UW Madison's

2006 ACM-ICPC Team Test #2

November 5, 1:00-6:00pm

Overview:
This test consists of ten (!) problems, which will be referred to by the following names (respective of order):
add, art, budget, causal, dusk, gunman, hole, product, skyline, unix
As this is a team competition, you are allowed one computer for each team. Also, you may only use 25 pages of printed documentation in addition to the online STL and Java documentation.

Input/Output:
Your programs should take input from standard in (the keyboard) and output to standard out (the terminal). As is standard practice for the ICPC, you may assume that the input strictly follows the description in the problems. If a problem description does not indicate how the input will be terminated, you may assume it will be with the EOF character. It is your responsibility to ensure that your output **precisely** matches that described in the problems, or else risk your program being rejected with a “Presentation Error”.

Problem Submission:
To submit a program, send e-mail to mwa+icpc@cs.wisc.edu and attach your source code. The subject line should contain only the problem name, **prob,** that you are submitting, and the attached source code should be named **prob**.{c|cpp|java}. For example, if you are using C++ and submitting the problem “add”, the file should be named “add.cpp”. You will receive the results of your submission as soon as possible via e-mail. You may submit from any e-mail account of a team member, but please try to be consistent and use only one throughout the contest.

Clarifications:
As in the ICPC, you may submit clarification requests as well. They should be sent to mwa+icpc@cs.wisc.edu, with a subject of “Clarification-**prob**”, where **prob** is the name of the problem you wish to be clarified. Replace **prob** with “general” if there is an issue with the contest as a whole. Accepted clarification requests will be answered to all those taking the test via e-mail. Experience of previous years learns that most clarification requests are rejected, and receive a simple response such as “Read the problem description”.

Printing:
You may print to the printer at any time during the test.

GOOD LUCK!

Problem F

Add All

Input: standard input
Output: standard output

Yup!! The problem name reflects your task; just add a set of numbers. You may feel yourselves insulted by this menial task, so let's make it more interesting.

The addition operation now has a cost, and the cost of an addition is the sum of the two numbers to be added. So, to add **1** and **10**, you incur a cost of **11**. If you want to add **1**, **2** and **3**. There are several ways –

1 + 2 = 3, cost = 3 3 + 3 = 6, cost = 6 Total = 9	1 + 3 = 4, cost = 4 2 + 4 = 6, cost = 6 Total = 10	2 + 3 = 5, cost = 5 1 + 5 = 6, cost = 6 Total = 11

I hope you already understand your mission: add a set of integers so that the total cost is minimal.

Input

Each test case will start with a positive number, **N (2 ≤ N ≤ 5000)** followed by **N** positive integers (all are less than **100000**). Input is terminated by a case where the value of **N** is zero. This case should not be processed.

Output

For each case print the minimum total cost of addition in a single line.

Sample Input

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

Output for Sample Input

```
9
19
```

Problem setter: Md. Kamruzzaman, EPS

Modern Art

The famous painter Mel Borp is working on a brilliant series of paintings that introduce a new experimental style of Modern Art. At first glance, these paintings look deceptively simple, since they consist only of triangles of different sizes that seem to be stacked on top of each other. Painting these works, however, takes an astonishing amount of consideration, calculation, and precision since all triangles are painted without taking the brush off the canvas. How exactly Mel paints his works is a well-kept secret.

Recently, he started on the first painting of his new series. It was a single triangle, titled $T_{0,0}$. After that, he created $T_{1,1}$, the basis for his other works (see Figure 1).

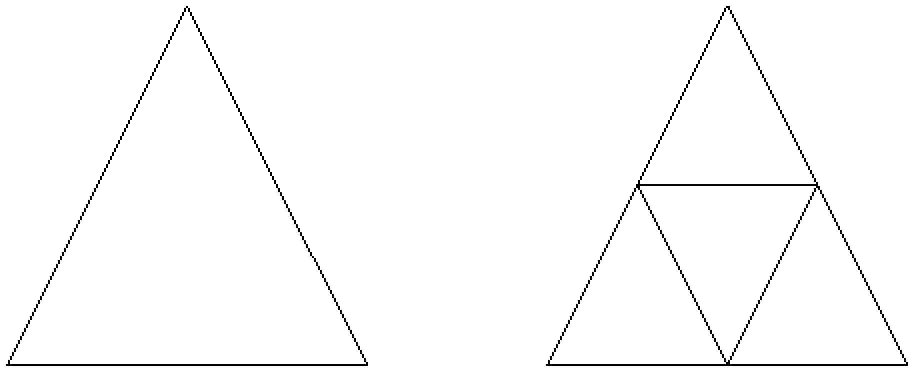


Figure: Early work: $T_{0,0}$ (left) and $T_{1,1}$ (right).

Then he decided to take his experimenting one step further, and he painted $T_{1,2}$ and $T_{3,2}$. Compare Figure 1 and Figure 2 to fully appreciate the remarkable progression in his work.

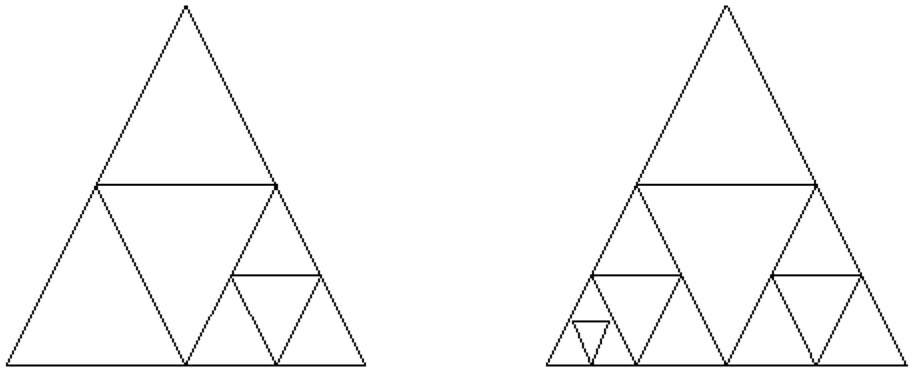


Figure: Advanced work: $T_{1,2}$ (left) and $T_{3,2}$ (right).

Note that the shape of the painting can be deduced from its title, $T_{p,q}$, as follows:

- $T_{0,0}$ is a single triangle (see Figure 1);
- $T_{1,1}$ consists of a smaller, inverted triangle placed inside $T_{0,0}$, so that the result consists of four smaller triangles (see Figure 1);
- if $p > 0$ and $q > 1$, then the painting looks like $T_{p,q-1}$, except that an inverted triangle has been placed inside the bottom-right triangle of $T_{p,q-1}$, splitting it into four smaller triangles (compare for example $T_{1,1}$ and $T_{1,2}$);

- if $p > 1$ and $q > 0$, the painting looks like $T_{p-1,q}$, except that an inverted triangle has been placed inside the bottom-left triangle of $T_{p-1,q}$, splitting it into four smaller triangles;
- other values of p and q: it is not a valid title of a painting.

The triangles of a painting look all the same (each triangle is an isosceles triangle with two sides of the same length), but their height and width depend on the size of the canvas Mel used.

Mel wanted to end the series with $T_{10,10}$, the most complex painting he thought he would be able to paint.

But no matter how many times he tried, he could not get it right. Now he is desperate, and he hopes you can help him by writing a program that prints, in order, the starting and ending coordinates of the lines Mel has to paint. Of course, you will need to know how Mel paints his works, so we will now reveal his secret technique.

As an example, take a look at $T_{1,2}$ (see Figure 3):

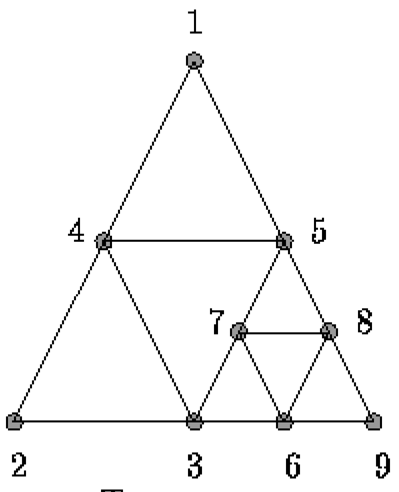


Figure: $T_{1,2}$.

Mel always starts at the top of the top triangle, drawing a line straight to the lower-left corner of the lower left triangle (in this example, 1-2), continuing with a line to the lower-right corner of that triangle (in this example, 2-3). Next, he works his way up by drawing a line to the top of that triangle (in this example, 3-4). Now he has either reached the starting point again (finishing yet another masterpiece) or he has reached the lower-left corner of another triangle (in this example, 1-4-5). In the latter case, he continues by drawing the bottom line of that triangle (4-5) and after that he starts working on the triangle or triangles that is or are located underneath the lower-right corner of that triangle, in the same way. So he continues with (5-3), (3-6), (6-7), (7-8), (8-6), (6-9), and (9-1) as the finishing touch.

Input Specification

The first line of the input contains the number of test cases. Each test case consists of one line containing four non-negative integers p, q, x, and y, separated by spaces. $T_{p,q}$ is the title of the painting and (x,y) are the coordinates of the top of the top triangle. Further, $p, q \leq 10$ and $x,y < 32768$. All triangles have a nonzero area.

Output Specification

For every test case, the output contains the pairs of (x,y) integer coordinates of the starting and ending points of all lines Mel has to draw for the painting $T_{p,q}$ in the right order, in the format

$$(startX, startY)(endX, endY)$$

followed by a newline. The output for each test case must be followed by an empty line.

Sample Input

```
2
0 0 1 1
1 2 512 1024
```

Sample Output

```
(1,1)(0,0)
(0,0)(2,0)
(2,0)(1,1)

(512,1024)(0,0)
(0,0)(512,0)
(512,0)(256,512)
(256,512)(768,512)
(768,512)(512,0)
(512,0)(768,0)
(768,0)(640,256)
(640,256)(896,256)
(896,256)(768,0)
(768,0)(1024,0)
(1024,0)(512,1024)
```

Budget Travel

An American travel agency is sometimes asked to estimate the minimum cost of traveling from one city to another by automobile. The travel agency maintains lists of many of the gasoline stations along the popular routes. The list contains the location and the current price per gallon of gasoline for each station on the list.

In order to simplify the process of estimating this cost, the agency uses the following rules of thumb about the behavior of automobile drivers.

- A driver never stops at a gasoline station when the gasoline tank contains more than half of its capacity unless the car cannot get to the following station (if there is one) or the destination with the amount of gasoline in the tank.
- A driver always fills the gasoline tank completely at every gasoline station stop.
- When stopped at a gasoline station, a driver will spend $2.00 on snacks and goodies for the trip.
- A driver needs no more gasoline than necessary to reach a gasoline station or the city limits of the destination. There is no need for a ``safety margin."
- A driver always begins with a full tank of gasoline.
- The amount paid at each stop is rounded to the nearest cent (where 100 cents make a dollar).

You must write a program that estimates the minimum amount of money that a driver will pay for gasoline and snacks to make the trip.

Input

Program input will consist of several data sets corresponding to different trips. Each data set consists of several lines of information. The first 2 lines give information about the origin and destination. The remaining lines of the data set represent the gasoline stations along the route, with one line per gasoline station. The following shows the exact format and meaning of the input data for a single data set.

Line 1:
One real number - the distance from the origin to the destination

Line 2:
Three real numbers followed by an integer

- The first real number is the gallon capacity of the automobile's fuel tank.
- The second is the miles per gallon that the automobile can travel.
- The third is the cost in dollars of filling the automobiles tank in the origination city.
- The integer (less than 51) is the number of gasoline stations along the route.

Each remaining line:
Two real numbers

- The first is the distance in miles from the origination city to the gasoline station.
- The second is the price (in cents) per gallon of gasoline sold at that station.

All data for a single data set are positive. Gasoline stations along a route are arranged in nondescending order of distance from the origin. No gasoline station along the route is further from the origin than the distance from the origin to the destination. There are always enough stations appropriately placed along the each route for any car to be able to get from the origin to the destination.

The end of data is indicated by a line containing a single negative number.

Output

For each input data set, your program must print the data set number and a message indicating the minimum total cost of the gasoline and snacks rounded to the nearest cent. That total cost must include the initial cost of filling the tank at the origin. Sample input data for 2 separate data sets and the corresponding correct output follows.

Sample Input

```
475.6
11.9 27.4 14.98 6
102.0 99.9
220.0 132.9
256.3 147.9
275.0 102.9
277.6 112.9
381.8 100.9
516.3
15.7 22.1 20.87 3
125.4 125.9
297.9 112.9
345.2 99.9
-1
```

Sample Output

```
Data Set #1
minimum cost = $27.31
Data Set #2
minimum cost = $38.09
```

Problem B: Advanced Causal Measurements (ACM)

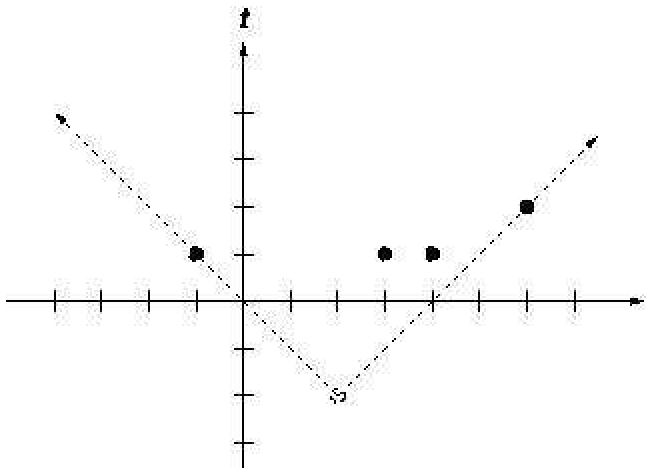
Causality is a very important concept in theoretical physics. The basic elements in a discussion of causality are *events*. An event *e* is described by its time of occurrence *t*, and its location, ***x***, and we write *e* = (*t*,***x***). For our concerns, all events happen in the one dimensional geometric space and thus locations are given by a single real number *x* as a coordinate on *x*-axis. Usually, theoretical physicists like to define the speed of light to be 1, so that time and space have the same units (actual physical units frighten and confuse theorists).

One event $e_1 = (t_1,x_1)$ is a *possible cause* for a second event $e_2 = (t_2,x_2)$ if a signal emitted at e_1 could arrive at e_2. Signals can't travel faster than the speed of light, so this condition can be stated as:

$$e_1 \text{ is a } \textit{possible cause} \text{ for } e_2 \textbf{ iff } t_2 >= t_1+|x_2-x_1|$$

Thus an event at (-1,1) could cause events at (0,0), (1,2), and (1,3), for example, but could not have caused events at (1,4) or (-2,1). Note that one event can cause several others.

Recently, scientists have observed several unusual events in the geometrically one dimensional universe, and using current theories, they know how many causes were responsible for these observations, but they know nothing about the time and space coordinates of the causes. You asked to write a program to determine the latest time at which the earliest cause could have occurred (i.e. the time such that at least one cause must have occurred on or before this time). Somewhat surprisingly, all the observed events have both space and time coordinates expressed by integer numbers in the range $-1000000 \le t, x \le 1000000$.



The figure on the right illustrates the first case from input: the earliest single event as a possible cause of all four events.

The first line of input is the number of cases which follow. Each case begins with a line containing the number *n* of events and the number *m* of causes, $1 \le n, m \le 100000$. Next follows *n* lines containing the *t* and *x* coordinates for each event.

Output consists of a single line for each case in the format as in the sample output, giving the latest time at which the earliest cause could have occurred, this will be an integer as our time units are not divisible.

Sample Input

```
4
4 1
1 -1
1 3
1 4
2 6
4 2
1 -1
1 3
1 4
2 6
4 3
1 -1
1 3
1 4
```

```
2 6
4 4
1 -1
1 3
1 4
2 6
```

Output for Sample Input

```
Case 1: -2
Case 2: 0
Case 3: 0
Case 4: 1
```

Daniel Robbins

Problem F: From Dusk till Dawn

Vladimir has white skin, very long teeth and is 600 years old, but this is no problem because Vladimir is a vampire.
Vladimir has never had any problems with being a vampire. In fact, he is a very successful doctor who always takes the night shift and so has made many friends among his colleagues. He has a very impressive trick which he shows at dinner partys: He can tell tell blood group by taste.
Vladimir loves to travel, but being a vampire he has to overcome three problems.

- First, he can only travel by train because he has to take his coffin with him. (On the up side he can always travel first class because he has invested a lot of money in long term stocks.)
- Second, he can only travel from dusk till dawn, namely from 6 pm to 6 am. During the day he has to stay inside a train station.
- Third, he has to take something to eat with him. He needs one litre of blood per day, which he drinks at noon (12:00) inside his coffin.

You should help Vladimir to find the shortest route between two given cities, so that he can travel with the minimum amount of blood. (If he takes too much with him, people will ask funny questions like "What do you do with all that blood?")

Input Specification

The first line of the input will contain a single number telling you the number of test cases.
Each test case specification begins with a single number telling you how many route specifications follow.
Each route specification consists of the names of two cities, the departure time from city one and the total travelling time. The times are in hours. Note that Vladimir can't use routes departing earlier than 18:00 or arriving later than 6:00.
There will be at most 100 cities and less than 1000 connections. No route takes less than one hour and more than 24 hours. (Note that Vladimir can use only routes with a maximum of 12 hours travel time (from dusk till dawn).) All city names are shorter than 32 characters.
The last line contains two city names. The first is Vladimir's start city, the second is Vladimir's destination.

Output Specification

For each test case you should output the number of the test case followed by "`Vladimir needs # litre(s) of blood.`" or "`There is no route Vladimir can take.`"

Sample Input

```
2
3
Ulm Muenchen 17 2
Ulm Muenchen 19 12
Ulm Muenchen 5 2
Ulm Muenchen
10
Lugoj Sibiu 12 6
Lugoj Sibiu 18 6
Lugoj Sibiu 24 5
Lugoj Medias 22 8
Lugoj Medias 18 8
Lugoj Reghin 17 4
Sibiu Reghin 19 9
Sibiu Medias 20 3
Reghin Medias 20 4
Reghin Bacau 24 6
```

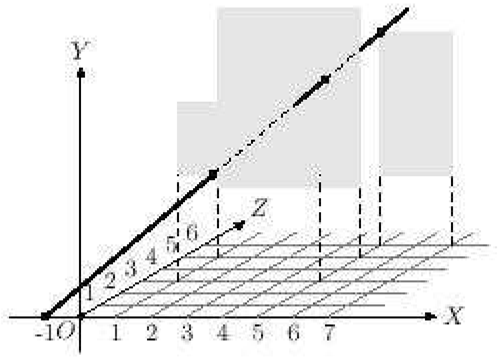
```
Lugoj Bacau
```

Sample Output

```
Test Case 1.
There is no route Vladimir can take.
Test Case 2.
Vladimir needs 2 litre(s) of blood.
```

	# 3219 - Gunman Europe - Northeastern Europe & Russian Republic - 2004/2005				
	PDF	PostScript			

Consider a 3D scene with *OXYZ* coordinate system. Axis *OX* points to the right, axis *OY* points up, and axis *OZ* points away from you. There is a number of rectangular windows on the scene. The plane of each window is parallel to *OXY*, its sides are parallel to *OX* and *OY*. All windows are situated at different depths on the scene (different coordinates $z > 0$).



A gunman with a rifle moves along *OX* axis ($y = 0$ and $z = 0$). He can shoot a bullet in a straight line. His goal is to shoot a single bullet through all the windows. Just touching a window edge is enough. Your task is to determine how to make such shot.

Input

Input file consists of several test cases. The first line of each case contains a single integer number n ($2 \leq n \leq 100$) -- the number of windows on the scene. The following n lines describe the windows. Each line contains five integer numbers $x_{1i}, y_{1i}, x_{2i}, y_{2i}, z_i$ ($0 < x_{1i}, y_{1i}, x_{2i}, y_{2i}, z_i < 1000$). Here (x_{1i}, y_{1i}, z_i) are coordinates of the bottom left corner of the window, and (x_{2i}, y_{2i}, z_i) are coordinates of the top right corner of the window ($x_{1i} < x_{2i}, y_{1i} < y_{2i}$). Windows are ordered by z coordinate ($z_i > z_{i-1}$ for $2 \leq i \leq n$).

Output

For each test case, the output must be as follows: Write the single word `UNSOLVABLE' if the gunman cannot reach the goal of shooting a bullet through all the windows. Otherwise, on the first line output a word `SOLUTION'. On the next line output x coordinate of the point from which the gunman must fire a bullet. On the following n lines output x, y, z coordinates of the points where the bullet goes through the consecutive windows. All coordinates in the output file must be printed with six digits after decimal point.

Separate the output for consecutive cases by a single blank line.

Sample Input

```
3
1 3 5 5 3
1 2 5 7 5
5 2 7 6 6
3
```

```
2 1 5 4 1
3 5 6 8 2
4 3 8 6 4
```

Sample Output

```
SOLUTION
-1.000000
2.000000 3.000000 3.000000
4.000000 5.000000 5.000000
5.000000 6.000000 6.000000

UNSOLVABLE
```

Northeastern Europe & Russian Republic 2004-2005

Problem F
A Hole to Catch a Man

Input : standard input
Output : standard output

How can a manhole be a hole if it is covered? Perhaps, to prove a manhole a hole, most of the manholes of Dhaka are uncovered. So now manhole means *a hole to catch a man*. Anyway, the new Mayor of Dhaka does not like this definition and he has recently been highly acclaimed by general people for ordering corresponding department to cover all the manholes of the city within a month.

Manhole Cover Manufacturing Corporation (*MCMC*) somehow managed to get the order. (Yes, this is a big deal, since a lot of manhole covers are to be made). MCMC makes the cover using steel, and they import polygonal steel sheets of different shapes and thickness from abroad. Then they melt the sheets to make the circular manhole covers, which also differ in size and thickness.

MCMC needs a program which, given dimensions of a number of steel sheets, will calculate how many manhole cover can be made from these sheets. You are to help them by writing the program.

Input

The input file consists of several data blocks.

Each data block starts with an integer *N,* the number of polygonal steel sheets. *i*'th line of the next *N* lines starts with thickness of the *i*'th sheet followed by co-ordinates of the polygons' corner points in some order (clockwise or anti-clockwise). Each line consists of a series of real numbers in following format:

$$T_i\, X_0\, Y_0\, X_1\, Y_1\, X_2\, Y_2 \ldots \ldots X_n\, Y_n\, X_0\, Y_0$$

Where T_i is the thickness of the sheet, and X_i Y_i are the coordinates of corner points. The line ends with co-ordinate of the first point. Last line of each data block will have two real numbers, *R* and *T*, radius and thickness of the manhole cover respectively.

Input file ends with a data block with $N = 0$.

Output

For each data block, print the number of manhole cover in separate line.

Sample Input:

```
2
2 0 0 0 10 5 15 12 10 10 0 0 0
5 0 0 5 100 100 0 0 0
5 3
1
2 0 0 10 0 10 10 0 10 0 0
5 2
0
```

Sample Output:

```
107
1
```

Rezaul Alam Chowdhury, Suman Kumar Nath, Tarique Mesbaul Islam.

Problem D - Maximum Product

Time Limit: 1 second

Given a sequence of integers $\mathbf{S = \{S_1, S_2, ..., S_n\}}$, you should determine what is the value of the maximum positive product involving consecutive terms of **S**. If you cannot find a positive sequence, you should consider **0** as the value of the maximum product.

Input

Each test case starts with $\mathbf{1 \le N \le 18}$, the number of elements in a sequence. Each element $\mathbf{S_i}$ is an integer such that $\mathbf{-10 \le S_i \le 10}$. Next line will have **N** integers, representing the value of each element in the sequence. There is a blank line after each test case. The input is terminated by end of file (EOF).

Output

For each test case you must print the message: **Case #M: The maximum product is P.**, where **M** is the number of the test case, starting from **1**, and **P** is the value of the maximum product. After each test case you must print a blank line.

Sample Input

```
3
2 4 -3

5
2 5 -1 2 -1
```

Sample Output

```
Case #1: The maximum product is 8.

Case #2: The maximum product is 20.
```

Problem setter: Sérgio Queiroz de Medeiros

The Skyline Problem

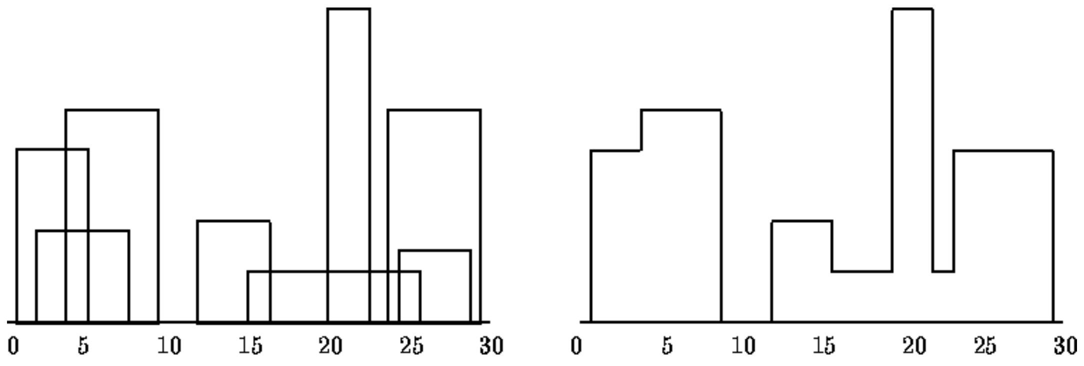
Background

With the advent of high speed graphics workstations, CAD (computer-aided design) and other areas (CAM, VLSI design) have made increasingly effective use of computers. One of the problems with drawing images is the elimination of hidden lines -- lines obscured by other parts of a drawing.

The Problem

You are to design a program to assist an architect in drawing the skyline of a city given the locations of the buildings in the city. To make the problem tractable, all buildings are rectangular in shape and they share a common bottom (the city they are built in is very flat). The city is also viewed as two-dimensional. A building is specified by an ordered triple (L_i, H_i, R_i) where L_i and R_i are left and right coordinates, respectively, of building *i* and H_i is the height of the building. In the diagram below buildings are shown on the left with triples (1,11,5), (2,6,7), (3,13,9), (12,7,16), (14,3,25), (19,18,22), (23,13,29), (24,4,28)

the skyline, shown on the right, is represented by the sequence: (1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19, 18, 22, 3, 23, 13, 29, 0)



The Input

The input is a sequence of building triples. All coordinates of buildings are positive integers less than 10,000 and there will be at least one and at most 5,000 buildings in the input file. Each building triple is on a line by itself in the input file. All integers in a triple are separated by one or more spaces. The triples will be sorted by L_i , the left *x*-coordinate of the building, so the building with the smallest left *x*-coordinate is first in the input file.

The Output

The output should consist of the vector that describes the skyline as shown in the example above. In the skyline vector $(v_1, v_2, v_3, \ldots, v_{n-2}, v_{n-1}, v_n)$, the v_i such that *i* is an even number represent a horizontal line (height). The v_i such that *i* is an odd number represent a vertical line (*x*-coordinate). The skyline vector should represent the ``path'' taken, for example, by a bug starting at the minimum *x*-coordinate and traveling horizontally and vertically over all the lines that define the skyline. Thus the last entry in the skyline vector will be a 0. The coordinates must be separated by a blank space.

Sample Input

```
1 11 5
2 6 7
3 13 9
12 7 16
14 3 25
19 18 22
23 13 29
24 4 28
```

Sample Output

```
1 11 3 13 9 0 12 7 16 3 19 18 22 3 23 13 29 0
```

A Plug for UNIX

You are in charge of setting up the press room for the inaugural meeting of the United Nations Internet eXecutive (UNIX), which has an international mandate to make the free flow of information and ideas on the Internet as cumbersome and bureaucratic as possible.

Since the room was designed to accommodate reporters and journalists from around the world, it is equipped with electrical receptacles to suit the different shapes of plugs and voltages used by appliances in all of the countries that existed when the room was built. Unfortunately, the room was built many years ago when reporters used very few electric and electronic devices and is equipped with only one receptacle of each type. These days, like everyone else, reporters require many such devices to do their jobs: laptops, cell phones, tape recorders, pagers, coffee pots, microwave ovens, blow dryers, curling irons, tooth brushes, etc. Naturally, many of these devices can operate on batteries, but since the meeting is likely to be long and tedious, you want to be able to plug in as many as you can.

Before the meeting begins, you gather up all the devices that the reporters would like to use, and attempt to set them up. You notice that some of the devices use plugs for which there is no receptacle. You wonder if these devices are from countries that didn't exist when the room was built. For some receptacles, there are several devices that use the corresponding plug. For other receptacles, there are no devices that use the corresponding plug.

In order to try to solve the problem you visit a nearby parts supply store. The store sells adapters that allow one type of plug to be used in a different type of outlet. Moreover, adapters are allowed to be plugged into other adapters. The store does not have adapters for all possible combinations of plugs and receptacles, but there is essentially an unlimited supply of the ones they do have.

Input

The input will consist of several case. The first line of the input contains the number of cases, and it's followed bya blank line. The first line of each case contains a single positive integer *n* ($1 \leq n \leq 100$) indicating the number of receptacles in the room. The next *n* lines list the receptacle types found in the room. Each receptacle type consists of a string of at most 24 alphanumeric characters. The next line contains a single positive integer *m* ($1 \leq m \leq 100$) indicating the number of devices you would like to plug in. Each of the next *m* lines lists the name of a device followed by the type of plug it uses (which is identical to the type of receptacle it requires). A device name is a string of at most 24 alphanumeric characters. No two devices will have exactly the same name. The plug type is separated from the device name by a space. The next line contains a single positive integer *k* ($1 \leq k \leq 100$) indicating the number of different varieties of adapters that are available. Each of the next *k* lines describes a variety of adapter, giving the type of receptacle provided by the adapter, followed by a space, followed by the type of plug.

There's a blank line between test cases.

Output

For each case, print a line containing a single non-negative integer indicating the smallest number of devices that cannot be plugged in. Print a blank line between cases.

Sample Input

```
1

4
A
B
C
D
5
laptop B
phone C
pager B
clock B
comb X
3
B X
X A
X D
```

Sample Output

```
1
```

Miguel A. Revilla
2000-02-09