uses to generate the entrance code to his fortress in Oakland. The code is always the number of distinct binary search trees with some number of nodes, that have a specific property. To keep this code short, he only uses the least significant nine digits.

The property is that, for each node, the height of the right subtree of that node is within one of the height of the left subtree of that node. Here, the height of a subtree is the length of the longest path from the root of that subtree to any leaf of that subtree. A subtree with a single node has a height of 0, and by convention, a subtree containing no nodes is considered to have a height of -1.



Input

Input will be formatted as follows. Each test case will occur on its own line. Each line will contain only a single integer, N, the number of nodes. The value of N will be between 1 and 1427, inclusive.

Output

Your output is one line per test case, containing only the nine-digit code (note that you must print leading zeros).

Sample Input	Sample Output
1 3 6 21	000000001 000000001 000000004 000036900

Problem E — limit 10 seconds

LatticeLand

Description

LeaperLad wakes in LatticeLand on a disk suspended above a lake of lava. Leaper spies his HeloPak on one of the disks; with it he knows he can escape this nefarious trap.

The disks are quite far apart, however; without some momentum, he can only jump to an immediately adjacent disk. Once he has acquired the speed to make that jump, he can accelerate on every disk he touches.

He notices the disks are laid out in a rectangular grid, with a disk on each grid point. He calculates that on each disk he can accelerate or decelerate his speed by one unit in the horizontal or vertical direction (but not both on the same disk). Alternatively, he can just maintain his speed when stepping on a disk. Thus, in a straight line, from a standing start, he can jump one unit, then two units, then three, then two, then one.

Some pairs of disks are joined by walls of fire that he knows he must not touch. He can get arbitrarily close to one of these walls, but he must not touch one. Nor can he fall off the edge of the grid.

How quickly can LeaperLad reach his HeloPak and stop on that disk?

Input

Input will have one problem per input line. The input line will contain a sequence of integers, each separated by a single space.

The first two integers will be w and h, the width and height of the grid. Each of these values will be between 1 and 64, inclusive. Following that will be two integers representing the coordinates of the disk that LeaperLad wakes on. After that will be two integers representing the coordinates of the disk that the HeloPak is on. The next integer will be f, the number of fire walls. There will be six or fewer fire walls. After that will be f sets of 4 integers, representing the two coordinates of the end points of the walls.

For all coordinates, the first number will be between 0 and $w - 1$, inclusive, and the second number will be between 0 and $h - 1$, inclusive. All fire walls will be at least one unit long. The HeloPak and LeaperLad will never start on the same disk, nor will either start on a disk that is on a firewall. There will always be a way for LeaperLad to reach his HeloPak.

There will be no more than 50 problems.

Output

For each input line, print a single integer indicating the minimal number of moves needed for LeaperLad to reach his HeloPack. Pay close attention to the first couple of examples; they clarify how moves are counted.

Examples

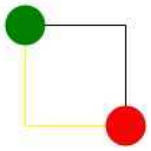
Example 1

```
2 1 0 0 1 0 0
```

This requires two moves. In the first move, LeaperLad accelerates one unit in the positive x direction, and hops onto the destination disk. In the second move, he decelerates to the required speed of zero (note that although LeaperLad's position does not change during the second move, it nonetheless counts towards the total).

Example 2

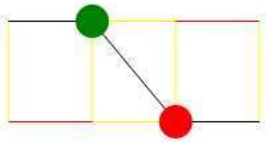
```
2 2 0 0 1 1 0
```



This requires four moves. LeaperLad first moves to the right, as in the prior example, but he must decelerate in the x direction first, then accelerate in the y direction to jump down, then decelerate again to become motionless.

Example 3

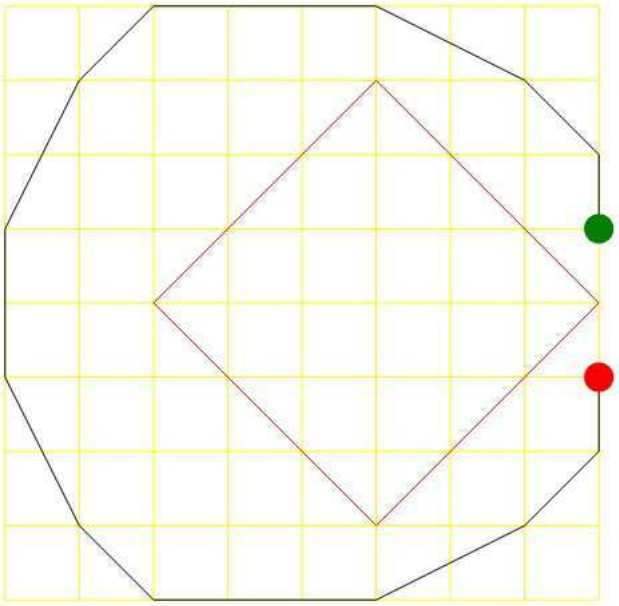
```
4 2 1 0 2 1 2 0 1 1 1 2 0 3 0
```



This requires eight moves. LeaperLad cannot jump diagonally from a standing stop; he needs to back up to get some momentum first. So first he must move to position $(0, 0)$, then decelerate to turn around, then accelerate in the x direction while jumping to his original location. His momentum allows him to accelerate in the y direction to make a diagnoal move. Once landing at his destination, he has one unit of momentum in both x and y directions, so he must decelerate, first in the y direction (which takes him to position $(3, 1)$, overshooting his destination). Then he decelerates to turn around, jumps to his destination once more, and then decelerates to be motionless, for a total of eight moves.

Example 4

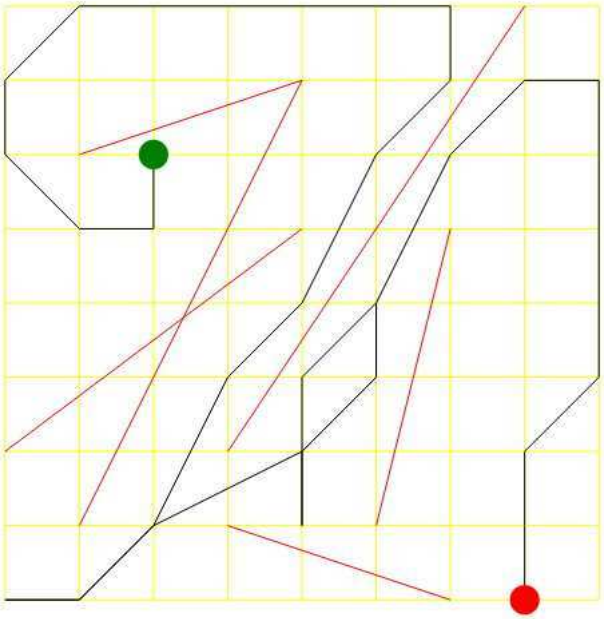
```
9 9 8 3 8 5 4 8 4 5 1 5 1 2 4 2 4 5 7 5 7 8 4
```



This requires 16 moves.

Example 5

```
9 9 2 2 7 8 6 0 6 4 3 6 8 3 7 1 7 4 1 3 6 7 0 4 1 1 2 5 7 6 3
```



This requires 43 moves.

Sample Input	Sample Output
2 1 0 0 1 0 0 2 2 0 0 1 1 0 4 2 1 0 2 1 2 0 1 1 1 2 0 3 0 9 9 8 3 8 5 4 8 4 5 1 5 1 2 4 2 4 5 7 5 7 8 4 9 9 2 2 7 8 6 0 6 4 3 6 8 3 7 1 7 4 1 3 6 7 0 4 1 1 2 5 7 6 3	2 4 8 16 43

2724 - Eurodiffusion

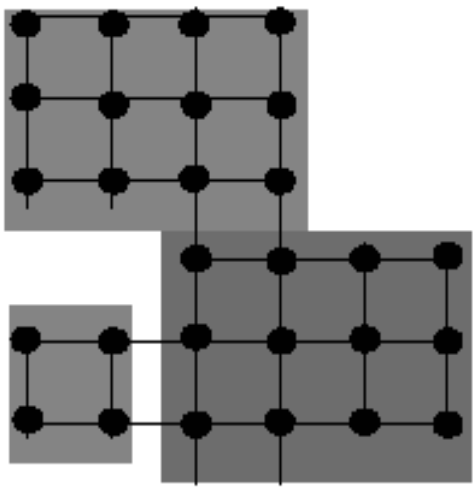
World Finals - Beverly Hills - 2002/2003

On January 1, 2002, twelve European countries abandoned their national currency for a new currency, the euro. No more francs, marks, lires, guldens, kroner,... only euros, all over the eurozone. The same banknotes are used in all countries. And the same coins? Well, not quite. Each country has limited freedom to create its own euro coins:

> ``Every euro coin carries a common European face. On the obverse, member states decorate the coins with their own motif. No matter which motif is on the coin, it can be used anywhere in the 12 Member States. For example, a French citizen is able to buy a hot dog in Berlin using a euro coin with the imprint of the King of Spain." (source: http://europa.eu.int/euro/html/entry.html)

On January 1, 2002, the only euro coins available in Paris were French coins. Soon the first non-French coins appeared in Paris. Eventually, one may expect all types of coins to be evenly distributed over the twelve participating countries. (Actually this will not be true. All countries continue minting and distributing coins with their own motifs. So even in a stable situation, there should be an excess of German coins in Berlin.) So, how long will it be before the first Finnish or Irish coins are in circulation in the south of Italy? How long will it be before coins of each motif are available everywhere?

You must write a program to simulate the dissemination of euro coins throughout Europe, using a highly simplified model. Restrict your attention to a single euro denomination. Represent European cities as points in a rectangular grid. Each city may have up to 4 neighbors (one to the north, east, south and west). Each city belongs to a country, and a country is a rectangular part of the plane. The figure below shows a map with 3 countries and 28 cities. The graph of countries is connected, but countries may border holes that represent seas, or non-euro countries such as Switzerland or Denmark. Initially, each city has one million (1000000) coins in its country's motif. Every day a representative portion of coins, based on the city's beginning day balance, is transported to each neighbor of the city. A representative portion is defined as one coin for every full 1000 coins of a motif.



A city is *complete* when at least one coin of each motif is present in that city. A country is *complete* when all of its cities are complete. Your program must determine the time required for each country to become

complete.

Input

The input consists of several test cases. The first line of each test case is the number of countries ($1 \leq c \leq 20$). The next c lines describe each country. The country description has the format: *name* x_l y_l x_h y_h,

where *name* is a single word with at most 25 characters; x_l, y_l are the lower left city coordinates of that country (most southwestward city) and x_h, y_h are the upper right city coordinates of that country (most northeastward city). $1 \leq x_l \leq xh \leq 10$ and $1 \leq y_l \leq y_h \leq 10$.

The last case in the input is followed by a single zero.

Output

For each test case, print a line indicating the case number, followed by a line for each country with the country name and number of days for that country to become complete. Order the countries by days to completion. If two countries have identical days to completion, order them alphabetically by name.

Use the output format shown in the example.

Sample Input

```
3
France  1 4 4 6
Spain          3 1 6 3
Portugal    1 1 2 2
1
Luxembourg  1 1 1 1
2
Netherlands 1 3 2 4
Belgium     1 1 2 2
0
```

Sample Output

```
Case Number 1
   Spain   382
   Portugal   416
   France   1325
Case Number 2
   Luxembourg   0
Case Number 3
   Belgium   2
   Netherlands   2
```

Beverly Hills 2002-2003

Problem D: I Conduit!

Irv Kenneth Diggit works for a company that excavates trenches, digs holes and generally tears up people's yards. Irv's job is to make sure that no underground pipe or cable is underneath where excavation is planned. He has several different maps, one for each utility company, showing where their conduits lie, and he needs to draw one large, consolidated map combining them all. One approach would be to simply draw each of the smaller maps one at a time onto the large map. However, this often wastes time, not to mention ink for the pen-plotter in the office, since in many cases portions of the conduits overlap with each other (albeit at different depths underground). What Irv wants is a way to determine the minimum number of line segments to draw given all the line segments from the separate maps.

Input

Input will consist of multiple input sets. Each set will start with a single line containing a positive integer n indicating the total number of line segments from all the smaller maps. Each of the next n lines will contain a description of one segment in the format

x_1 y_1 x_2 y_2

where (x_1, y_1) are the coordinates of one endpoint and (x_2, y_2) are the coordinates of the other. Coordinate values are floating point values in the range $0 \ldots 1000$ specified to at most two decimal places. The maximum number of line segments will be 10000 and all segments will have non-zero length. Following the last input set there will be a line containing a 0 indicating end of input; it should not be processed.

Output

For each input set, output on a single line the minimum number of line segments that need to be drawn on the larger, consolidated map.

Sample Input

```
3
1.0 10.0 3.0 14.0
0.0 0.0 20.0 20.0
10.0 28.0 2.0 12.0
2
0.0 0.0 1.0 1.0
1.0 1.0 2.15 2.15
2
0.0 0.0 1.0 1.0
1.0 1.0 2.15 2.16
0
```

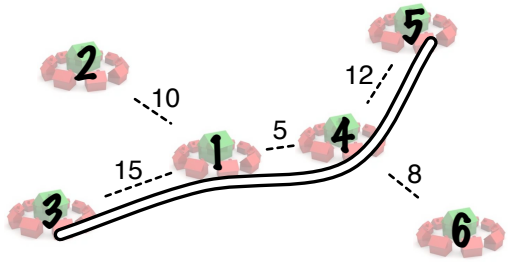
Sample Output

```
2
1
2
```

H Highway Construction (`highway.{c,cc,java}`)

As head of the Accessible Commuting Movement (ACM), you've been lobbying the mayor to build a new highway in your city. Today is your lucky day, because your request was approved. There is one condition though: *You* must provide the plan for the best highway artery to construct, or else it's not going to happen!

You have a map that shows all communities in your city, each with a unique number, where you may place highway on-ramps. On the map are a set of roadways between pairs of communities, labelled with driving distances, which you may choose to replace with your highway line. Using this network of roadways, there is exactly one route from any one community to another. In other words, there are no two different sets of roadways that would lead you from community A to community B.



You can build a single highway that runs back and forth between any two communities of your choosing. It will replace the unique set of roadways between those two communities, and an on-ramp will be built at every community along the way. Of course, residents of communities that will not have an on-ramp will have to drive to the nearest one that does in order to access your new highway.

You know that long commutes are very undesirable, so you are going to build the highway so that longest drive from any community to the nearest on-ramp is minimized. Given a map of your city with the roadways and driving distances, what is the farthest distance from any community that someone would have to drive to get to the nearest on-ramp once your new highway is complete?

H.1 Input

The input consists of multiple test cases. Each test case is a description of a city map, and begins with a single line containing an integer N ($2 \leq N \leq 100{,}000$), the number of communities in the city. Then $N-1$ lines follow, each containing three integers, i, j ($1 \leq i, j \leq n$), and d ($1 \leq d \leq 10{,}000$). Each line indicates that communities i and j are connected by a roadway with driving distance d. Input is followed by a single line with $N = 0$, which should not be processed. For example:

```
6
2 1 10
3 1 15
1 4 5
4 5 12
4 6 8
0
```

H.2 Output

For each city map, output on a single line the farthest distance from any community to the nearest on-ramp of the new highway. For example:

```
10
```