CSULA PROGFEST 2014
Problem 1
The Sorrows of Young Jack

Spring break is approaching. Jack, a huge film buff, is having trouble deciding what movies to watch during the break. Your job is to help Jack with his decisions by organizing the potential movies in Jack's radar for him. Jack gives you a list of movies, each with the following fields:

- Title
- Genre(s)
- Audience Rating
- Critic Rating

Your program should be able to group the movies based on the field Jack specifies, and within those groups sort the movies by the following criteria:

- First, sort by Audience Rating in descending numerical order
- then sort by Critic Rating in descending numerical order
- then sort by Genre in descending alphabetical order
- then sort by Title in descending alphabetical order
- If a movie contains multiple genres, sort by Title instead. You can assume Title is unique for all movies since Jack despises remakes of classics.

After grouping and sorting the movies, you should then show the field values that the movies are grouped on, sorted by the order specified by Jack(ascending or descending), and under each field values show the list of movies with matching field values, sorted by the criteria. When comparing field values, ignore all cases(i.e. "comedy" will be considered equal to "Comedy"). Convert the genres to all lowercase in the output. Some movies could be under more than one genre: the genre's will be listed one by one on the same line, separated by a comma. When grouping by genre, if a movie fits into more than one genre, it should be listed under all matching genres. *Lists of genres, such as "sci-fi/thriller", are not considered a separate genre.*

Input

The first line will contain one integer, denoting n number of movies to be listed, $5 \leq n \leq 100$, followed by a comma, followed by a string that specifies the field to group the movies on, followed by a comma, followed by the order to sort the groups, either ASC or DESC. The field will be either "Audience Rating", "Critic Rating", or "Genre(s)". Then for each movie, five lines will be shown in order: a line containing the title of the movie, followed by a line containing the audience rating a, $0 \leq a \leq 5$, followed by a line containing the critic rating c, $0 \leq c \leq 5$, followed by a line containing the genres, which can be either a single string, or a list of strings separated by commas.

Output

For the first line, print out "Grouped By: ", followed by the name of the grouping field, followed by a comma, then a single whitespace, then the sorting order of the groups, either DESC or ASC. For each value of the grouping field, print out the field value, followed by a single colon. Then for the list of movies with the matching field value, for each field, print out: a line contain exactly two whitespaces in the beginning, followed by the field name specified in the problem description, followed by a single colon, followed by a single whitespace, then the field value. For each movie, after printing out its fields, print out a single line starting with two whitespaces followed by a ˜, except the last movie in the group, which should not print this line. No Trailing whitespaces are allowed before or after each line.

Sample Input

```
6,Critic Rating,DESC
The Room
thriller,comedy
1
1
Eraserhead
thriller,horror
5
5
Casablanca
Thriller,romance
5
5
Transformers
action,Sci-Fi
4
1
Twelve Angry Men
drama
5
5
The Shining
horror
5
5
```

Sample Output

```
Grouped By: Critic Rating, DESC
5:
  Title: The Shining
  Genre(s): horror
  Audience Rating: 5
  Critic Rating: 5
  ~
  Title: Twelve Angry Men
  Genre(s): drama
  Audience Rating: 5
  Critic Rating: 5
  ~
  Title: Eraserhead
  Genre(s): thriller,horror
  Audience Rating: 5
  Critic Rating: 5
  ~
  Title: Casablanca
  Genre(s): thriller,romance
  Audience Rating: 5
  Critic Rating: 5
1:
  Title: Transformers
  Genre(s): action,sci-fi
  Audience Rating: 4
  Critic Rating: 1
  ~
  Title: The Room
  Genre(s): thriller,comedy
  Audience Rating: 1
  Critic Rating: 1
```

CSULA PROGFEST 2014
Problem 2
The Coverup

Drinking a cup of warm, fuzzy hot chocolate on a typical winter day may be one of the enjoyable things to do. However, chocolate stains are notoriously difficult to clean up. Byran's parents specifically told him not to drink hot chocolate in his room, which has white walls. Obviously, Bryan ignored their warnings, and did the exact opposite. While sipping on his hot chocolate, he sneezed, and hot chocolate was spilled all over his room. He scrubbed and cleaned as hard as he could, but several small stains still remain on the wall. Unable to clean up the stains, Bryan wants to buy some posters to cover up the stains to avoid the fallout from his parents. His local poster shop only sells square posters. What is the smallest square poster he can buy to cover it up? Assume that the stains are points on the integer lattice in the plane. The poster can be oriented in any way. Your program will find the area of the smallest square poster that will cover up all the stains.

Input

The first line of input contains a single integer T expressed in decimal with no leading zeroes, denoting the number of test cases to follow. The subsequent lines of input describe the test cases.

Each test case begins with a single line, containing a single integer n expressed in decimal with no leading zeroes, the number of points to follow; each of the following n lines contains two integers x and y, both expressed in decimal with no leading zeroes, giving the coordinates of one of your points.

You are guaranteed that $T \leq 30$ and that no data set contains more than 30 points. All points in each data set will be no more than 500 units away from (0,0).

Output

Print, on a single line with two decimal places of precision, the area of the smallest square containing all of your points.

Sample Input

```
2
4
-2 -2
2 -2
2 2
-2 2
4
10 2
10 -2
-10 2
-10 -2
```

Sample Output

```
16.00
288.00
```

CSULA PROGFEST 2014
Problem 3
Lazy Days

Eddie is a sophomore studying computer science, and is failing his discrete mathematics course. It's not that he doesn't understand the concepts: it's just that he's too lazy, or too smart(so he claims), to do homework. Staring at the chapter homework for functions and relations, he realized that he could write a program to do the homework for him. He learned the following definitions in class:

Definition 1. *A **relation** from a set A to a set B is a set of ordered pairs (a, b) where a is an element of A and b is an element of B. i.e.* $\{(-4, 1), (-2, 3), (0, -5)\}$*, set A is* $\{-4, -2, 0\}$ *and set B is* $\{1, 3, -5\}$ *The set A is the **domain** of the relation and set B is its **codomain**.*

Definition 2. *A **function** is a relation that satisfies:*

- *for each element a in the domain, there is an element b in its codomain such that (a, b) is in the relation, and*
- *if (a, b) and (a, c) are in the relation, then b = c.*

Definition 3. ***one-to-one**: A function is said to be one-to-one, if and only if each element in the codomain corresponds to no more than one element in the domain.*

Definition 4. ***onto**: A function is said to be onto, if and only if each element of the codomain of the function has a corresponding element in the domain.*

Definition 5. ***bijective**: A function is said to be bijective if and only if it is both one-to-one and onto.*

Your job is to design a program that, given a domain and codomain and a list of relations using them, determine what type of functions the relations are, if they are functions at all.

Input

The first line will contain a list of D integers separated by commas, denoting the domain for all relations to be tested, $1 \leq D \leq 100$. The second line will contain a list of C integers separated by commas, denoting the codomain for all relations to be tested, $1 \leq C \leq 100$. The third line will contain a single integer, denoting the number of relations to be tested. Each relation consists of one line, denoting the relation in set notation.

Output

For each relation, print, on a single line whether it's "not a function" or "function". If it is a function, print a single comma after "function", followed by either "one-to-one", "onto", "bijective", or nothing at all if it doesn't fall into any of the categories.

Sample Input

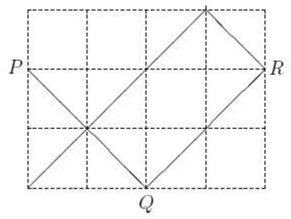
```
1,2,6,7,8,9
1,2,3,4,5
4
{(1 1),(1 2),(2 3),(6 5),(7 4)}
{(1 1),(2 2),(6 3),(7 4),(8 5)}
{(1 4),(2 4),(6 4),(7 4),(8 4)}
{(1 1),(2 2),(6 3),(7 5),(8 5),(9 4)}
```

Sample Output

```
not a function
function,bijective
function
function,onto
```

CSULA PROGFEST 2014
Problem 4
Rock-A-Doodle

During his algorithms class, Eric, bored out of his mind, started doodling on his paper. He started at a boundary grid point P; note that P is a corner of one or two grid cells. He drew a diameter of one of those cells and continued on a straight line until reaching point Q on another edge of the grid. Then he started another line from Q, perpendicular to the line PQ until hitting another edge at point R. He kept drawing lines as above, until he could not draw a new line, either because a perpendicular line would not start with a cell diameter or it would fall on an already drawn line. Then he was puzzling how he would be able to count the number of minimal rectangles he has created on the grid. At this time, the professor noticed him and asked him what he was doing. "Sorry, I was designing a problem for ProgFest," he said.



You are to write a program to, given the dimensions R and C of a grid, the coordinates x and y of a point P on one of the edges of the grid, and the direction (up, left), (down, left), (up, right), or (down, right) of the first line, help Eric to find out the number of minimal rectangles.

Input

The input file contains multiple test cases. The first line of the input, contains t, the number of test cases that follow. Each of the following t blocks, describes a test case.

The first line of each block, contains two positive integers R and C, which are the number of horizontal and vertical gridlines, respectively ($2 \leq R, C \leq 1000$). The coordinates of the starting point comes on the next line, as two positive integers y and x, ($1 \leq y \leq R$, $1 \leq x \leq C$); for the upper-left point of the grid, we have x = y = 1. The third line of each test case, contains a two-character code which is one of the following:

- "DR", the first move is in the down-right direction,
- "DL", the first move is in the down-left direction,
- "UL", the first move is in the up-left direction,
- "UR", the first move is in the up-right direction.

Output

For each test case, output one line containing the number of non-overlapping rectangles formed.

Sample Input

```
2
6 5
1 1
DR
9 10
4 1
UR
```

Sample Output

```
6
3
```

CSULA PROGFEST 2014
Problem 5
The Message

Note passing in class is a frowned upon activity, and an ancient one due to the popularization of cell phones. Nevertheless, the adrenaline rush of successfully passing a note without being noticed by the instructor still draws students to this forbidden activity. However, big rewards come with big risks. When a instructor catches a student passing a note, he or she often reads the note out loud. It could be very embarrassing for the parties involved, depending on the content of the note. Andrew, wanting to give a note to a girl he has a crush on, thought of a way to encode his message before passing it on. The encoding scheme goes as follows:

- for each word(could be one or more characters, each word is separated by a white space), reverse its order, including punctuation.
- for each reversed word, shift each character one forward in the alphabet, ignoring punctuation.

The decoding scheme simply runs the encoding scheme in reverse. Your job is to write a program for Andrew and his first love to encode and decode messages using his scheme. Remember, Andrew's first love depends on you.

Input

Words are character sequences(mixed uppercase/lowercase) that include punctuation, separated by a single white space. The first line contains a single integer, denoting the number of messages to encode/decode. Messages to be decoded will start with the character '$',and messages to be decoded will start with the character '@'. Each message will contain a line of words. You may assume the message only contains the upper case/lower case alphabet excluding uppercase and lowercase z, and the following characters: the comma, the period, the question mark, the exclamation mark, the minus sign, and single quotes.

Output

For each input, print out either the decoded/encoded messages, excluding the '$' and '@' characters, keeping the original character cases with a single whitespace between the words like the input.

Sample Input

```
3
$J FWPM !VPZ xfseoB--
@I've liked you for a long time, too! Go easy on the caps though. --Michelle
$fw'J efljm vpz spg b hopm ,fnju !ppu pH ztbf op fiu tqbd .ihvpiu fmmfidjN--
```

Sample Output

```
I LOVE YOU! --Andrew
fw'J efljm vpz spg b hopm ,fnju !ppu pH ztbf op fiu tqbd .ihvpiu fmmfidjN--
I've liked you for a long time, too! Go easy on the caps though. --Michelle
```

CSULA PROGFEST 2014
Problem 6
The Perks of Being A Programmer

Michael, a retired COBOL programmer, opened up a bakery in Los Angeles, focusing on making breads named with programming puns. His best selling item is, of course, the rubber duck, a duck shaped banana bread designed to help programmers debug their code and refuel their energy. As a treat to his fellow programmers, he decided to give them a special discount. The discount scheme is the following:

- A non-programmer gets no discount and pays full price.
- Each programmer will tell Michael his or her preferred language. If it is COBOL, he or she gets no discount at all and pays full price.
- Each non-COBOL programmer is offered an additional 10 percent discount on their total purchases, after all discounts applied.
- If a non-COBOL programmer buys two or more items, he or she gets a 15 percent discount.
- If a non-COBOL programmer buys 5 or more items, he or she gets an additional 10 percent discount after applying the two item discount.

  Your job is to write a program, not in COBOL, that will read in a list of orders and produce a total and a final total after applying the discount scheme.

Input

The first line will contain a single integer, denoting the number of purchase orders to be processed. Each purchase order will be printed as a single line in the following format: whether the customer is a programmer(either 'yes' or 'no'), followed by a single comma, followed by their preferred programming language(nothing if the customer is not a programmer), followed by a single comma, followed by list of the price of the items purchased, each price is separated by an white space.

Output

For each purchase order, in a single line, print out the total price before applying discounts, followed by a single comma, followed by the total price after applying discounts. Both prices should be rounded to the nearest hundredth. For example, 15.545 will become 15.55 and 15.544 will become 15.54.

Sample Input

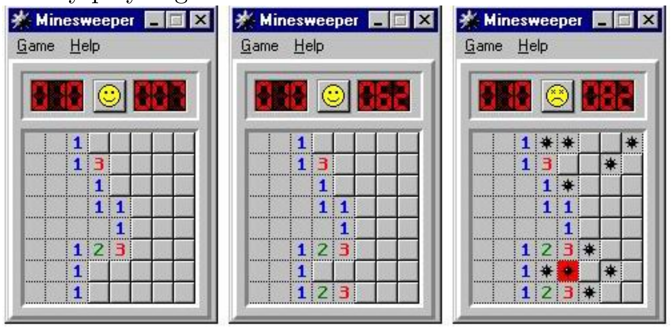
```
4
no,,10.25 2.25 2.25
yes,COBOL,2.25 2.25 2.25
yes,Python,20.20 21.45 42.08 8.62 10.02
yes,Java,10.25 10.25
```

Sample Output

```
14.75,14.75
6.75,6.75
102.37,70.48
20.50,15.68
```

CSULA PROGFEST 2014
Problem 7 Minesweeper

The game Minesweeper is played on an n by n grid. In this grid are hidden m mines, each at a distinct grid location. The player repeatedly touches grid positions. If a position with a mine is touched, the mine explodes and the player loses. If a positon not containing a mine is touched, an integer between 0 and 8 appears denoting the number of adjacent or diagonally adjacent grid positions that contain a mine. A sequence of moves in a partially played game is illustrated below.



Here, n is 8, m is 10, blank squares represent the integer 0, raised squares represent unplayed positions, and the figures resembling asterisks represent mines. The leftmost image represents the partially played game. From the first image to the second, the player has played two moves, each time choosing a safe grid position. From the second image to the third, the player is not so lucky; he chooses a position with a mine and therefore loses. The player wins if he continues to make safe moves until only m unplayed positions remain; these must necessarily contain the mines.

Your job is to read the information for a partially played game and to print the corresponding board.

Input

The first line of input contains a single postitive integer $n \leq 10$. The next n lines represent the positions of the mines. Each line represents the contents of a row using n characters: a period indicates an unmined positon while an asterisk indicates a mined position. The next n lines are each n characters long: touched positions are denoted by an x, and untouched positions by a period. The sample input corresponds to the middle figure above.

Output

Your output should represent the board, with each position filled in appropriately. Positions that have been touched and do not contain a mine should contain an integer between 0 and 8. If a mine has been touched, all positions with a mine should contain an asterisk. All other positions should contain a period.

Sample Input

```
8
...**..*
......*.
....*...
........
........
.....*..
...**.*.
.....*..
xxx.....
xxxx....
xxxx....
xxxxx...
xxxxx...
xxxxx...
xxx.....
xxxxx...
```

Sample Output

```
001.....
0013....
0001....
00011...
00001...
00123...
001.....
00123...
```

CSULA PROGFEST 2014
Problem 8
Text Formalization

One duty Jimmy has at the ACM is to formalize the language and grammar used in texts. Part of this job is expanding contractions and certain acronyms. A contraction in English is a word or phrase formed by omitting or combining some of the sounds of a longer phrase. For example, ”don’t” is a contraction for ”do not” and ”o’clock” comes from ”of the clock.”

An acronym is a series of letters (or word) formed from the initial letters of a name or from combining parts of a series of words. For example, ”ACM” for ”Association for Computing Machinery” or ”radar” for ”radio detecting and ranging.”

Your job is to take a list of contractions and acronyms, and expand all contractions and some acronyms in a text.

Input

Input begins with two numbers, $C < 50$ and $A < 50$, indicating respectively the number of contractions and acronyms Jimmy must expand. The next C lines list a contraction and its formal expansion. Following will be a list of A acronyms and their expansions, each on individual lines. Both contractions and acronyms will be presented in the following format: ”contraction or acronym” -> ”expansion”

Output

Output each text exactly as input, except for necessary expansions. All contractions must be fully expanded. Each contraction may appear as listed, entirely uppercase, or capitalized (first letter uppercase, remaining letters as listed). The expansion should follow the same rule; if a contraction is uppercased, the expansion should be uppercased as well. If more than one case applies, choose the earliest matching case in the list: ”as listed,” ”uppercased,” and ”capitalized.”

Since acronyms are useful for understanding and identifying names, only modify the first instance of an acronym in each text. An instance of an acronym must match the case exactly (”acm” is not an instance of ”ACM”). The modification consists of replacing the acronym with the expansion, followed by a space, followed by the acronym in brackets. This allows the reader to connect the acronym with the fully expanded term. Ignore recursive acronyms that are created after expanding some previous text.

The terminating line of ’#’ should be printed after each text. If more than one expansion or acronym match can be valid, use the one which starts earlier in the text. If several begin at the same letter, use the one appearing earliest in the input lists. Use the sample below to illustrate the process.

Sample Input

```
3 4
"doesn’t" -> "does not"
"isn’t" -> "is not"
"can’t" -> "cannot"
"ACM" -> "Association for Computing Machinery"
"CSULA" -> "California State University, Los Angeles"
"IO" -> "input output"
"ASAP" -> "as soon as possible"
Creating problems for the CSULA Progfest programming contest
isn’t an easy feat.
We have to solve all the problems before hand and we can’t just use the sample IO files.
If you find any errors, please submit them on the website and we will correct them ASAP.
Signed,

ACM CSULA Chapter
#
The ACM CSULA Chapter,
welcomes students of all majors to join.
But let’s be serious, non-programmers can’t understand programming puns.
#
```

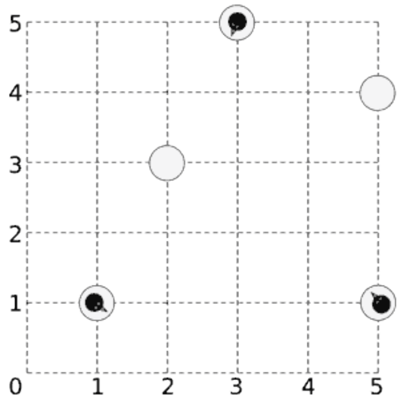
Sample Output

```
Creating problems for the California State University, Los Angeles (CSULA) Progfest programming contest
is not an easy feat.
We have to solve all the problems before hand and we cannot just use the sample input output (IO) files.
If you find any errors, please submit them on the website and we will correct them as soon as possible (ASAP).
Signed,

Association for Computing Machinery (ACM) CSULA Chapter
#
The Association for Computing Machinery (ACM) California State University, Los Angeles (CSULA) Chapter,
welcomes students of all majors to join.
But let’s be serious, non-programmers cannot understand programming puns.
```

CSULA PROGFEST 2014
Problem 9
Ice Hopping

Somewhere in Canada, for some strange reason, Richard and his family are standing on a number of ice floes. As seen on the diagram below(which corresponds to the first test case), The family would like to meet on the same floe. They don't want to fall the water, as it would be life threatening, so they have use their limited jumping distance to get together by jumping from piece to piece. However, due to global warming, the floes are showing cracks, and they get damaged further by the force needed to jump to another floe. Richard calculated how many times a person can jump off each floe before it breaks apart and disappears. Landing on an ice floe does not damage it. You have to help the family find all floes where they can meet.



Input

On the first line one positive number: the number of test cases, at most 100. After that per test case:

- One line with the integer N ($1 \leq N \leq 100$) and a floating-point number D ($0 \leq D \leq 100000$), denoting the number of ice pieces and the maximum distance a person can jump.
- N lines, each line containing x_i, y_i, n_i and m_i, denoting for each ice piece its X and Y coordinate, the number of people on it and the maximum number of times people can jump off this piece before it disappears ($-10000 \leq x_i, y_i \leq 10000$, $0 \leq n_i \leq 10$, $1 \leq m_i \leq 200$).

Output

Per test case:

- One line containing a space-separated list of 0-based indices of the pieces on which all people can meet. If no such piece exists, output a line with the single number -1.

Sample Input

```
2
5 3.5
1 1 1 1
2 3 0 1
3 5 1 1
5 1 1 1
5 4 0 1
3 2
-1 0 5 10
0 0 4 1
2 0 2 1
```

Sample Output

```
1 2 4
-1
```