BENELUX ALGORITHM PROGRAMMING CONTEST
2006 LEIDEN

Qualifying contest

Problem set

September 30, 2006

A Cheesy Chess

Any similarity of this problem to the game Chess is completely coincidental.

Problem

Cheesy Chess is a simple two-person game. It is played on an 8×8 board. Each player has one piece. The players take turns in moving their respective pieces.

The first player, say White, has a king. In one move, it can move one position in any of the eight directions, horizontally, vertically or diagonally, as long as it stays on the board. The second player, say Black, has a pawn. In one move, it can move exactly one position downwards. In fact, the pieces have to make such moves. They may not stay at their positions.

The White king is said to *capture* the Black pawn, if it moves onto the position currently occupied by the pawn. The aim of the White king is to do exactly this. The aim of the Black pawn is to reach the bottom line of the board *safely*. As we will see later, however, there are also other ways for White and Black to win.

The game is complicated by the presence of forbidden fields and dangerous fields. A forbidden field is a position on the board where neither the White king, nor the Black pawn may come. A dangerous field is a position where the Black pawn may come, but where the White king may not move onto.

In addition to the fixed dangerous fields, which are dangerous for the entire game, there are (at most) two other, floating dangerous fields, which depend on the position of the Black pawn. They are adjacent to the pawn's position: the position to the bottom left and bottom right of the pawn, for as far as these positions exist within the boundaries of the board and are not forbidden. All other positions are called open fields, even if they are occupied by either of the pieces.

For example, we may have the following situation, where forbidden fields, dangerous fields and open fields are denoted by `'F'`, `'D'` and `'·'`, respectively, the White king is denoted by `'K'` and the Black pawn is denoted by `'P'`.

·	·	·	·	·	·	·	·
·	·	P	·	·	·	·	D
·	D	·	D	·	·	K	·
·	·	·	·	·	F	·	·
·	·	D	D	D	·	·	·
·	·	D	F	D	D	·	·
·	·	D	D	D	·	·	·
·	·	·	·	·	·	·	·

This illustration does not reveal whether the positions occupied by the White king and the Black pawn are dangerous or open, and whether the dangerous fields adjacent to the position of the pawn are fixed dangerous fields or not.

Due to a move of the Black pawn, the White king's position may become dangerous. This is not a problem: in the next move, the White king has to move to another, open field anyway. The White king *blocks* the Black pawn, if Black is to move, but the position below the pawn is occupied by the White king. In this case, the pawn cannot move.

The game ends, when

- the White king captures the Black pawn; in this case, White wins;
- the White king is to move, but cannot move to an open field; in this case, Black wins;
- the Black pawn is to move, but cannot move to an open field or a dangerous field; if the pawn is at the bottom line of the board, then Black wins, otherwise White wins.

You have to find out which player will win, given that White is the first player to move and given that White plays optimally.

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:

- A description of the board, consisting of 8 lines, corresponding to the 8 lines of the board, from top to bottom. Each line contains a string of 8 characters from {`'F'`, `'D'`, `'.'`}. Here, `'F'` denotes a forbidden field, `'D'` denotes a fixed dangerous field and `'.'` (a period) denotes an open field.

  Of course, an open field may become dangerous due to the position of the Black pawn.
- One line with two integers x_K and y_K ($1 \leq x_K, y_K \leq 8$), separated by a single space, specifying the initial position of the White king. Here, x_K denotes the column (counted from the left) and y_K denotes the row (counted from below).

  This initial position is neither a forbidden field, nor a fixed dangerous field.
- One line with two integers x_P and y_P ($1 \leq x_P, y_P \leq 8$), separated by a single space, specifying the initial position of the Black pawn. Here, x_P denotes the column (counted from the left) and y_P denotes the row (counted from below).

  This initial position is not a forbidden field, and is different from the initial position of the White king.

Output

For every test case in the input file, the output should contain a single line containing the string `"White"` (if White wins) or `"Black"` (if Black wins).

Example

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

Input

```
2
........
.......D
........
.....F..
..DDD...
..DFDD..
..DDD...
........
7 6
3 7
........
........
........
........
........
........
........
........
3 1
6 3
```

Output

```
Black
White
```

B Frobenius

Problem

The *Frobenius problem* is an old problem in mathematics, named after the German mathematician G. Frobenius (1849–1917).

Let $a_1, a_2, \ldots, a_n$ be integers larger than 1, with greatest common divisor (gcd) 1. Then it is known that there are finitely many integers larger than or equal to 0, that cannot be expressed as a lineair combination $w_1a_1 + w_2a_2 + \ldots + w_na_n$, using integer coefficients $w_i \geq 0$. The largest of such nonnegative integers is known as the ***Frobenius number*** of $a_1, a_2, \ldots, a_n$ (denoted by $\mathrm{F}(a_1, a_2, \ldots, a_n)$). So: $\mathrm{F}(a_1, a_2, \ldots, a_n)$ is the largest nonnegative integer that cannot be expressed as a nonnegative integer linear combination of $a_1, a_2, \ldots, a_n$.

For $n = 2$ there is a simple formula for $F(a_1, a_2)$. However, for $n \geq 3$ it is much more complicated. For $n = 3$ only for some special choices of a_1, a_2, a_3 formulas exist. For $n \geq 4$ no formulas are known at all.

We will consider here the Frobenius problem for $n = 4$. In this case our version of the problem can be formulated as follows. Let four integers a, b, c and d be given, with $a, b, c, d > 1$ and gcd(a, b, c, d) = 1. We want to know two things.

- How many nonnegative integers less than or equal to $1,000,000$ cannot be expressed as a nonnegative integer linear combination of the values a, b, c and d ?
- Is the Frobenius number of a, b, c and d less than or equal to 1,000,000 and if so, what is its value?

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, containing four integers a, b, c, d (with $1 < a, b, c, d \leq 10,000$ and gcd(a, b, c, d) = 1), separated by single spaces.

Output

For every test case in the input file, the output should contain two lines.

- The first line contains the number of integers between 0 and 1,000,000 (boundaries included) that cannot be expressed as $a \cdot w + b \cdot x + c \cdot y + d \cdot z$, where w, x, y, z are nonnegative (meaning ≥ 0) integers.
- The second line contains the Frobenius number if this is less than or equal to 1,000,000 and otherwise -1, meaning that the Frobenius number of a, b, c and d is larger than 1,000,000.

Example

Input

```
3
8 5 9 7
5 8 5 5
1938 1939 1940 1937
```

Output

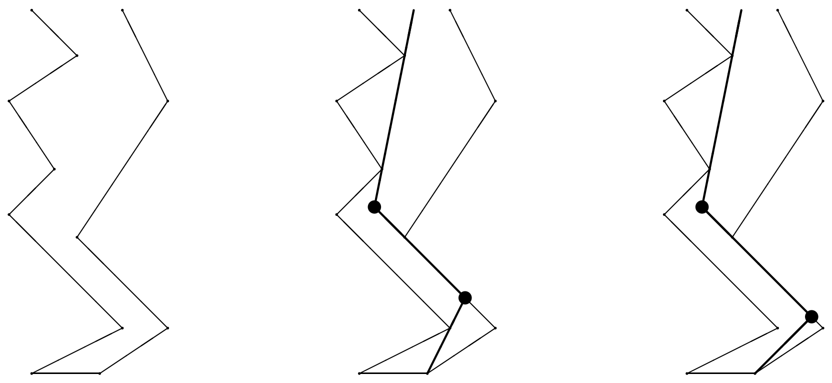
```
6
11
14
27
600366
-1
```

C Mineshaft

Problem

A plumber has been hired to build an air pipe through a mineshaft from the bottom to the surface. The mineshaft was built without modern technology, so it winds its way up through the earth. Because it is very time consuming to bring the tools necessary to bend the pipe below the surface, the plumber wants to minimize the number of bends in the pipeline.

For example, for the mineshaft in the first picture below, the minimal number of bends in a pipeline from the bottom to the surface is two. Different optimal solutions exist, one of which is shown in the second picture. The bullets indicate the bends in the pipeline.



The two walls of the mineshaft are formed by sequences of straight segments. The numbers of segments in the two sequences may be different. Further, the horizontal distance between the walls of the mineshaft may vary, but is always positive. Both walls start at the same level and end at the same level.

On the way from the bottom of the mineshaft to the surface, the level (the y-coordinate) increases with every segment of a wall. Hence, the mineshaft does not have horizontal plateaus or 'ceilings', and at no point does it go back down again.

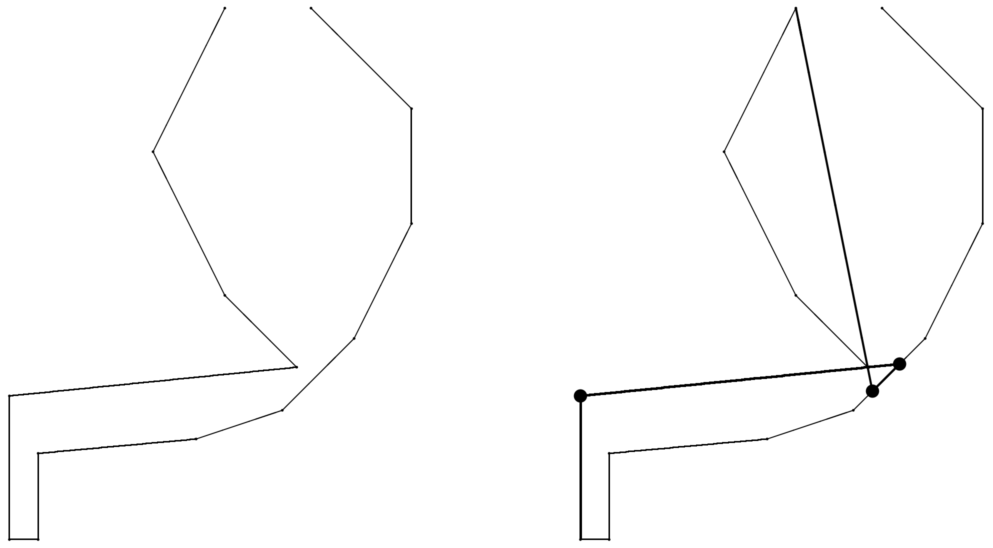
For the purpose of this task, you may assume the diameter of the pipeline to be 0. At no point may the pipeline cross the walls. In order to attach the pipeline firmly to the wall, each segment of the pipeline has to touch the walls at (at least) two different places. However, the bending points of the pipeline are weak. They cannot be used to attach the pipeline to the walls. The end points of the pipeline, though, at the bottom and the top of the mineshaft, may be used to attach a segment to the walls.

Hence, the solution in the third picture above (also having two bends) is not allowed, because the lowest segment of the pipeline can be attached to the walls at only one place: at the bottom of the right wall.

The pipeline must start anywhere at the bottom of the mineshaft, and must end anywhere on the imaginary line between the top of the left wall and the top of the right wall. Note, however, that the endpoints of the pipeline may only be used to attach the pipeline, if they touch a wall. In particular, the endpoint at the bottom cannot be attached to any position at the bottom which is not the bottom of a wall.

Finally, the angle that the pipeline makes at a bending point can take any value α satisfying $-180^\circ < \alpha < 180^\circ$ and (of course) $\alpha \neq 0$.

Note that sometimes it may be useful to have the pipeline intersect with itself. For example, in the mineshaft below, this is needed to get from the bottom to the top of the mineshaft with only three bends.



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_1 ($2 \leq N_1 \leq 25$): the number of points describing the left wall of the mineshaft.
- N_1 lines with the coordinates of the points describing the left wall of the mineshaft, from the bottom to the top. The i^{th} line contains two integers x_i and y_i ($-1,000 \leq x_i \leq 1,000$ and $0 \leq y_i \leq 1,000$) separated by a single space: the x- and y-coordinate of the point, respectively.

  The y-coordinates are monotonically increasing: $y_1 < y_2 < \cdots < y_{N_1}$.
- One line with one integer N_2 ($2 \leq N_2 \leq 25$): the number of points describing the right wall of the mineshaft.
- N_2 lines with the coordinates of the points describing the right wall of the mineshaft, from the bottom to the top. The i^{th} line contains two integers x'_i and y'_i ($-1,000 \leq x'_i \leq 1,000$ and $0 \leq y'_i \leq 1,000$) separated by a single space: the x- and y-coordinate of the point, respectively.

  The y-coordinates are monotonically increasing: $y'_1 < y'_2 < \cdots < y'_{N_2}$.

We always have $x_1 < x'_1$, $y_1 = y'_1$, $x_{N_1} < x'_{N_2}$ and $y_{N_1} = y'_{N_2}$. The walls described by the sequences of points do not cross or even touch each other.

Output

For every test case in the input file, the output should contain a single number, on a single line: the minimum number of bends in the pipeline to make it from the bottom of the mineshaft to the top, under the conditions from the problem description.

Example

These testcases correspond to the pictures in the problem description.

Input

```
2
7
4 0
8 2
3 7
5 9
3 12
6 14
4 16
5
7 0
10 2
6 6
10 12
8 16
6
-10 10
-10 20
10 22
5 27
0 37
5 47
8
-8 10
-8 16
3 17
9 19
14 24
18 32
18 40
11 47
```

Output

```
2
3
```

D Colour sequence

Problem

We have a pile of cards consisting of 100 cards that are coloured on both sides. There is a finite number of colours (at most 26). In addition there are special cards called jokers. Jokers have a joker sign on both sides, which can assume any of the possible colours. We consider here a one-player card game, in which the player is challenged to derive a given colour sequence from a given row of cards, following certain rules.

Before the actual beginning of the game a colour sequence S of length at most 100 (not containing a joker) is given. Furthermore a number of cards are chosen from the pile and are put in a row. The sides turned upwards form a row of colours. Now the aim for the player is to create the colour sequence S with the cards from the row in the following way. For each card in the row the player decides whether or not to turn it over. When the card is turned over, only the colour on the other side is visible. Jokers may be part of the row of cards.

These steps lead to the final sequence of colours formed by the visible side of the cards in the row. If the player has been able to turn the cards in such a way that the pre-given colour sequence S is contained (from left to right) in the final row of colours, the player wins. If not, he loses. In matching the pre-given colour sequence to the row, cards in the row may be skipped, as long as the relative order of the colours is preserved. A joker can assume any colour. For example, the colour sequence (red, blue, yellow) is contained in (green, joker, blue, red, yellow), and (blue, green, blue, green) is contained in (red, blue, joker, yellow, joker, blue, green, green).

Your job is to find out if the player can win, given the colour sequence S and the row of cards chosen from the pile. This means that the sequence of colours that are face up is known, and so are the colours on the other side of these cards.

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 describing the colour sequence S. This line contains a string of m (with $1 \leq m \leq 100$) characters from the set {`'A'`, `'B'`, ..., `'Z'`}, denoting the colours. Different colours correspond to different characters. For example: `"BGBG"` denotes the sequence blue, green, blue, green.

- Two lines, corresponding to the row of cards chosen from the pile. Each of these lines contains a string of k ($1 \leq k \leq 100$) characters from the set {`'A'`, `'B'`, ..., `'Z'`, `'*'`}. The character `'*'` denotes a joker, which can play the role of any of the possible colours.

  The string in the first line corresponds to the row of colours on the visible side of the cards. The string in the second line corresponds to the row of colours on the hidden side of the cards.

  So for the i^{th} card in the row, the first line gives the colour of the side turned upwards and the second line shows the colour of the side face down. Obviously the strings on both lines have the same length. Furthermore, a `'*'` in one line (denoting a joker) always corresponds to a `'*'` in the other line at the corresponding position.

Output

For every test case in the input file, the output should contain one line. This line contains `"win"` if the colour sequence S can be achieved by the player by turning the right cards upside down, and `"lose"` if this is not the case.

Example

Input

```
3
RBY
B*RRB
G*BRY
BGBG
RZ*Y*PGG
AB*Y*BCB
BAPC
BUBCDAPVDAVVDLPF
VLDCUSPGLSGPPVDD
```

Output

```
win
win
lose
```

E Projects

Problem

In a certain week, a company wants to finish m projects. To this end, the company can employ at most n people from the unemployment agency for a period of one week. Each external employee will cost the company `salary` euro, unless the project in which he/she is involved is not completed in time. In that case no payment is due.

For each project the company knows from experience the probability that the project will be completed within a week, as a function of the number of employees working on it. These probabilities are given as percentages p_{ij}, where i (with $1 \leq i \leq m$) is the number of the project and j is the number of people working on it. Of course, when nobody is working on a project i, the probability p_{i0} is zero percent.

If project i is indeed finished within a week, the company earns `reward`(i) euro; if it is not ready in time, the company has to pay a fine of `punishment`(i) euro.

Of course the company wants to maximise its total expected profit[1] at the end of the week by finding the optimal number of external employees to hire, and how to divide them over the projects. The optimal number of employees is the total number of people needed to achieve the maximal expected profit. Your task in this matter is to calculate this optimal number of external employees. Remember that at most n people are available. Furthermore: if a person is employed, he/she works on one and only one project.

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 m with $1 \leq m \leq 100$: the number of projects.
- One line with one integer n with $0 \leq n \leq 100$: the maximal number of available employees.
- One line with one integer `salary` with $0 \leq$ `salary` $\leq 1,000$: the salary of one employee. Remember that the salary is given in euros.
- m lines, each line corresponding to a project i, containing n integers $p_{i1}, p_{i2}, \ldots, p_{in}$ (the percentages, with $0 \leq p_{i1}, p_{i2}, \ldots, p_{in} \leq 100$), followed by two integers corresponding to the `reward` and the `punishment` for project i. All values are separated by single spaces. Both `reward` and `punishment` are given in euros and are between 0 and 100,000 (boundaries included).

Output

For every test case in the input file, the output should contain two lines.

- The first line contains the maximal expected profit in eurocents.
- The second line contains the total number of external employees that must be hired in order to achieve this maximal expected profit. If the maximal expected profit can be achieved by different (total) numbers of employees, then these different numbers must be given in increasing order. Numbers have to be separated by single spaces.

[1]Let p ($0 \leq p \leq 1$) be the probability that a job is finished in time, and let E_1 be the profit in that case. Furthermore, let E_2 be the (negative) profit in case the job is not finished in time. Then the expected profit for this particular job is $p \cdot E_1 + (1 - p) \cdot E_2$

Example

Input

```
3
1
4
200
90 100 100 100 2000 0
2
2
100
80 80 2100 500
0 100 1700 500
3
4
100
100 80 80 70 1000 100
100 90 80 90 500 50
100 70 60 50 700 100
```

Output

```
162000
1
100000
1 2
190000
3
```

F Booksort

Problem

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.

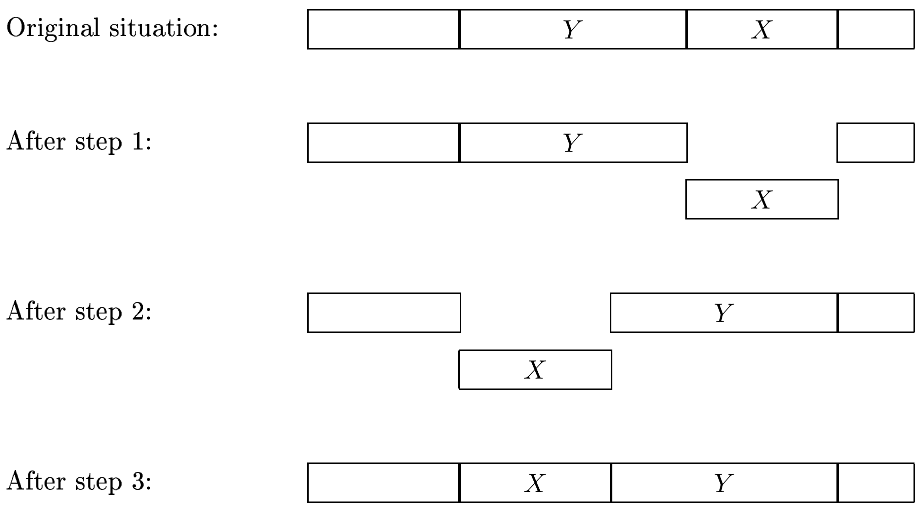
Other students use the unique identification code (which they can find in an online catalogue) to find the books they want to borrow. For them, it is important that the books are really sorted on this code. Also for the librarian, it is important that the books are sorted. It makes it much easier to 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"`.

Example

Input

```
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
```

Output

```
2
3
5 or more
```

G Oulipo

Problem

The French author Georges Perec (1936–1982) once wrote a book, La disparition, without the letter `'e'`. He was a member of the *Oulipo* group. A quote from the book:

> Tout avait l'air normal, mais tout s'affirmait faux. Tout avait l'air normal, d'abord, puis surgissait l'inhumain, l'affolant. Il aurait voulu savoir où s'articulait l'association qui l'unissait au roman : sur son tapis, assaillant à tout instant son imagination, l'intuition d'un tabou, la vision d'un mal obscur, d'un quoi vacant, d'un non-dit : la vision, l'avision d'un oubli commandant tout, où s'abolissait la raison : tout avait l'air normal mais ...

Perec would probably have scored high (or rather, low) in the following contest. People are asked to write a perhaps even meaningful text on some subject with as few occurrences of a given "word" as possible. Our task is to provide the jury with a program that counts these occurrences, in order to obtain a ranking of the competitors. These competitors often write very long texts with nonsense meaning; a sequence of 500,000 consecutive `'T'`s is not unusual. And they never use spaces.

So we want to quickly find out how often a word, i.e., a given string, occurs in a text. More formally: given the alphabet {`'A'`, `'B'`, `'C'`, ..., `'Z'`} and two finite strings over that alphabet, a word W and a text T, count the number of occurrences of W in T. All the consecutive characters of W must exactly match consecutive characters of T. Occurrences may overlap.

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 the word W, a string over {`'A'`, `'B'`, `'C'`, ..., `'Z'`}, with $1 \leq |W| \leq 10,000$ (here $|W|$ denotes the length of the string W).
- One line with the text T, a string over {`'A'`, `'B'`, `'C'`, ..., `'Z'`}, with $|W| \leq |T| \leq 1,000,000$.

Output

For every test case in the input file, the output should contain a single number, on a single line: the number of occurrences of the word W in the text T.

Example

Input	Output
`3`	`1`
`BAPC`	`3`
`BAPC`	`0`
`AZA`	
`AZAZAZA`	
`VERDI`	
`AVERDXIVYERDIAN`	

H Lucky Light

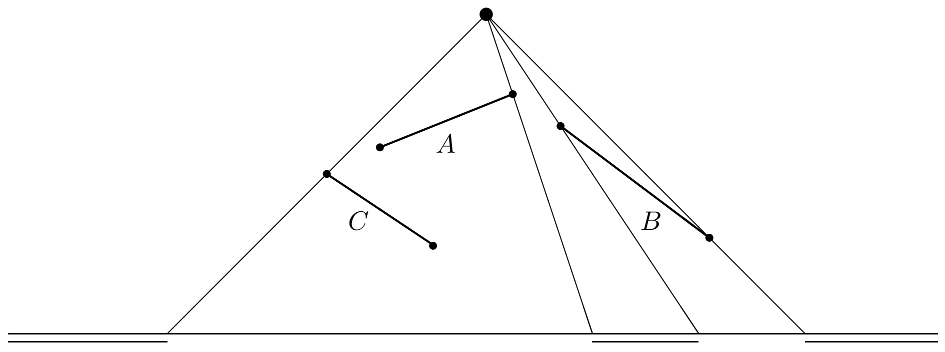
Problem

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 $0 \le n \le 100$: 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 $-100 \le x_L \le 100$ and $1 < y_L \le 1,000$.
- 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 $-100 \le x_i, u_i \le 100$ and $0 < y_i, v_i < y_L$, for $1 \le i \le n$.

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.

Example

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

Input

```
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
```

Output

```
3
2
```

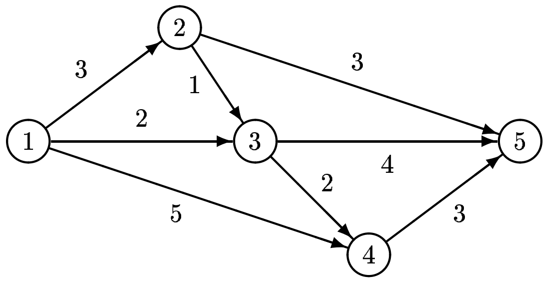
I Sightseeing

Problem

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 $2 \leq N \leq 1,000$ and $1 \leq M \leq 10,000$: 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 $1 \leq L \leq 1,000$, 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 $10^9 = 1,000,000,000$.

Example

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

Input

```
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
```

Output

```
3
2
```