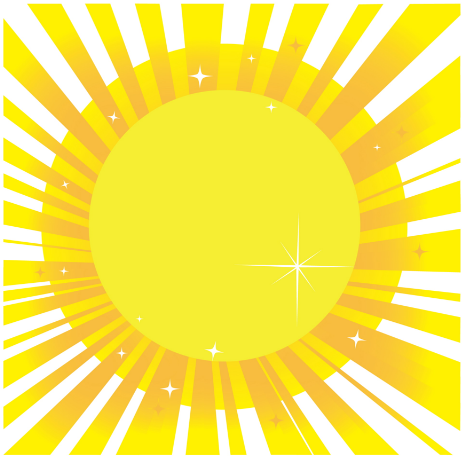
COMPUTER AND INFORMATION SCIENCE

UNIVERSITY OF OREGON

Twenty-first Annual University of Oregon Eugene Luks Programming Competition

2017 April 29, Saturday
am10:00 – pm2:00



Problem Contributors
Daniel Lowd, Eugene Luks, Chris Wilson

Technical Assistance and Organization
Lauradel Collins, Anton Matschek

How Many Teams?

You have a new job assisting your local computer science department in forming teams for a programming competition. Each team must be formed as a group of three friends, and you need to write a program to determine from a list of (pair-wise) friends how many possible teams could be formed.

For example, if we have students numbered 1, 2, 3, and 4 and it turns out that all are friends with each other, then there would be 4 possible teams:

```
possible teams
   1 2 3
   1 2 4
   1 3 4
   2 3 4
```

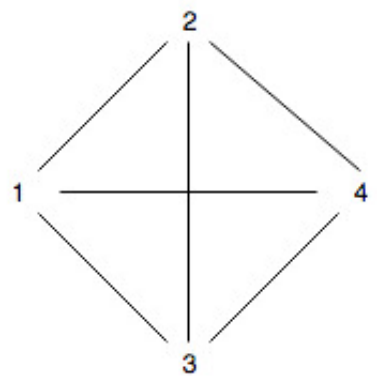
The list of friends will be given as an undirected graph. There will be an edge between U and V if U and V are mutual friends. Note that if U and V are friends, then V and U are friends (friendship goes both ways). However, If U and V are friends, and V and W are friends, it *might* not be true that U and W are friends. That would only be true if U and W are in the friendship list: that is, if (U,W) is also an edge.

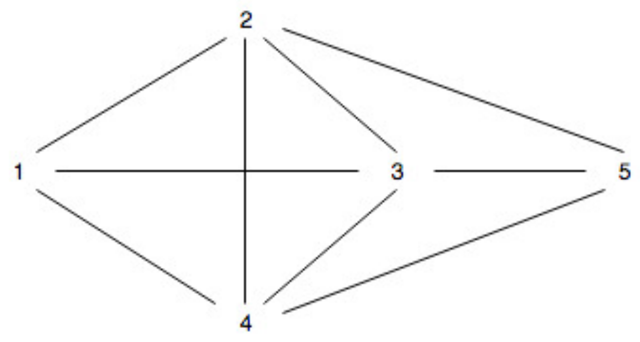
The input will start with a number C, $C \leq 10$, of test cases. Each test case starts with two integers N and M, on separate lines, where $N \leq 100$, is the number of students. The next number M, $M \leq 4950$, is number of edges (friendships). The following M lines contain a pair U V, where $1 \leq U, V \leq N$ represents an edge.

The output should consist of one integer for each test case: the number of teams that could be formed of three people, all pairs of whom are friends. In other words, find the number of distinct triples U V W such that (U,V), (V,W), and (W,U) are edges. (Note that a team such as 1 3 4 is the same as 4 1 3 and should only be counted once.)

See reverse for Sample I/O.

Sample Input
```
2
4
6
1  2
1  3
1  4
2  3
2  4
3  4
5
9
1  2
1  3
1  4
2  3
2  4
2  5
3  4
3  5
4  5
```





Sample Output
```
4
7
```

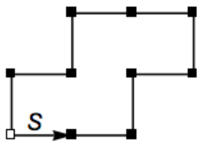
LOTH-CATTERIES

Since their domestication on Lothal by Star Wars rebels, loth-cats have dominated the interstellar pet-trade. Pet emporiums throughout the Outer Rim have been constructing catteries to house these adorable-but-temperamental creatures. Loth-cattery walls must be constructed from aromatic Lothal-pine panels which come in one-meter widths and are equipped with hinges that allow straight extensions or right-angle turns. This enables constructions of catteries that simulate the Loth-cats' twisty burrows.

The specification for a cattery construction consists a sequence of panel-positions comprising the perimeter. After a starting panel "S" a panel position is indicated by its orientation with respect to its predecessor:

"F" indicates continue in the same direction.
"R" indicates turn right.
"L" indicates turn left.

Thus, the string "SFLRLLFLRL" specifies the cattery



(In this picture, the first panel starts horizonally from the lower left.) Since animal welfare regulations require *one square meter of space per loth-cat*, this cattery can accommodate 4 loth-cats.

In this problem you are asked to determine the loth-cat capacity of specified cateries.

The first line of the input will declare the number *N* of problem instances to be considered. Each of the next *N* lines will contain a string that starts with "S" and consists otherwise of "F"s, "R"s, and "L"s. The string will always specify the closed, non self-intersecting, perimeter of a legitimate cattery.

For each cattery, the output should indicate the log-cat capacity (that is, the area of the enclosed region).

Sample Input

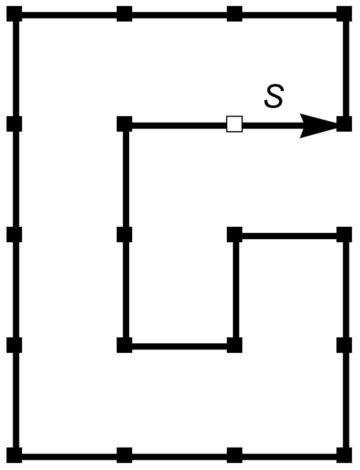
```
3
SFLRLLFLRL
SLLFFLFFFLFFLFLLRRFR
SFRLLFRLLFFFRFLLRLLRRLLRLL
```

Sample Output

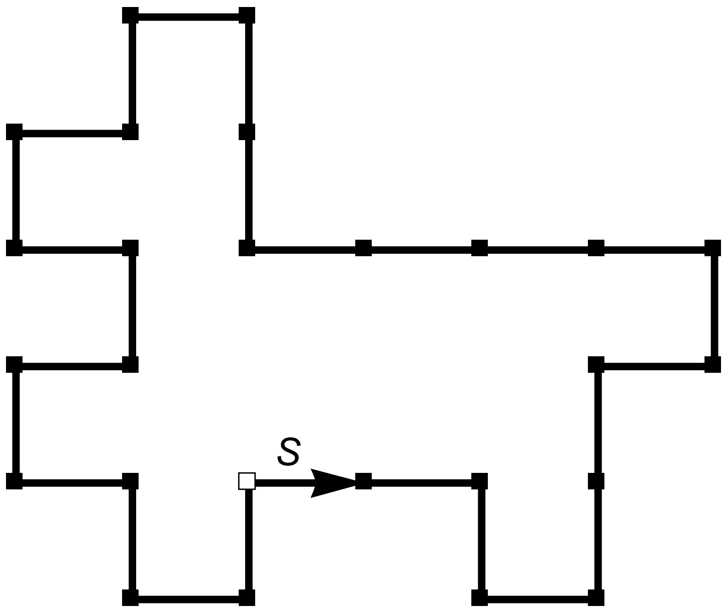
```
4
9
15
```

The second and third samples are sketched on the reverse.

SLLFFLFFFLFFLFLLRRFR



SFRLLFRLLFFFRFLLRLLRRLLRLL



CANCEL THE FEWEST TALKS

The Computer Science Department at the University of Turing has many applications for their rare faculty positions, so the time slots to give talks there are highly coveted. All the applicants will give a talk on the same day in the same room, and each person's talk has a fixed start and finish time. Since these times may overlap, and no talks will be allowed to occur at the same time, your job is to write a program that will determine the *fewest* number of talks to cancel.

Suppose we indicate a talk's start and finish time as a pair *(start,finish)*, where *start* and *finish* are integers. We allow that one talk may start immediately after one talk ends (so the finish time of one talk can be equal to the start time of the next). For example:

- If the talks are (1,2), (3,4), (1,3), (2,3), then the answer is **1**. This is because we can cancel only (1,3) and keep (1,2), (2,3), (3,4).
- If the talks are (1,2), (1,2), (1,2), then **2** talks must be cancelled as only one talk can be given in the (1,2) slot.
- With an input talk schedule of (3,4), (4,5), (7,9), (9,10), none have to be cancelled so the correct output is **0**.

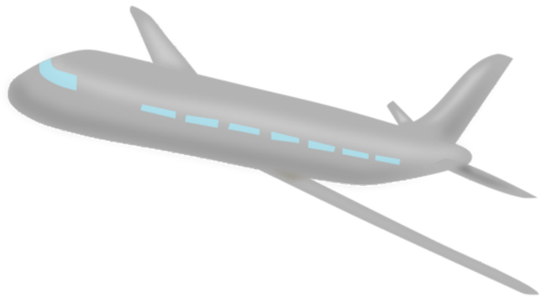
The input will start with a number C, C ≤20, of test cases. Each test case will start with an integer N, 1≤N≤ 500, the number of talks for this case. This is followed by N lines of the form S F, where S and F are integers, 0≤S<F≤500, with S indicating the start time of a talk and F the finish time. You may **not** assume that the list of talks is sorted by either the start time S or the finish time F, but you can assume that S is strictly earlier than F (that is, S<F).

The output needs to be a single integer R for each test case on its own line. This number R is the fewest number of talks that will need to be removed from the schedule (cancelled) so that the remaining talks do not overlap.

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

Sample Output
```
1
4
```

AIRLINE REACCOMMODATION



Each group of customers who flies on Turing Airlines (TA) (motto: "We Fly the Computable Skies!") specifies an amount of money they would accept as compensation for being placed on a later flight. When TA needs to make room for crew members, it removes one or more groups of customers, paying each group the amount of money they specified. For example, a family of four might accept being removed for $1200, and a sports team with 15 players might accept being removed for $3000. TA can pay the family $1200 to make room for four crew members, pay the sports team $3000 to make room for 15 crew members, or pay them both a total of $4200 to make room for 19 crew members. However, TA never removes a partial group.

As a staff engineer for Turing Airlines, you need to implement an algorithm to compute the lowest cost required to remove a specified number of passengers, **k**. You may not remove part of a group, but you are allowed to remove groups totaling more than **k** passengers as long as the cost is minimized.

For example, suppose we need to remove 2 customers and we have the following 4 groups:

Group Size	Cost to Remove
1	$100
1	$150
2	$300
3	$200

The cheapest way to remove (at least) 2 customers is to pay the last group $200.

The first line of the input is the number of test cases C (C ≤ 10), each case representing one flight. Each test case begins with the number of passengers to remove, K (1 ≤ K ≤ 100), and the number of customer groups, N (1 ≤ N ≤ 100). For each group of customers, a line specifies the number of customers in the group (S ≤ 100) and the amount of money they require as compensation (M ≤ 1,000,000). All numbers are integers.

The output should consist of one integer for each test case, specifying the minimum total cost to reaccommodate the passengers.

Sample Input
```
2
2 4
1 100
1 150
2 300
3 200
3 5
1 100
1 200
1 400
2 500
3 450
```

Sample Output
```
200
450
```

PENULTIMATE PLACE

```
1587757598421260967737З
0684333105560027512881８
9158103556020438257883１
7451476866839157746654７
3226015166724808998952５
3965079480050324329465５
1337958326189618487154３
0192280772058839151559９
5313954992421535363674６
9979311746316451339460４
9290479894023066192126６
5579903061012709912534４
5441905752388665008544３
```

The double-exponential a^{b^c} can represent a huge number even with modest values of *a,b,c*. For example, 3^{5^4} expands to the 299-digit number in the box to the right. Conversely, the famously-immense *googol* is compactly expressed by 10^{10^2}.

This problem will deal with such double-exponentials with *a,b,c* each lying between 2 and 99. Mercifully, you will have to produce only the next-to-last digit of the expansion of the number.

The first line of the input will declare the number *N* of problem instances to be considered. Each of the next *N* lines will contain a triple of integers *a b c* separated by single spaces. The output should indicate the *next-to-last* digit (i.e., the *tens' place*) in the decimal expansion of a^{b^c}.

For example, since $5^{2^3} = 5^8 = 390625$, the input "5 2 3" should yield the output "2".

Sample Input

```
8
2 2 2
3 2 4
3 5 4
7 8 9
9 9 9
2 7 4
5 3 20
10 10 2
```

Sample Output

```
1
2
4
0
8
5
2
0
```

DO THE RANKINGS AGREE?

Your team has been retained by the director of a competition who supervises a panel of judges. The competition asks the judges to assign integer scores to competitors – the higher the score, the better. Although the event has standards for what score values mean, each judge is likely to interpret those standards differently. A score of 100, say, may mean different things to different judges.

The director's main objective is to determine which competitors should receive prizes for the top positions. Although absolute scores may differ from judge to judge, the director realizes that relative rankings provide the needed information – if two judges rank the same competitors first, second, third, ... then they agree on who should receive the prizes.

Your team is to write a program to assist the director by comparing the scores of pairs of judges. The program is to read two lists of integer scores in competitor order and determine the highest ranking place (first place being highest) at which the judges disagree.

The first line of input will be a number C, the number of competitor pairs. What follows will be a series of C score list pairs. Each pair begins with a single integer giving the number of competitors N, $1 < N < 1{,}000$. The next N integers are the scores from the first judge in competitor order. These are followed by the second judge's scores – N more integers, also in competitor order. Scores are in the range 0 to 100,000 inclusive. Judges are not allowed to give ties, so each judge's scores will be unique. Values are separated from each other by one or more spaces.

For each score pair, print a line with the integer representing the highest-ranking place at which the judges do not agree. If the judges agree on every place, print a line containing only the word 'agree'. Use the format below: "`Case`", one space, the case number, a colon and one space, and the answer for that case with no trailing spaces.

Sample Input

```
2
4
3 8 6 2
15 37 17 3
8
80 60 40 20 10 30 50 70
160 100 120 80 20 60 90 135
```

Sample Output

```
Case 1: agree
Case 2: 3
```