Carnegie Mellon University

Invitational Programming Competition

March 19, 2005

Problems: 8
Pages: 14 (Including this one)
Input: From console
Output: To console

A. Nature Preserve

Description

A new program, jointly sponsored by the Commonwealth's Department of Interior and the City's Park Commission aims to establish wildlife preserves within the inner city. Abandoned inner city lots are reclaimed, cleared, and planted with once indigenous, now endangered tree species. In order to facilitate the planting and management of the trees, they are planted in a rectangular lattice. Because of the size of the city lots, there are never more than 100 trees per row or per column of the lattice.

Due to the recently prevalence of hot, dry summers, the Park Commission has installed sprinkler systems in all of the preserves. The sprinklers are configured in one of two ways. In one configuration a hose forms a circle, centered at a tree. In the second configuration, the hose forms a rectangle and is strung from tree-to-tree-to-tree. In either case, the system successfully waters those trees within or intersecting the hose. Each preserve can have one or more sprinkler systems installed.

The Department of the Interior is concerned about the haphazard approach taken by the Park Commission, which seemed more concerned about keeping water off of the surrounding streets than getting it to the precious trees. Your job, given descriptions of the planting and of the sprinkler system, is to determine how many trees are *not* being properly watered.

Input

The input consists of several lines. The first line is the number of preserve descriptions to follow. Each preserve description contains two sections. The first section is the planting description. The second section contains one or more sprinkler descriptions, with one line per sprinkler description.

Each planting description contains three integers separated by spaces: the number of rows of trees, *R*, the number of columns of trees, *C*, and the number of sprinklers, *S*.

Each sprinkler description begins with either a *C* for circular pattern or a *R* for rectangular pattern. Following a *C* are the coordinates of the tree that marks the center, in the form of an integer row and an integer column, and then an integer indicating the radius of the sprinkler pattern, where the radius is measured in trees. A radius of 0 indicates that only the center tree is watered.

Following a *R* are the coordinates of two corners defining the sprinkler system, where each corner is defined by the location of a tree specified as a row followed by a column, and the first tree is located at (0, 0). The coordinates denoting the rectangle can be in any order, but will uniquely define one rectangle.

Output

The output is a single integer per line, one for each preserve, in the same order as the preserves were described within the input. The integer indicates the total number of trees *not* watered by the sprinkler systems within the preserve.

Sample Input:

```
2
8 8
R 4 5 7 7
R 7 3 4 0
R 4 4 4 4
R 0 0 3 7
R 7 4 5 4
10 10
R 0 0 0 9
C 4 4 4
R 9 9 0 9
C 8 7 1
R 3 0 0 3
R 7 3 9 0
R 8 5 9 4
C 8 6 1
R 5 9 6 0
R 9 8 7 9
R 1 8 3 6
```

Sample Output

```
0
0
```

B. Perilous Pond

Description

With a tap of her wand and a wave of her broom, the wicked witch turned both the prince and the princess into frogs and set them free to enjoy life as commoners. Shortly after the formerly royal lovers began exploring their new saliental existence, they gained a new perspective on the dangers of the park. Their old friend the happy-go-lucky duck now seemed to consider them a dinner second only to breadcrumbs.

The former princess hopped some number of lilies ahead of her lover in a game of hard to get – only to discover that she had been cornered by a duck who was clearly eyeing her as dinner. Her only hope is that her prince will grab a mouthful of nearby breadcrumbs and deliver them to the duck, before she becomes the main course.

The breadcrumbs are at hand, or rather in mouth, but frog navigation isn't trivial – especially for those newly frogged. Navigating a pond requires leaping from lily pad to lily pad. And, lily pads can be delicate platforms. If a frog leaps too far, the pad can collapse upon take-off or landing. And, unfortunately, the witch wasn't kind enough to transform these royals into frogs that can swim.

But, frog instincts are good. A frog can rapidly survey a pond, locate the lily pads, and determine the greatest leap that each will support. Since a frog must both jump and land, and the force of each is equal, a frog can jump no more than is supported by the weaker of the two pads end points. When jumping, frogs can move at a rate of one foot per second.

Your job is to determine if the prince can save his princess.

Input

The first line of the input indicates the number of games described. Each subsequent game description contains several lines describing the game.

The pond is rectangular. The first line of each game's description specifies the dimensions of the pond, in feet and fractions thereof. It contains two numbers, the width of the pond and the length of the pond.

The next line is a single integer, *L*, which is the number of lily pads in the game. The next *L* lines specify the locations of the lily pads, one pad per line. The locations are measured from the corner which, from the prince's perspective, is on the far left. The first number per line is the distance to the right, a.ka., the x-coordinate. The second number per line is the distance from the back of the pond, a.k.a, the y-coordinate. The third number in the line is the strength of the pad – in other words, the maximum length of a leap that can be made to or from it.

Appearing after each of the three numbers on each line is a character, one of *S*, *F*, or *I*. *S* signifies the starting location, e.g., the initial location of the prince. *F* signifies the finish location, e.g., the location of the princess and the duck. *I* signifies any other, possibly intermediate, pad.

The last line of each game's description contains a single number, the amount of time, in seconds and fractions thereof, until the princess becomes dinner.

Output

There should be one line of output for each of the games in the input. The output should be presented in the same order as the input.

In the event that the princess becomes dinner, the output should consist of a single line, DINNER! In the event that the princess is saved, the output should consist of two numbers separated by a single space. The first number should be the total number of hops. The second number should be the amount of time the prince had to spare.

Sample Input

Sample Input:

```
2
6.5 6.5
3.5 0.0 2.25 F
3.5 2.0 2.25 I
3.5 4.0 2.25 I
3.5 6.0 2.25 S
2.5 3.0 5.25 I
5.5 2.0 3.0 I
5.5 3.0 9.0 I
5.5 4.0 6.2 I
7.5
9.5 7.2
1.5 3.5 3.9 S
3.5 1.5 3.0 I
2.5 1.5 3.0 I
3.5 3.5 1.0 I
4.5 3.5 1.0 F
99.5
```

Sample Output

```
3 1.5
DINNER
```

C. Counting Triangles

Description

Young Daphne lay in bed one night staring out the window, looking up at the wondrous sky. A sky full of stars. When she was little, she fell asleep by counting the stars. But now, in her adolescence, that game seems too easy. Instead, tonight, she has decided to count triangles. She wants to know how many unique triangles she can form from the stars in the sky – and resolves that she won't sleep until this question is answered. She considers two triangles to be unique if, and only if, they differ in the length of at least one side.

Fortunately for Daphne, and her early morning alchemy class, she has already cataloged each of the stars visible from her window and assigned them coordinates in the Cartesian plane, such that the upper left corner of her view has the coordinates (0,0).

Your job is to help Daphne by writing a program that will, given the stars' coordinates as input, compute the total number of unique triangles.

Input

The input consists of *V* views, where each view consists of a list of *N* stars. The first line of input is the number of views*, V*. Subsequent input describes each of these *V* views.

Each view's description begins with a single line containing a non-negative integer, *N*, the number of stars visible in this view. Each subsequent line within a view's description contains two non-negative integers, $X <= 10000$, and $Y <= 10000$, the (*X,Y*) coordinates of a star as visible form this view.

Output

For each of the *N* views within the input, there should be one line of output. This line should contain the number of unique triangles within the view. The views should be represented in the output in the same order as they appear within the input.

Sample Input

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

```
6
5 5
7 5
5 7
6 10
8 10
6 12
```

Sample Output

```
5
7
7
```

D. **Lawmakers**

The leader of a small country is responsible for approving or vetoing a list of proposed new laws. There are N laws, numbered 1 through N. He has assembled a board of advisors containing the nation's top experts in a wide array of fields to help with the decision. The advisors have been asked to make recommendations. A simple recommendation is can come in either of two forms:

Accept X
Reject X

The first indicates that the good of the country depends on accepting law X and the second indicates that the good of the country depends on rejecting law X. The leader also allows his advisors to submit a simple form of compound recommendation, which looks like this:

[simple recommendation] or [simple recommendation]

A compound recommendation indicates that, for the good of the country, at least one of the two simple recommendations must be followed.

The leader would like to comply with all the recommendations, but sometimes that turns out to be impossible! When that happens the leader's secretary must
schedule a meeting among the advisors to rethink their recommendations.

You, the leader's secretary, are responsible for determining whether complying
with all the recommendations is impossible.

Input format:

N [Number of simple recommendations]
[simple recommendations, 1 per line]
[compound recommendations, 1 per line]

Output format:

One line consisting of the text "Meeting required" or "Meeting not required"

Sample Input:

```
2 2 1
Accept 1
Reject 2
Accept 1 or Reject 2
```

Sample Output:

```
Meeting not required
```

E. Mosquito Planet

Description

After Darth Vader had attacked the Rebel base on the ice planet of Hoth, the Rebels temporarily moved to another unexplored planet, and set their base on a huge open plateau. The planet's dominant life form was mosquitos, and the rebels had to deploy a huge number of counter-mosquito units (CMUs). These units were simple robots, which walked to specified locations on the plateau and then continuously sprayed mosquito repellent; once a CMU began spraying, it could no longer change its location. The Rebels later discovered that the winds on the plateau always blew to the north and east, which affected the distribution of repellent. Specifically, if a CMU's coordinates were (x_0, y_0), then its repellent covered all points to the north and east of (x_0, y_0), as shown in Figure 1; in other words, it covered all points (x, y) such that $x >= x_0$ and $y >= y_0$. After the Rebels had noticed it, they realized that some CMUs were redundant; that is, they could be removed without reducing the protected area. For example, if the deployment included four CMUs in Figure 2, then the CMUs (2, 3) and (4, 4) were redundant. To save repellent, the Rebels decided to remove the redundant units. Your task is to write a program that determines the number of these redundant units.

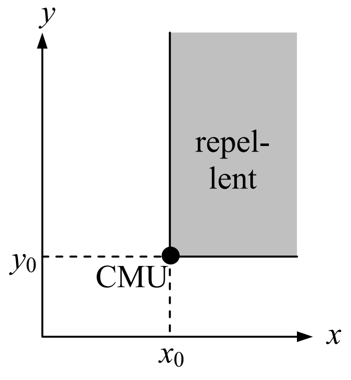

Figure 1

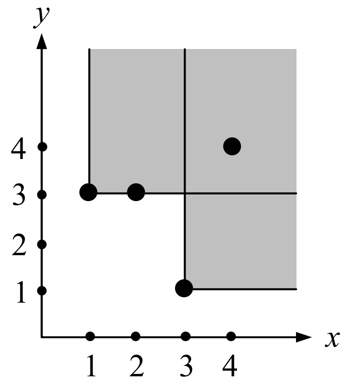

Figure 2

Input

The input includes multiple test cases; the first line of the input is a single integer between 1 and 100, which is the number of test cases. The first line of each test case is a single integer between 1 and 1000000, which is the number of counter-mosquito units. The other lines of the test case represent these units, one unit per line. The description of a unit consists of two integers between 1 and 1000000, separated by a single space, which are the unit's coordinates.

Output

The output should show the number of redundant CMUs for each test case; each number should be on a separate line, with no surrounding spaces.

Sample Input

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

Sample Output

```
0
1
2
```

F. We’re meeting *where*?

Universities are centers of activities. Classes. Dances. Study sessions. Office hours. Committee meetings. Lab meetings. Grading parties. Programming contest. Each of these events requires space – conference rooms, classrooms, hallway niches, almost anything can do. The scheduling of the available events into available spaces is always a contentious affair. Although the competition can never be eliminated and will always be among the best played academic sports, good scheduling can keep the competition all in good fun.

Given a list of events, where each event is specified by its start time and end time, determine the minimum number of rooms required to satisfy the demand. If two events overlap, they require separate rooms; for instance, the events with time intervals [1..4] and [4..8] can be in the same room, whereas [1..4] and [3..7] require separate rooms.

Input

The input consists of a list of schedules. The first line of the input contains a single integer, *S,* indicating the total number of independent schedule descriptions to follow. Each schedule description begins with a line consisting of a single, positive, integer, *E,* th*e* total number of events within the schedule. The following *E* lines each contain an event specification. The first integer of the event specification is the event’s start time. The second integer within the event specification is the event’s finish time. The finish time is strictly greater than the start time. The time values are between 1 and 10^8, and the total number of events within a schedule is between 1 and 100,000.

Output

The output is a single integer number, which represents the minimal required number of rooms.

Sample input
```
2
7
1 4
4 5
3 6
5 7
5 8
7 9
0 0
2
1 9
9 12
```

Sample output
```
3
1
```

G. Characteristic squares?

Description

The Traveler Scrutinizing Administration (TSA) is evaluating a new traveler identification technique for correctness. This technique collects biometric data from each traveler at the security checkpoint. For rapid identification, each traveler's data is then encoded as the four coordinates, within a two-dimensional plane, of characteristic square. It is believed that no two travelers will have the same characteristic square – a square with the same orientation and side lengths.

In order to evaluate this technology, the TSA conducts a test at least one airport using volunteers., each of which is scanned exactly once. The resulting biometric data is encoded, as described above, and the vertices are stored within a data file. In order to obscure the data, an unknown number of arbitrary data points are introduced into the data file. These data files are then merged and sent to the top secret joint-military Special Engineering Innovations (SEI) lab for analysis. Their job is to determine if any two travelers had the same characteristic square – such a result would demonstrate the identification system to be inaccurate.

Unfortunately, the TSA recorded only the coordinates of the vertices – not the characteristic squares. Is everything lost? Your job is to determine if the data, with lines lost and extra points added, can still demonstrate that no two travelers had the same characteristic rectangles.

Input

The first line of input is a non-negative integer indicating the number of data sets to follow. Each data set begins with a non-negative integer indicating the total number of points to follow. Each subsequent line within a data set contains two non-negative integers, the first is the X-coordinate. The second is the Y-coordinate. Should the same coordinates be recorded more than once, they will appear in the data more than once.

Output

The output consists of one line for each data set. The output line is UNIQUE if the data set demonstrates that no two volunteers within the data set could have had the same characteristic rectangle or AMBIGUOUS if this cannot be shown.

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

```
12
3 0
7 0
8 0
12 0
7 3
8 3
0 4
15 4
4 7
11 7
4 14
11 14
```

Sample Output

```
AMBIGUOUS
UNIQUE
```

H. Prime Gambling

Description

The university administration has decided to install several slot machines in the university center, which are called Casino Machines for Universities (CMUs). This machines are similar to the usual casino slot machines, but the gambling is based on various science facts. In particular, one machine allows students to bet on the number of primes in a specified interval. After a student drops a quarter into this machine, it displays an interval of integers, and the student has to guess the number of primes in this interval. If her guess is larger than the correct answer, she loses; if her guess is an underestimate, she gets back her quarter; finally, if her guess is right, she wins a T-shirt with the slogan "Gamble less, study more." Your task is to write a program that verifies the correctness of guesses.

Input

The input is a list of games, one game per line. The first line of indput indicates the total number of games. Each subsequent line describes a single game. The description of a game includes three integers, separated by single spaces. The first two integers are the beginning and end of an interval; these integers are between 2 and 1,000,000, and the first of them is no greater than the second. The third integer is a student's guess about the number of primes in this interval, which is between 0 and 100,000. The total number of games is at most 1,000,000..

Output

The output shows the outcome of each game, one outcome per line. If the guess is greater than the correct answer, the outcome is "*loss*"; if it is less than the correct answer, the outcome is "*draw*"; finally, if it is correct, the outcome is "*win*".

Sample input

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

Sample output

```
win
win
draw
loss
```