A Geodes

Problem

In the mountains of Geodesia, geodes are found. These 'hollow' stones contain cavities with crystal formations around them. The beautifully colored crystals can be sold for a high price to people both inside and outside of Geodesia. The problem with geodes is that one cannot see from the outside of a given rock whether it is a geode or not. Out of every thousand rocks found in Geodesia, only some are geodes.

In order to find out whether a given rock is in fact a geode, one can try to consider its density

$$\rho = \frac{m}{V},$$

where m represents its mass and V its volume. Since a geode contains empty space, a rock contaning one is expected to have a lower density than other rocks. If the density of a rock is too high, it would be a waste of time and effort to further investigate it.

But how could one determine the density of each of the enormous number of rocks found in the mountains of Geodesia? Mining crafts collect rocks, which they put on mobile conveyer belts. Weighing the rocks automatically at a fast rate is no problem, but how to quickly determine their volume? Measuring e.g. the volume displacement of a liquid (like H_2O), where each rock has to be put in and taken out individually is a time consuming process.

So why not try to estimate the volume just by *looking* at the rock? Along the conveyer belt, two cameras are placed, perpendicular to each other. Of every rock that passes, they each take a picture. These pictures are sent to a computer for processing. The computer calculates from these pictures an upper bound for the volume of the rock, $V_{\max}$. A lower bound for the density of the rock is then given by

$$\rho_{\min} = \frac{m}{V_{\max}} \leq \frac{m}{V} = \rho.$$

This lower bound can then be used to reject rocks that are certainly too heavy to be geodes.

The computer uses a relatively simple trick for estimating the maximal volume. It considers the pictures taken as silhouettes, as parallel projections of a rock onto a plane. Since the two cameras are placed perpendicular to each other, the projection planes are also perpendicular to each other.

In other words, choose a coordinate system in three dimensional space as follows: z runs along the vertical direction and x and y run along the perpendicular horizontal directions in which the two cameras are placed. Let $S \subset \mathbb{R}^3$ represent a rock. Then the cameras yield the two parallel projections

$$P_{xz} = \{(x, z) : (x, y, z) \in S \text{ for some } y \in \mathbb{R}\} \text{ and}$$
$$P_{yz} = \{(y, z) : (x, y, z) \in S \text{ for some } x \in \mathbb{R}\}$$

on the xz and yz plane respectively. See Figure 1 for an example.

Now a curious property of the rocks found in Geodesia makes the measurement of the volume easier: they are all *convex*[1]. Furthermore, the projections can be considered as polygons. So first each of the two pictures taken of a rock is converted into a convex polygon by an image recognition program. You are to write a program that will do the second step: compute from these two polygons $V_{\max}$: the maximum volume of any rock having exactly these two polygon-shaped projections.

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

[1] A set S (in $\mathbb{R}^2$, $\mathbb{R}^3$, or in fact in any vector space you like) is convex if any straight line segment connecting any two points $a, b \in S$ is again completely within S. Formally this means that for all $a, b \in S$ and all $\lambda \in [0, 1]$, also $\lambda a + (1 - \lambda)b \in S$.

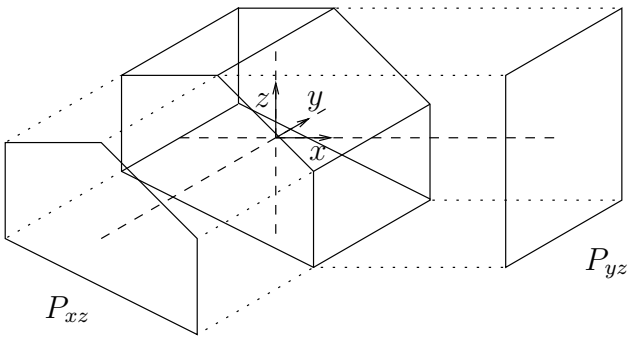


Figure 1: Example object with projections P_{xz} and P_{yz}.

- One line with the integer n_{xz}, the number of vertices describing the projection on the xz plane.
- n_{xz} lines, each with two integers x and z, separated by a single space: the coordinates of one vertex of the xz projection.
- One line with the integer n_{yz}, the number of vertices describing the projection on the yz plane.
- n_{yz} lines, each with two integers y and z, separated by a single space: the coordinates of one vertex of the yz projection.

Both projections are convex polygons, and the minimum and maximum z coordinates are the same values. The vertices of the polygons are given in clockwise order. Consecutive vertices of the polygons are different, but there may be three or more consecutive vertices on one line. The numbers in the input satisfy $3 \leq n_{xz}, n_{yz} \leq 1,000$ and $|x|, |y|, |z| \leq 500$.

Output

For every test case in the input, the output should contain a single number on a single line: V_{max}, the maximum volume of any object having the projections given in the input, rounded in the usual way to two decimals behind the decimal point, where $\frac{n+\frac{1}{2}}{100}$ is to be rounded to $\frac{n+1}{100}$, with $n = 0, 1, \ldots, 99$. A round-off error of 0.01 is permitted in your answer.

Example

This example corresponds to Figure 1.

Input	Output
1 5 1 0 1 -1 -1 0 -1 1 0 1 5 -1 1 0 1 1 1 1 -1 -1 -1	5.00

B Bowling

Problem

Last Saturday, Paul, Ivo and Nick went to a bowling alley. They hired a lane for one hour, and played three games.[2] In this casual game play, the guys were simply counting the number of games they won: the overall winner would be the one who won the most games. (In case of a tie, they would have had to figure out another way to pronounce the winner.)

This time, Paul won two games and Nick one, so Paul was pronounced the winner. Ivo wondered how it was possible he lost all three games, even though he felt his scores were not too bad. So he looked again at the scoreboard and saw this:

Game	1	2	3
Paul	102	86	94
Ivo	98	73	112
Nick	95	84	125

Paul indeed won two games, and Nick one, but Ivo noted that if he would have played his games in another order, namely 112 – 98 – 73, *he* would have won the two games, and Paul none!

Ivo's remark is not as unreasonable as one might think at first glance: in bowling, each game is played independent of the other players, except perhaps for some psychological side effects. The successive scores of an individual player can also be assumed to be more or less independent and identically distributed.

Ivo wisely kept his remarks to himself last Saturday, but later he started to think about using his idea of 'potential number of wins' as a measure of performance. He concluded that it would also be interesting to know the *minimum* number of wins for each player. Also, it would be nice not only to allow permutation of the games of the player whose number of wins is being maximized or minimized, but also permutation of the games of the other players. Finally, Ivo has to define when exactly a player 'wins' a game: a player wins a game if and only if his score is strictly greater than the score of each of the other players.

Ivo asks you to write a program that calculates for each player i:

- m_i, the minimal number of games player i would have won, regardless of the permutation of the scores of each individual player, and
- M_i, the maximal number of games player i could have won if the scores of each individual player were permuted optimally to that purpose.

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with two integers P and G, separated by a single space, that indicate the number of players and the number of games respectively ($2 \leq P \leq 100$ and $2 \leq G \leq 1,000$).
- P lines, one for each player, each with G integers, separated by single spaces: the successive scores for that player.

Each score is between 0 and 300 (both inclusive).

Output

For every test case in the input, the output should contain P lines, one for each player in the input. On the i^{th} line should be two integers, separated by a single space: m_i, followed by M_i.

[2] In ten-pin bowling, a *game* consists of ten *frames* per player, each of which holds up to two chances to knock down the ten pins at the end of the lane. (As you might know, the last frame might be tricky.)

Example

The first example below corresponds to the example in the text.

Input

```
2
3 3
102 86 94
98 73 112
95 84 125
3 4
100 110 112 116
98 112 110 112
90 98 113 113
```

Output

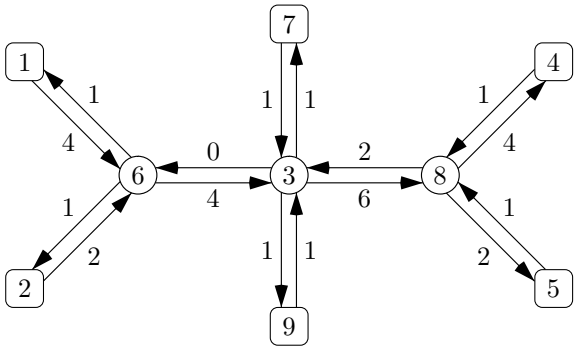
```
0 2
0 2
1 2
1 3
0 2
1 2
```

C High Spies

Problem

You have been approached by a spy agency to determine the amount of contraband goods that are being traded among several nefarious countries. After being shipped from its country of origin, each container of goods is routed through at least one neutral port. At the port, the containers are stored in a warehouse before being sent on their way, so you cannot trace individual containers from their country of origin to their final destination. Satellite cameras can tell you the number of containers travelling in each direction on each leg of the journey. They cannot distinguish individual containers, nor do you have information on the times individual containers have been observed. You know, however, that every container takes the shortest possible route to its destination.

The task is to determine both the maximum and minimum number of containers that could have been travelling from one country to another. The transport network that is observed is an unrooted tree, with countries as leaves and ports as internal nodes. For each edge the number of containers is given in two directions. You know from the description above that no container leaves a port in the direction it came from.



In the figure above we see six countries $1, 2, 4, 5, 7, 9$ and three ports $3, 6, 8$. Although all edges on the route from country 1 to country 4 have a capacity of at least 4, it is not possible to send four containers along this route. In that case, in port 6 one of the containers arriving from country 2 would be forced to return, which is forbidden.

This problem was taken from Dennis E. Shasha's monthly column *Puzzling Adventures*, published in *Scientific American*, July 2003.

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case describes a single unrooted tree, in the following format:

- One line with one integer n ($3 \leq n \leq 10,000$): the number of nodes (countries and ports).
- $n-1$ lines with four integers a, b, c_1, c_2 each ($1 \leq a, b \leq n$ and $0 \leq c_1, c_2 \leq 1,000$), describing an edge from node a to (another) node b with c_1 containers counted in the direction from a to b and c_2 in the other direction.
- One line with two integers fr and to ($1 \leq fr, to \leq n$, $fr \neq to$): the begin and end nodes (countries, i.e., leaves in the tree).

Integers on the same line are separated by single spaces. The input satisfies Kirchhoff's law: in each port the number of incoming containers equals the number of outgoing containers. This in itself is not sufficient to guarantee the existence of a transport of containers that matches the

description (as containers never move back from where they came). In each test case given a matching transport exists.

Output

For every test case in the input, the output should contain a single line with two integers, separated by a single space, the minimum and maximum number of containers transported from *fr* to *to* fitting the data given.

Example

Six countries, connected via three ports, corresponding to the picture in the text. Information is required on the number of containers shipped from country 1 to country 4.

Input

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

Output

```
1 3
```

D Digital Friends

Problem

Two positive integers are called *friends* if they consist of the same decimal digits. So 123 and 32331313323213 are friends, but 123 and 22121221 are not.

Two positive integers (that are not friends) are called *almost friends* if a single neighbour exchange in one of them results in a pair of friends. A *neighbour exchange* changes two neighbouring digits a and b into $a-1$ and $b+1$, or into $a+1$ and $b-1$, provided that these new digits are still in the range $0 \ldots 9$, and that no leading zero is generated. So 123 and 2223042 are almost friends (let $04 \rightarrow 13$), and 137 and 470 are neither friends nor almost friends (note that $13 \rightarrow 04$ is not allowed).

The problem is to determine if two given integers are friends or almost friends.

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with two integers x and y, separated by a single space, with $0 < x, y < 10^{100}$. Both integers start with a non-zero digit.

Output

For every test case in the input, the output should contain a single line with the string `"friends"` or `"almost friends"` or `"nothing"`, reflecting the property of the two given integers.

Example

The examples below correspond to the four examples mentioned in the text.

Input	Output

`4`
`123 32331313323213`
`123 22121221`
`123 2223042`
`137 470`
	`friends`
	`almost friends`
	`almost friends`
	`nothing`

E The Bavarian Beer Party

Problem

The professors of the *Bayerische Mathematiker Verein* have their annual party in the local Biergarten. They are sitting at a round table each with his own pint of beer. As a ceremony each professor raises his pint and toasts one of the other guests in such a way that no arms cross.

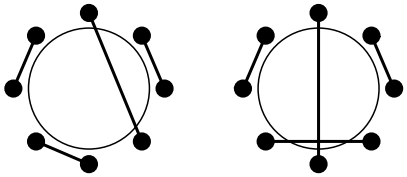


Figure 2: Toasting across a table with eight persons: no arms crossing (left), arms crossing (right)

We know that the professors like to toast with someone that is drinking the same brand of beer, and we like to maximize the number of pairs of professors toasting with the same brand, again without crossing arms. Write an algorithm to do this, keeping in mind that every professor should take part in the toasting.

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with an even number p, satisfying $2 \leq p \leq 1,000$: the number of participants.
- One line with p integers (separated by single spaces) indicating the beer brands for the consecutive professors (in clockwise order, starting at an arbitrary position). Each value is between 1 and 100 (boundaries included).

Output

For every test case in the input, the output should contain a single number on a single line: the maximum number of non-intersecting toasts of the same beer brand for this test case.

Example

Input	Output
2 6 1 2 2 1 3 3 22 1 7 1 2 4 2 4 9 1 1 9 4 5 9 4 5 6 9 2 1 2 9	3 6

F Evacuation

Problem

Fires can be disastrous, especially when a fire breaks out in a room that is completely filled with people. Rooms usually have a couple of exits and emergency exits, but with everyone rushing out at the same time, it may take a while for everyone to escape.

You are given the floorplan of a room and must find out how much time it will take for everyone to get out. Rooms consist of obstacles and walls, which are represented on the map by an `'X'`, empty squares, represented by a `'.'` and exit doors, which are represented by a `'D'`. The boundary of the room consists only of doors and walls, and there are no doors inside the room. The interior of the room contains at least one empty square.

Initially, there is one person on every empty square in the room and these persons should move to a door to exit. They can move one square per second to the North, South, East or West. While evacuating, multiple persons can be on a single square. The doors are narrow, however, and only one person can leave through a door per second.

What is the minimal time necessary to evacuate everybody? A person is evacuated at the moment he or she enters a door square.

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with two integers Y and X, separated by a single space, satisfying $3 \leq Y, X \leq 12$: the size of the room.
- Y lines with X characters, each character being either `'X'`, `'.'` or `'D'`: a valid description of a room.

Output

For every test case in the input, the output should contain a single line with the minimal evacuation time in seconds, if evacuation is possible, or `"impossible"`, if it is not.

Example

Input

```
3
5 5
XXDXX
X...X
D...X
X...D
XXXXX
5 12
XXXXXXXXXXXX
X..........D
X.XXXXXXXXXX
X..........X
XXXXXXXXXXXX
5 5
XDXXX
X.X.D
XX.XX
D.X.X
XXXDX
```

Output

```
3
21
impossible
```

G Decompression

Problem

Consider the following scheme for permuting a string of letters: first we create all different rotations of the string, then we sort them in lexicographical order, and finally we take the last letter of each rotation and concatenate them to form a new string [3].

For the string `"LEIDEN."`, for example, the seven rotations in lexicographical order are (where the period `'.'` comes before any letter):

```
.LEIDEN
DEN.LEI
EIDEN.L
EN.LEID
IDEN.LE
LEIDEN.
N.LEIDE
```

so this results in the string `"NILDE.E"`.

At first glance this permutation doesn't seem useful at all, but it has an interesting property. If there are a lot of equal substrings in the original string (which might happen in case of a real language), then a lot of equal consecutive letters occur after the permutation. Therefore the resulting string is very suitable for *block compression*, where blocks of equals letters are replaced by that letter followed by a number which specifies how often that letter occurs. If the letter only occurs once, no number is added. For example the string `"AAABCC"` would be replaced by `"A3BC2"`.

Your task is now to decompress this final string to the original string. Note, however, that permuting the string is not entirely reversible: you can only obtain the original string up to a rotation (i.e., each rotation would lead to the same permuted string). To overcome this, the original string will consist of uppercase letters followed by a single period (`'.'`). This will define the initial rotation.

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with the compressed string. This string will consist of uppercase letters, numbers and a single period, and will be a valid block compressed string. The length of the original string that resulted in the compressed one will be no more than $1,000,000$ characters.

Output

For every test case in the input, the output should contain a single line with the decompressed string.

Example

Input	Output

`3`
`NILDE.E`
`N13.E13`
`LRSGORTNOMOMAIOSROC2.GAPTE2NS`
	`LEIDEN.`
	`ENENENENENENENENENENENENEN.`
	`PROGRAMMINGCONTESTSARESOCOOL.`

[3] This transformation is called the Burrows-Wheeler transform.

H Rummikub

Problem

Rummikub is a simple game, often played by elderly people versus their grandchildren. It is a tile-based game and each of the tiles has a colored number written on it. There are four available colors: yellow, red, green and black. A tile is described by its number followed by a lower case letter, which is the first letter of its color. All tiles are unique.

The goal of the game is simple: get rid of all your tiles. This sounds easier than it is, since you can only use them in two specific ways. You can get rid of tiles by constructing *runs* or *groups*. A run is composed of three or more consecutive numbers of the same color, for example 9r, 10r, 11r or 60g, 61g, 62g, 63g, 64g, 65g. A group is composed of three or four tiles with the same numbers, and therefore different colors. Examples are 1r, 1g, 1b and 17r, 17g, 17b, 17y. In constructing the runs and groups, you may only use each tile once.

The value of a run or group is the sum of the numbers on the tiles. Your score is the sum of the values of all the runs and groups you construct. What is your maximum possible score?

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with one integer N, satisfying $1 \leq N \leq 400$: the number of tiles you have.
- One line with N strings separated by single spaces, each consisting of a number between 1 and 100 and a character which is either `'y'`, `'r'`, `'g'` or `'b'`: the available tiles. All tiles are unique.

Output

For every test case in the input, the output should contain a single number, on a single line: the maximum possible score.

Example

Input	Output
3 5 7g 7b 7r 8r 9r 7 23b 1y 24b 1r 93b 1b 100r 8 2y 2r 2g 2b 4g 5g 6g 7g	24 3 30

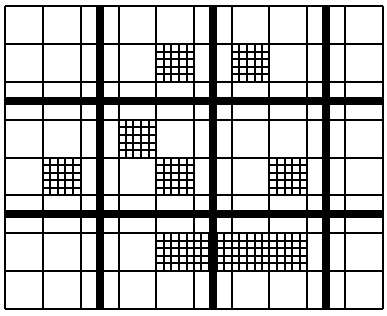
I Make it Manhattan

Problem

Chaos City has grown out of control. Buildings have been built everywhere and the city layout is a complete mess. The mayor decided that this must come to an end, and wants to create a nice, structured city.

After some research, he found the ideal way to make this happen. Inspired by the New York district of Manhattan, he wants all buildings organized in a rectangular grid, separated by avenues running from North to South and streets running from West to East. These streets and avenues should all be separated by the same distance D.

In the current situation, the buildings are already organized in a rectangular grid. In fact, each building fills exactly one square in this grid. However, with all the buildings randomly scattered across the city, it may be impossible to build the roads without demolishing a couple of buildings. To keep most citizens happy, the mayor wants to demolish as few buildings as possible. Given the current locations of the buildings, what is this minimum number?



The above picture illustrates the problem. The shaded squares are the initial locations of the buildings. If the roads should be separated by a distance of three, the thick lines indicate the optimal placement of the roads and one building has to be demolished.

Input

The first line of the input contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with two integers D and N, separated by a single space, satisfying $1 \leq D \leq 1,000$ and $0 \leq N \leq 100,000$: the distance between two roads, and the number of buildings in the city, respectively.
- N lines with two integers x_i and y_i, separated by a single space, satisfying $-10^9 \leq x_i, y_i \leq 10^9$: the positions of the buildings.

Output

For every testcase in the input, the output should contain a single line with the minimum number of buildings that has to be demolished.

Example

This example corresponds to the picture in the text.

Input

```
1
3 10
1 0
2 0
3 0
4 0
1 2
0 3
1 5
3 5
4 2
-2 2
```

Output

```
1
```

Problem J. Booksort

The Leiden University Library has millions of books. When a student wants to borrow a certain book, he usually submits an online loan form. If the book is available, then the next day the student can go and get it at the loan counter. This is the modern way of borrowing books at the library.

There is one department in the library, full of bookcases, where still the old way of borrowing is in use. Students can simply walk around there, pick out the books they like and, after registration, take them home for at most three weeks.

Quite often, however, it happens that a student takes a book from the shelf, takes a closer look at it, decides that he does not want to read it, and puts it back. Unfortunately, not all students are very careful with this last step. Although each book has a unique identification code, by which the books are sorted in the bookcase, some students put back the books they have considered at the wrong place. They do put it back onto the right shelf. However, not at the right position on the shelf.

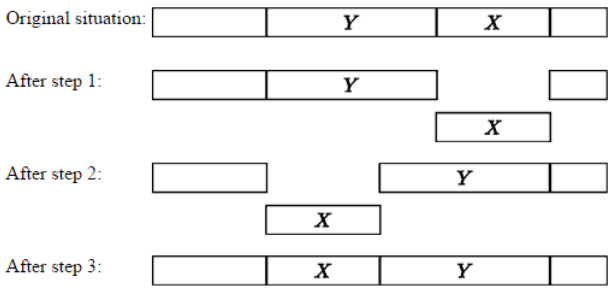
Other students use the unique identification code (which they can find in an online catalogue) to find the books they want to borrow. For them, it is important that the books are really sorted on this code. Also for the librarian, it is important that the books are sorted. It makes it much easier to check if perhaps some books are stolen: not borrowed, but yet missing.

Therefore, every week, the librarian makes a round through the department and sorts the books on every shelf. Sorting one shelf is doable, but still quite some work. The librarian has considered several algorithms for it, and decided that the easiest way for him to sort the books on a shelf, is by *sorting by transpositions*: as long as the books are not sorted,

1. take out a block of books (a number of books standing next to each other),
2. shift another block of books from the left or the right of the resulting 'hole', into this hole,
3. and put back the first block of books into the hole left open by the second block.

One such sequence of steps is called a *transposition*.

The following picture may clarify the steps of the algorithm, where X denotes the first block of books, and Y denotes the second block.



Of course, the librarian wants to minimize the work he has to do. That is, for every bookshelf, he wants to minimize the number of transpositions he must carry out to sort the books. In particular, he wants to know if the books on the shelf can be sorted by **at most 4 transpositions.** Can you tell him?

Input

The first line of the input file contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with one integer n with $1 \leq n \leq 15$: the number of books on a certain shelf.
- One line with the n integers $1, 2, \ldots, n$ in some order, separated by single spaces: the unique identification codes of the n books in their current order on the shelf.

Output

For every test case in the input file, the output should contain a single line, containing:

- if the minimal number of transpositions to sort the books on their unique identification codes (in increasing order) is $T \leq 4$, then this minimal number T;
- if at least 5 transpositions are needed to sort the books, then the message "`5 or more`".

Examples

standard input	standard output
3 6 1 3 4 6 2 5 5 5 4 3 2 1 10 6 8 5 3 4 7 2 9 1 10	2 3 5 or more

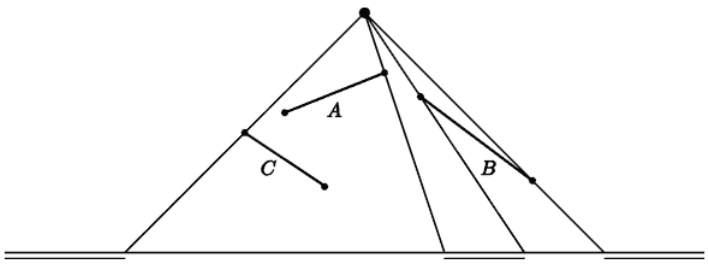
Problem K. Lucky Light

We have a (point) light source at position (x_L, y_L) with $y_L > 0$, and a finite series of line segments, all of finite non-zero length, given by the coordinates of their two endpoints. These endpoints are all different. The line segments are all situated above the x-axis ($y > 0$).

The segments cast their shadows onto the x-axis. We assume that the shadows of two segments either do not overlap at all, or have an overlap that has some non-zero width (they do not just touch). We also assume that for each segment its shadow is more than just one point, i.e., there is no segment that is directed toward the light source. The height of the light source (y_L) is at least 1 unit larger than the y-coordinates of the endpoints of the line segments. This guarantees that indeed each line segment has a bounded shadow on the x-axis.

The collection of shadows divides the x-axis into dark and lighted areas (intervals). The problem is to determine the number of lighted areas — which is at least 2 (if there is at least one line segment, otherwise it is 1).

In the picture below the three line segments A, B and C cause three lighted areas, as indicated.



Input

The first line of the input file contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with one integer n with $0 \leq n \leq 100$: the number of line segments.
- One line with two integers x_L and y_L, the coordinates of the light source, separated by a single space. The coordinates satisfy $-100 \leq x_L \leq 100$ and $1 \leq y_L \leq 1000$.
- n lines, each containing four integers x_i, y_i, u_i and v_i, separated by single spaces, that specify x- and y-coordinates of the two endpoints (x_i, y_i) and (u_i, v_i) of the i-th line segment, where $-100 \leq x_i, u_i \leq 100$ and $0 < y_i, v_i < y_L$, for $1 \leq i \leq n$.

Output

For every test case in the input file, the output should contain a single number, on a single line: the number of lighted areas.

Examples

standard input	standard output
2 3 50 60 55 45 30 35 64 39 92 18 20 30 40 16 2 -10 50 -10 1 10 11 -10 11 10 1	3 2

Notes

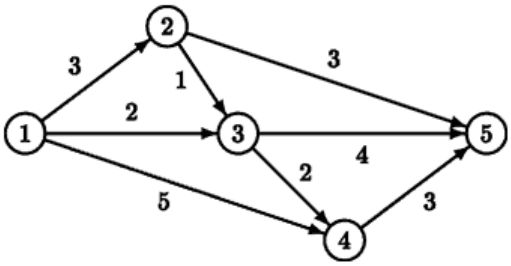
The first test case below corresponds to the picture in the problem description. The second test case has two crossing line segments.

Problem L. Sightseeing

Tour operator Your Personal Holiday organises guided bus trips across the Benelux. Every day the bus moves from one city S to another city F. On this way, the tourists in the bus can see the sights alongside the route travelled. Moreover, the bus makes a number of stops (zero or more) at some beautiful cities, where the tourists get out to see the local sights.

Different groups of tourists may have different preferences for the sights they want to see, and thus for the route to be taken from S to F. Therefore, Your Personal Holiday wants to offer its clients a choice from many different routes. As hotels have been booked in advance, the starting city S and the final city F, though, are fixed. Two routes from S to F are considered different if there is at least one road from a city A to a city B which is part of one route, but not of the other route.

There is a restriction on the routes that the tourists may choose from. To leave enough time for the sightseeing at the stops (and to avoid using too much fuel), the bus has to take a short route from S to F. It has to be either a route with minimal distance, or a route which is one distance unit longer than the minimal distance. Indeed, by allowing routes that are one distance unit longer, the tourists may have more choice than by restricting them to exactly the minimal routes. This enhances the impression of a personal holiday.



For example, for the above road map, there are two minimal routes from $S = 1$ to $F = 5$: $1 \to 2 \to 5$ and $1 \to 3 \to 5$, both of length 6. There is one route that is one distance unit longer: $1 \to 3 \to 4 \to 5$, of length 7.

Now, given a (partial) road map of the Benelux and two cities S and F, tour operator Your Personal Holiday likes to know how many different routes it can offer to its clients, under the above restriction on the route length.

Input

The first line of the input file contains a single number: the number of test cases to follow. Each test case has the following format:

- One line with two integers N and M, separated by a single space, with $2 \leq N \leq 1000$ and $1 \leq M \leq 10000$: the number of cities and the number of roads in the road map.
- M lines, each with three integers A, B and L, separated by single spaces, with $1 \leq A, B \leq N$, $A \neq B$ and $1 \leq L \leq 1000$, describing a road from city A to city B with length L.

  The roads are unidirectional. Hence, if there is a road from A to B, then there is not necessarily also a road from B to A. There may be different roads from a city A to a city B.
- One line with two integers S and F, separated by a single space, with $1 \leq S, F \leq N$ and $S \neq F$: the starting city and the final city of the route.

  There will be at least one route from S to F.

Output

For every test case in the input file, the output should contain a single number, on a single line: the number of routes of minimal length or one distance unit longer. Test cases are such, that this number is at most $10^9 = 1000000000$.

Examples

standard input	standard output
2 5 8 1 2 3 1 3 2 1 4 5 2 3 1 2 5 3 3 4 2 3 5 4 4 5 3 1 5 5 6 2 3 1 3 2 1 3 1 10 4 5 2 5 2 7 5 2 7 4 1	3 2

Notes

The first test case above corresponds to the picture in the problem description.