A. All-Out Arctic Warfare

In the cold regions of the Arctic circle lives a strange[1] kind of penguins. These penguins breed like rabbits and as a result suffer from overpopulation.

The two penguin tribes of the northern Drygalski Island have found a good solution for this dire situation: each year they attack each other in a game they call 'All-Out Arctic Warfare'. This game of warfare is unlike any human war, and is bound to strict rules.

At the eve of battle the 'surplus' members of the two tribes gather in the War Zone. There they line up behind their colleagues, the opposing tribes facing each other. Then, as the sun sets for the first time in the New Year, they start to take turns attacking. The penguin at the front of a line assails the opponent in front of him, either by breaking his legs, poking out his eyes, or by shooting him through the head with a 9mm semi-automatic pistol. All these options result in the same outcome, the immediate demise of his opponent.

After centuries of playing this game, several subspecies of penguin have developed. A recent scientific expedition discovered that all penguins on Drygalski Island fall into one of three classes:

1. Pygoscelis warfarum rockus, which has a hard exoskeleton.
2. Pygoscelis warfarum papyrus, which has huge foldable wings.
3. Pygoscelis warfarum scissorum, which has razor sharp fins.

Each subspecies has a special tactic during battles. When a P. rockus attacks a P. scissorum he can use his exoskeleton to effortlessly shatter the sharp wings of his opponent. Similarly, a P. papyrus can attack a P. rockus by smothering his opponent with his large wings, and a P. scissorum can use his sharp fins to slice right through the P. papyrus' wings. In each case the thrill of this victory compels the penguin to immediately attack the next opponent as well.

On the other hand, when a penguin faces an opponent who cannot be slain with such a specialized attack, he instead shoots him through the head. For this purpose each penguin carries a small pistol containing a single ice bullet. After this final move he becomes defenceless, and has to yield his turn to the enemy. In a typical round of battle, a Pygoscelis will defeat zero or more opponents in quick succession by having a natural advantage over them, and then shoot the first opponent he meets against which he has no such advantage.

All-Out Arctic Warfare continues in this way until no penguins remain standing on one of the sides. The surviving tribe is declared the winner of the games, and receives five highly valued ice cubes and a square kilometer of territory from the losing side.

This year a unique situation has developed in the Arctic Circle: a penguin in one of the tribes, The Tux alliance, has observed the line of penguins from the opposing Devil tribe marching towards the War Zone. This year it is The Tux alliance's turn to attack first which allows them to pick an optimal line-up for the coming engagement.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

[1] These penguins are really quite strange, if only because all other penguins live on the southern hemisphere.

- A line with a string of n characters ($0 < n < 10^6$), the penguins in the Devil tribe line-up starting from the front. The character `'R'` stands for P. rockus, `'P'` stands for P. papyrus and `'S'` stands for P. scissorum.

Output

For each test case:

- One line containing a single number, the minimal number of penguins The Tux alliance needs to put in the line to ensure that his tribe wins this year's All-Out Arctic Warfare.

Example

Input	Output
3 RPS SSSPP PPPRPPP	2 3 2

B. Hedge Maze Relay Race

Mazes are a popular element in fairy tales as well as in modern sporting events. The Hedge Maze Relay Race is one of these events, combining cunning and insight with speed and endurance. The goal of this race is for a team to traverse a maze as quickly as possible. All contestants start in the center of the maze with a single baton per team. One team member carries this baton and may pass it on to other team members. To win the race a single team member has to be outside the maze, holding the baton.

This year's race is organized in the Groningen FloraFun Gardens$^{\text{TM}}$. The local maze is of the 'square' type: it is laid out in a grid with patches of grass, each one square meter in size, separated by hedges of the same size. The thing that makes this maze Fun$^{\text{TM}}$ is the variation in hedges. Besides your garden variety hedge, there are also hedges with a gate, allowing children and small animals to crawl through them. Finally, to retain the minimum legal density of Flora$^{\text{TM}}$ there are also some flower beds in the maze. Children don't dare cross these flower beds (they might get dirty!) but adults can safely step over them. The adults can even cross a series of flower beds and the children and small animals can crawl also under a series of hedges that have a gate.

The Hedge Maze Relay Race is an *everything goes* kind of race: Team members can drop the baton, pick it up, throw it and hit each other over the head with it. Everyone is allowed to move at the same time (or not), up to a speed limit of $1\,{}^{m}\!/_{s}$. Your team has been randomly assigned the blue baton, which means that you get a sneak peak at a map of the maze, which should allow you to plan a good route.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line containing two positive integers, h_g, h_f, the maximum height for a person to be able to use a small gate and the minimum height for a person to be able to step over a flower bed, both in centimeters.
- A line containing a single positive integer $n < 100$, the size of your team.
- n lines, each containing a single positive integer, the height of each team member in centimeters.
- A line containing two odd integers $r, c < 10^3$, the number of rows and columns in the maze.
- r lines each containing c characters, representing the maze. '`#`' represents a hedge, '`^`' represents a hedge with a gate, '`%`' represents a flower bed, and '`.`' represents grass.

Output

For each test case:

- One line containing a single integer, the minimal time it takes for your team to finish the race. There is always a way to finish the race.

Example

Input	Output
2 150 180 1 170 9 9 ######### ....#.... ###.#.### #...%...# #.##.##.# #.#...^.# #.#.#.#.# #...#...# ######### 150 180 1 190 9 9 ######### ....#.... ###.#.### #...%...# #.##.##.# #.#...^.# #.#.#.#.# #...#...# #########	18 8

C. Brick Stacking

During their lunch hour a group of paviors likes to play a game of brick stacking, using leftover bricks which have uniform density and depth, but vary in width and height. The object of this game is to create a large construction of bricks, referred to as a 'stack'. The only restriction on stacks is that each brick touches at most one other brick (not counting bricks on top of it). Bricks can also be placed on the ground at $y = 0$.

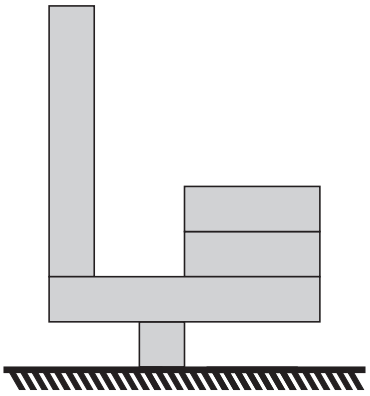

Figure 1: A stable stack in a game of brick stacking.

The one with the highest stack gets his lunch for free!

After a couple of years of playing this game the paviors have become so good at this game, and thus their stacks so high, that the wind has begun to interfere with the judgings. Frequently they argue about whether or not a fallen stack was in fact stable (and it was just the wind that caused it to fall) or not. Clearly they need a better way of judging their stacks.

Can you help these paviors in need by creating a program which will determine whether or not the given stack(s) are stable?

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line with a single integer $n < 10^5$, the number of bricks in the stack.
- n lines, each with four integers x, y, w, h, the position ($0 \leq x, y < 10^9$) of the bottom-left corner of the brick and the size ($0 \leq w, h < 10^9$, $wh < 10^9$) of the brick.

Output

For each test case:

- One line describing whether the stack is stable: either "`stable`" if everything is stable or "`collapsed`" if some part will collapse.

Example

Input	Output
4 5 0 2 1 6 3 2 3 1 3 3 3 1 2 0 1 1 0 1 5 1 2 0 0 4 1 1 1 20 1 2 1 1 1 1 1 2 2 1 2 1 0 1 1 1 1 2 1	stable collapsed collapsed stable

The first example corresponds to figure 1.

D. Family Politics

Each year the Calzone family goes on an outing together. Like all traditional Italian families it is a very large family, which makes it almost impossible to reach an agreement. After weeks of intense debate, the choice has been narrowed down to two possible destinations: the *Interesting Topic Museum* and the *Happy Fun Beach*. Now the final choice of the outing is put to a vote.

The adults all prefer to go to the more interesting museum. But political considerations are more important than this preference: Each adult family member x has a number of younger relatives he/she resents, and if all these relatives agree on a particular choice, x *must* vote in the opposite way to ensure his place in the social order. When there is no such agreement, x can (and therefore will) vote for the museum without appearing weak.

On the other hand children don't like museums, and would prefer to go to the beach. They are also too young for politics and instead they will cooperate. By a combination of bribes, extortion and cute faces the children have managed to determine the entire political landscape. Now by voting strategically they will try to manipulate the Calzone family.

Long ago, it was deemed unfair to give everyone an equal vote. So now votes are weighed proportionally by age. While that sounds nice in theory, in practice it means that the oldest family member ('Gran') always casts the deciding vote. Plans to assassinate the ancient leader have thus far failed for lack of consensus, so for now the Calzone family is stuck with her.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line with two positive integers $a \leq 256$ and $c \leq 16$, the number of adults and children in the Calzone family.
- a lines, each containing an integer n_i ($0 < n_i \leq a + c$) and a list of n_i numbers, the family members adult i resents. Number 0 refers to the oldest family member, 1 to the second oldest, etc. Number a refers to the oldest child, $a + 1$ to the second oldest, etc.

Output

For each test case:

- One line containing the destination of the outing, either "`museum`" or "`beach`".

Example

Input	Output
2 2 1 2 1 2 1 2 3 3 2 3 3 2 2 3 3 3 4 5	museum beach

E. Box City

Box City offers storage facilities to the public. Anyone can store their stuff in one or more boxes for a small monthly fee per box. Because people in Western societies usually have lots of stuff, their services are very popular. Potential customers arrive almost daily with cartloads full of stuff they want to get rid of without throwing it away.

Of course these customers want their stuff safely stored as cheaply as possible. However, finding the optimal way of packing a bunch of stuff into boxes is an NP-Hard problem. This gives Box City a convenient excuse for not offering such an optimization service.

Instead all items are packed into storage boxes by following a simple procedure. The first item is placed as far in the back and to the left as possible (moving it to the back is more important). Then the next item is placed in the same way, then the third item, etcetera. This is repeated until an item no longer fits. Then the storage box is sealed and a new one is opened. Furthermore, to comply with regulations set by the fire department, a box with a hundred items must also be sealed. Items are never rotated or stacked, and they are placed into the boxes from above.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line with three integers $d, w < 10^4$ and $n < 10^5$. The depth and width of each box in millimeters, and the number of items a customer wants to store.
- n lines, each containing two positive integers $d_i \leq d$ and $w_i \leq w$, the size of each item in millimeters.

Output

For each test case:

- A line containing a single integer b, the number of storage boxes used to store the customer's stuff.
- A line containing b integers separated by spaces, the number of items in each box.

Example

Input	Output
3 10 10 1 3 4 4 4 5 2 2 1 3 1 2 3 1 2 2 3 4 6 3 1 1 1 3 1 1 2 2 2 2 1	1 1 2 4 1 2 3 3

F. Box Village

Box Village offers storage facilities to the public. Anyone can store their stuff in one or more boxes for a small monthly fee per box. Because people in Western societies usually have lots of stuff, their services are very popular. Potential customers arrive almost daily with cartloads full of stuff they want to get rid of without throwing it away.

Of course these customers want their stuff safely stored as cheaply as possible. However, finding the optimal way of packing a bunch of stuff into boxes is an NP-Hard problem. This gives Box Village a convenient excuse for not offering such an optimization service.

Instead only rectangular items of the same size are allowed together in a box. So storing a collection of stray cats is okay (as long as they are rectangular), but storing both apples and bananas is not. Items are never rotated or stacked, and they are placed into the boxes from above.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line with five positive integers $d, w < 10^9$, $d_1 \leq d, w_1 \leq w$ and $n < 10^5$. The depth and width of each storage box, the depth and width of a single item and the number of items the customer wants to store, respectively.

Output

For each test case:

- One line containing a single integer, the number of storage boxes used to store the customer's stuff.

Example

Input	Output
4 100 100 10 10 500 3 4 3 1 4 25 25 5 5 25 25 25 5 5 26	5 1 1 2

G. Word Search Puzzle

Many puzzles with prizes can be found in magazines these days, and several websites are dedicated to indexing them so that anyone can participate. With the financial recession, any prize would of course be very helpful, and solving many puzzles would greatly increase your chances of winning. However, solving all of these puzzles takes a lot of time and can be awfully boring, so automating this process would drastically improve your financial situation.

Solving word search puzzles is your first priority. These puzzles consist of a grid of letters and a small dictionary of words. Words from this dictionary can appear any number of times in the puzzle. Words can appear forwards and backwards, in all the eight ways shown in figure 2.

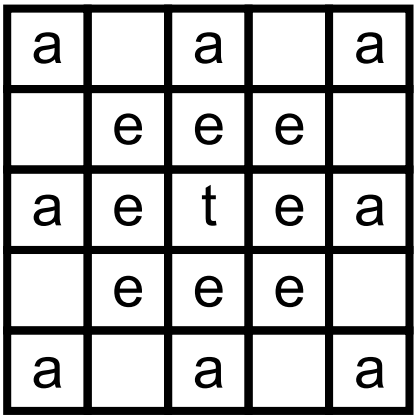

Figure 2: Illustrating the eight directions in which the word "tea" can be matched in a word search puzzle.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line with one integer $s < 10^5$, the number of words in the dictionary.
- s lines with a single dictionary word each. Each word consists of at least 1 and at most 100 lowercase characters.
- A line with two integers $h, w < 500$, the height and width of the puzzle.
- h lines with w characters each, the word search puzzle.

Output

For each test case:

- One line containing a single string, the letters not part of any word, in the order they appear in the input.

Example

Input	Output
1 7 spielberg like tree anywhere mouse eggs free 5 5 ekile eggse reugr fotgf mfree	et

H. Blueprint

Most architects are rather smart fellows. They work meticulously and take care of even the tiniest details. Buildings designed by such architects are well suited to the needs of their inhabitants, and are in general a pleasure to live and work in.

Some architects, however, are even more absent-minded than your average college professor. The buildings they design sometimes fall down, or at least lean over a bit. At other times an extra floor is added that is not in the original requirements. There was also an extraordinary case a couple of years ago where a house was built with 20 meter high doors, because of a measurement error.

Today it looks like such a situation might occur again. As you may know, blueprints are usually annotated with a scale factor. For example, scale 1 : 25 means that one centimeter on the blueprint corresponds to 25 centimeter in real life. But the architect of the postmodern housing complex "*Tesseract*" has forgotten to include a scale factor altogether! The construction workers now have a beautiful blueprint before them, but no idea how large everything should be.

Fortunately the architect has already communicated with several subcontractors, and amongst other things the right building materials have already been ordered. The revolutionary design of the complex is also very simple: all walls, floors and ceilings are made out of transparent aluminum and fiberglass. This construction allows for infinitesimally thin structures that are seamlessly fused together.

The "*Tesseract*" complex is only one story tall, 3 meters. The exterior walls can be described by a simple polygon, and there are no interior walls. With this information it should be possible to recover the missing scale factor.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line containing a positive floating point number, the total amount of construction material used in square meters.
- A line containing a positive integer $n < 10^6$, the number of exterior walls in the building.
- n lines, each containing two integers x_i and y_i ($|x_i|, |y_i| < 10^6$), the coordinates of each corner of the exterior walls on the blueprint in centimeters. The points are given in counterclockwise order.

Output

For each test case:

- One line containing the text "1:s", the scale factor of the blueprint rounded to the nearest integer.

Example

Input	Output
3 14 4 0 0 100 0 100 100 0 100 4000 4 0 0 100 0 100 100 0 100 108 5 6 0 12 8 12 14 8 14 0 8	1:1 1:42 1:50

I. Typechecker

Compilers are among the most complex pieces of software. The problem is that the input can contain programming mistakes such as syntax errors. However, syntax errors are just the beginning, much harder to handle are typing errors.

Consider a simplified version of the Java programming language, consisting of declarations and statements. These are specified by the following EBNF[2] grammar:

$$
\begin{aligned}
\langle \text{LETTER} \rangle &::= \text{“\_”} \mid \text{“A”} \mid \cdots \mid \text{“Z”} \mid \text{“a”} \mid \cdots \mid \text{“z”} \mid \text{“0”} \mid \cdots \mid \text{“9”} \\
\langle \text{NAME} \rangle &::= \langle \text{LETTER} \rangle \langle \text{LETTER} \rangle^* \\
\langle \text{TYPE} \rangle &::= \langle \text{NAME} \rangle \\
\langle \text{DECL} \rangle &::= \langle \text{TYPE} \rangle \ \langle \text{NAME} \rangle \ \text{“(”} \ \langle \text{PARAMLIST} \rangle \ \text{“)”} \ \text{“;”} \\
\langle \text{PARAMLIST} \rangle &::= \textit{empty} \mid \langle \text{PARAM} \rangle \ \big(\text{“,”} \ \langle \text{PARAM} \rangle\big)^* \\
\langle \text{PARAM} \rangle &::= \langle \text{TYPE} \rangle \ \langle \text{NAME} \rangle \\
\langle \text{STMT} \rangle &::= \langle \text{EXPR} \rangle \ \text{“;”} \\
\langle \text{EXPR} \rangle &::= \langle \text{NAME} \rangle \ \text{“(”} \ \langle \text{EXPRLIST} \rangle \ \text{“)”} \\
\langle \text{EXPRLIST} \rangle &::= \textit{empty} \mid \langle \text{EXPR} \rangle \ \big(\text{“,”} \ \langle \text{EXPR} \rangle\big)^*
\end{aligned}
$$

All nonterminals except for ⟨LETTER⟩ can be separated and surrounded by spaces. There is always at least one space character between two ⟨NAME⟩s.

In this simple language there are only function declarations and function call statements. Each function has a result type and zero or more parameters. For example, division could be done with a function

```
int div(int dividend, int divisor);
```

Just like in Java it is only allowed to call a function if a function with a matching name is declared that has the right number and types of parameters (although Java is a little more liberal, in this language the types must match exactly). Also just like Java, this simple language is case sensitive.

A program that contains an invalid function call is said to contain a *type error*. The compiler should of course be able to tell when a program contains such an error, before continuing. Otherwise it might generate invalid machine instructions, or reference non-existing memory locations.

Input

On the first line of the input is a positive integer, the number of test cases. Then for each test case:

- A line containing two positive integers $n_d, n_s < 10^6$, the number of declarations and the number of statements respectively.
- n_d lines, each containing a single declaration, i.e. a ⟨DECL⟩. Each line will be at most 5000 characters long.
- n_s lines, each containing a single statement, i.e. a ⟨STMT⟩. Each line will be at most 5000 characters long.

[2]Extended Backus-Naur Form. The notation x^* stands for zero or more of copies of x.

Output

For each test case:

- For each statement, a single line containing the text "`error`" if the statement contains a type error, and "`ok`" otherwise.

Example

Input	Output

1
4 6
int add(int a, int b);
int mul(int a, int b);
void println(String text);
int i();
add(i(),i());
mul(i());
println(i());
add(add(),add());
j();
i(i());
	ok
	error
	error
	error
	error
	error