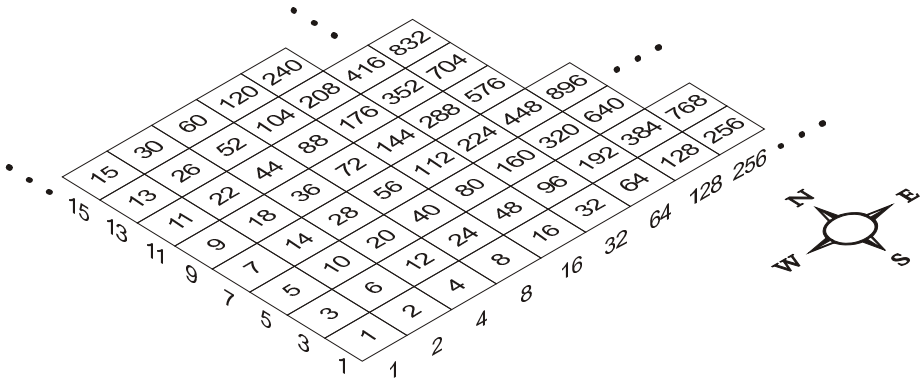
[C] City of Flatland

Program:	flatland.(c|cpp|java)
Input:	flatland.in
Output:	flatland.out

Description

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 $(1, 2, 4, 8, \ldots)$ and all its East-West streets using odd numbers $(1, 3, 5, \ldots)$. 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 Format

The input is made of one or more pairs of building numbers. Each pair `<S,T>` 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 `<0,0>` (which is not part of the test cases.)

Output Format

For each input pair `<S,T>`, the output file should include a line of the form:

`The distance between S and T is D.`

The output file should be in the same order as the input file.

Sample Input/Output

flatland.in

```
12 14
20 30
40 50
0 0
```

flatland.out

```
The distance between 12 and 14 is 3.
The distance between 20 and 30 is 6.
The distance between 40 and 50 is 12.
```

The 2001 25th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem B

Say Cheese

Input: cheese.in

Once upon a time, in a giant piece of cheese, there lived a cheese mite named Amelia Cheese Mite. Amelia should have been truly happy because she was surrounded by more delicious cheese than she could ever eat. Nevertheless, she felt that something was missing from her life.

One morning, her dreams about cheese were interrupted by a noise she had never heard before. But she immediately realized what it was — the sound of a male cheese mite, gnawing in the same piece of cheese! (Determining the gender of a cheese mite just by the sound of its gnawing is by no means easy, but all cheese mites can do it. That's because their parents could.)

Nothing could stop Amelia now. She had to meet that other mite as soon as possible. Therefore she had to find the fastest way to get to the other mite. Amelia can gnaw through one millimeter of cheese in ten seconds. But it turns out that the direct way to the other mite might not be the fastest one. The cheese that Amelia lives in is full of holes. These holes, which are bubbles of air trapped in the cheese, are spherical for the most part. But occasionally these spherical holes overlap, creating compound holes of all kinds of shapes. Passing through a hole in the cheese takes Amelia essentially zero time, since she can fly from one end to the other instantly. So it might be useful to travel through holes to get to the other mite quickly.

For this problem, you have to write a program that, given the locations of both mites and the holes in the cheese, determines the minimal time it takes Amelia to reach the other mite. For the purposes of this problem, you can assume that the cheese is infinitely large. This is because the cheese is so large that it never pays for Amelia to leave the cheese to reach the other mite (especially since cheese-mite eaters might eat her). You can also assume that the other mite is eagerly anticipating Amelia's arrival and will not move while Amelia is underway.

Input

The input file contains descriptions of several cheese mite test cases. Each test case starts with a line containing a single integer n ($0 \le n \le 100$), the number of holes in the cheese. This is followed by n lines containing four integers x_i, y_i, z_i, r_i each. These describe the centers (x_i, y_i, z_i) and radii r_i ($r_i > 0$) of the holes. All values here (and in the following) are given in millimeters.

The description concludes with two lines containing three integers each. The first line contains the values x_A, y_A, z_A, giving Amelia's position in the cheese, the second line containing x_O, y_O, z_O, gives the position of the other mite.

The input file is terminated by a line containing the number –1.

Output

For each test case, print one line of output, following the format of the sample output. First print the number of the test case (starting with 1). Then print the minimum time in seconds it takes Amelia to reach the other mite, rounded to the closest integer. The input will be such that the rounding is unambiguous.

Sample Input	Output for the Sample Input
1 20 20 20 1 0 0 0 0 0 10 1 5 0 0 4 0 0 0 10 0 0 -1	Cheese 1: Travel time = 100 sec Cheese 2: Travel time = 20 sec

The 2003 27th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem D

Eurodiffusion

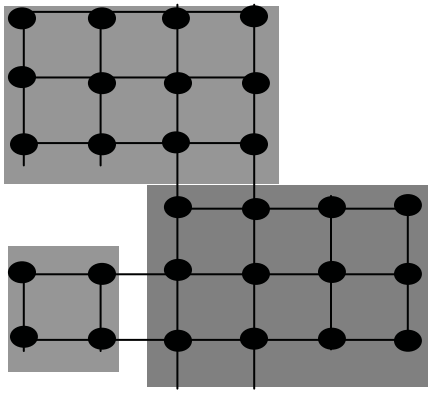
Input File: euro.in

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 ($1 \le c \le 20$). The next c lines describe each country. The country description has the format: *name xl yl xh yh*, where *name* is a single word with at most 25 characters; *xl*, *yl* are the lower left city coordinates of that country (most southwestward city) and *xh, yh* are the upper right city coordinates of that country (most northeastward city). $1 \le xl \le xh \le 10$ and $1 \le yl \le yh \le 10$.

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	Output for the 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	Case Number 1 Spain 382 Portugal 416 France 1325 Case Number 2 Luxembourg 0 Case Number 3 Belgium 2 Netherlands 2

Problem D

Number Game

Source: `numbergame.(c|cc|pas|java)`
Input: `numbergame.in`

Christine and Matt are playing an exciting game they just invented: the Number Game. The rules of this game are as follows.

The players take turns choosing integers greater than 1. First, Christine chooses a number, then Matt chooses a number, then Christine again, and so on. The following rules restrict how new numbers may be chosen by the two players:

- A number which has already been selected by Christine or Matt, or a multiple of such a number, cannot be chosen.
- A sum of such multiples cannot be chosen, either.

If a player cannot choose any new number according to these rules, then that player loses the game.

Here is an example: Christine starts by choosing 4. This prevents Matt from choosing 4, 8, 12, etc. Let's assume that his move is 3. Now the numbers 3, 6, 9, etc. are excluded, too; furthermore, numbers like: $7 = 3+4, 10 = 2\cdot 3+4, 11 = 3+2\cdot 4, 13 = 3\cdot 3+4, \ldots$ are also not available. So, in fact, the only numbers left are 2 and 5. Christine now selects 2. Since $5 = 2+3$ is now forbidden, she wins because there is no number left for Matt to choose.

Your task is to write a program which will help play (and win!) the Number Game. Of course, there might be an infinite number of choices for a player, so it may not be easy to find the best move among these possibilities. But after playing for some time, the number of remaining choices becomes finite, and that is the point where your program can help. Given a game position (a list of numbers which are not yet forbidden), your program should output all *winning moves*.

A winning move is a move by which the player who is about to move can force a win, no matter what the other player will do afterwards. More formally, a winning move can be defined as follows.

- A winning move is a move after which the game position is a losing position.
- A winning position is a position in which a winning move exists. A losing position is a position in which no winning move exists.
- In particular, the position in which all numbers are forbidden is a losing position. (This makes sense since the player who would have to move in that case loses the game.)

Input

The input file consists of several test cases. Each test case is given by exactly one line describing one position.

Each line will start with a number n ($1 \le n \le 20$), the number of integers which are still available. The remainder of this line contains the list of these numbers $a_1, \ldots, a_n$ ($2 \le a_i \le 20$).

The positions described in this way will always be positions which can really occur in the actual Number Game. For example, if 3 is not in the list of allowed numbers, 6 is not in the list, either.

At the end of the input file, there will be a line containing only a zero (instead of n); this line should not be processed.

Output

For each test case, your program should output "`Test case #`*m*", where *m* is the number of the test case (starting with 1). Follow this by either "`There's no winning move.`" if this is true for the position described in the input file, or "`The winning moves are:` w_1 w_2 ... w_k" where the w_i are all winning moves in this position, satisfying $w_i < w_{i+1}$ for $1 \leq i < k$. After this line, output a blank line.

Sample Input

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

Sample Output

```
Test Case #1
The winning moves are: 2

Test Case #2
There's no winning move.

Test Case #3
The winning moves are: 4 5 6
```

The 1998 22nd Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

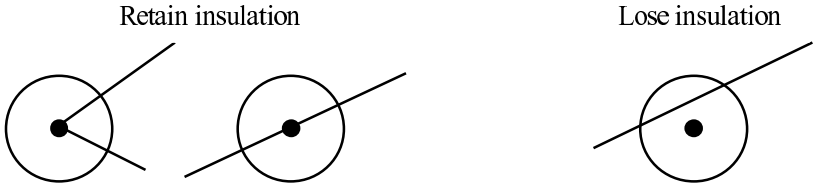
Problem A

Crystal Clear

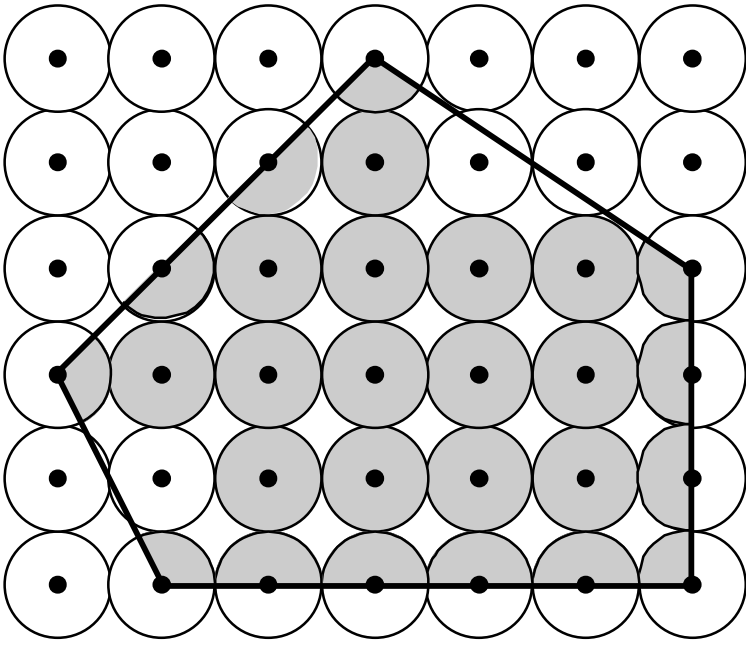
Input: crystal.in

A new high technology company has developed a material that it hopes to market as an insulator. The material consists of crystals and the square lattice on which the crystals are grown. The points on the lattice are at 1 centimeter intervals. The crystals are formed from seeds that are planted at the lattice points. Each crystal grows into a circle of diameter 1 centimeter.

Using this material in applications will require cutting the lattice into pieces. One of the problems in cutting the lattice is that some crystals will be sliced in the process. Slicing a crystal other than through the center completely destroys that crystal's insulation properties. (A cut touching a crystal tangentially does not destroy that crystal's insulation property.)



The insulation capacity of a piece is directly proportional to the total area of the insulating crystals (or parts of crystals) that are on the piece. The following figure shows a polygonal piece with its insulating crystals shaded.



Your job is to determine the insulating capacity of such polygonal pieces by computing the total area of the insulating crystals in it.

Input

The input consists of a sequence of polygon descriptions. Each description consists of a positive integer n ($3 \le n \le 25$) representing the number of vertices, followed by n pairs of integers. Each pair is the x and y coordinates of one vertex of the polygon. (The coordinate system is aligned with the lattice such that the integer coordinates are precisely the lattice points.)

Vertices of each polygon are given in clockwise order. No polygon will be degenerate. No coordinate will be larger than 250 in absolute value.

The input is terminated by zero for the value of n.

Output

For each polygon, first print its number ("`Shape 1`", "`Shape 2`", etc.) and then the area of the insulating crystals in cm^2, exact to three digits to the right of the decimal point.

The following sample corresponds to the previous illustration.

Sample Input

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

Output for the Sample Input

```
Shape 1
Insulating area = 15.315 cm^2
```

Problem A

Bridging signals

*Source code: signals.**

'Oh no, they've done it again', cries the chief designer at the Waferland chip factory. Once more the routing designers have screwed up completely, making the signals on the chip connecting the ports of two functional blocks cross each other all over the place. At this late stage of the process, it is too expensive to redo the routing. Instead, the engineers have to bridge the signals, using the third dimension, so that no two signals cross. However, bridging is a complicated operation, and thus it is desirable to bridge as few signals as possible. The call for a computer program that finds the maximum number of signals which may be connected on the silicon surface without crossing each other, is imminent. Bearing in mind that there may be thousands of signal ports at the boundary of a functional block, the problem asks quite a lot of the programmer. Are you up to the task?

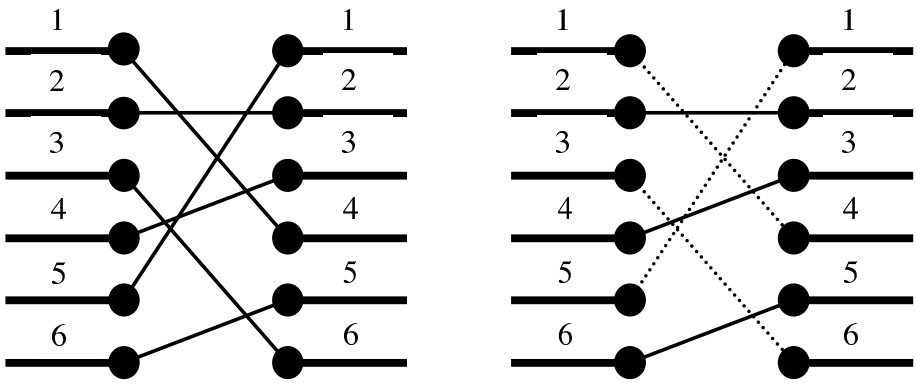


Figure 1. To the left: The two blocks' ports and their signal mapping (4,2,6,3,1,5). To the right: At most three signals may be routed on the silicon surface without crossing each other. The dashed signals must be bridged.

A typical situation is schematically depicted in figure 1. The ports of the two functional blocks are numbered from 1 to p, from top to bottom. The signal mapping is described by a permutation of the numbers 1 to p in the form of a list of p unique numbers in the range 1 to p, in which the i:th number specifies which port on the right side should be connected to the i:th port on the left side. Two signals cross if and only if the straight lines connecting the two ports of each pair do.

Input

On the first line of the input, there is a single positive integer n, telling the number of test scenarios to follow. Each test scenario begins with a line containing a single positive integer p<40000, the number of ports on the two functional blocks. Then follow p lines, describing the signal mapping: On the i:th line is the port number of the block on the right side which should be connected to the i:th port of the block on the left side.

Output

For each test scenario, output one line containing the maximum number of signals which may be routed on the silicon surface without crossing each other.

Example input:

```
4
6
4
2
```

Problem A, cont.

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

Example output:

```
3
9
1
4
```