Problem A - Skill Testing Question

Why work for your money when you can win it instead? You've decided that from now on you're going to enter every contest, draw and sweepstakes you come across in an effort to earn some ridiculous loads of cash. There's only one thing in your way: those pesky skill testing questions.

Luckily, you know that all skill testing questions are of the form (A + B) * (C – D). Given the amount of entries you're bound to write, it looks like writing a program to answer these questions for you is going to save a lot of time. Time you can spend earning more money!

Input

The first line of input contains an integer **T** (1 ≤ **T** ≤ 100), the number of test cases. Each test case consists of four integers, **A**, **B**, **C**, and **D** (-100 ≤ **A**, **B**, **C**, **D** ≤ 100).

Output

For each test case, output the answer to the skill testing question (A + B) * (C – D).

Sample Input

```
3
1 2 3 4
4 3 2 1
0 0 0 0
```

Sample Output

```
-3
7
0
```

Problem B - Starcraft II BM

I always hate it when n00bs beat me at Starcraft II. It's just intolerable! The best way to get back at them, of course, is to tell them exactly how bad they are. The process of telling people how bad they are is sometimes also called "BM".

Starcraft II has 4 possible race selections: Zerg, Terran, Protoss, and Random. It's generally a good idea to have an insult ready for each race, in addition to some general comments that you can make to anyone (regardless of race). For example, to a Zerg opponent that kills me with a Roach/Ling all in, I might tell him "Lucky Zerg, typical all-in trash". It's fairly apparent that I couldn't use this statement against a Protoss player, because it contains the word "Zerg" in it. An example of a general BM statement might be "Stay eZ, bronze n00b". Race specific BM contains the race as a substring (the race name is not case sensitive); general BM does not contain any race names in it. Race specific BM never contains more than one race name.

The input consists of several BM strings, followed by a series of races of your opponents. For each race, you should output all the BM that you can give them! Specifically, you should output all the race-specific BM that you know that applies to that race, as well as all the general BM.

Input

The input starts with an integer **T** (1 ≤ **T** ≤ 10). **T** test cases follow. Each test case begins with an integer **S** (1 ≤ **S** ≤ 10), the number of BM strings that you know. Following this are **S** lines of BM. Each line contains no more than 300 characters. After this is an integer **G** (1 ≤ **G** ≤ 10), the number of games that you'll play. Following this are **G** lines of race names.

Output

For each race name, output all the BM that applies to that race (both race-specific BM and general BM), in the order that they appear in the input. Do not print any blank lines between test cases.

Sample Input

```
1
6
ur eZ
Zerg all in. Typical.
lolololol Terran 1-1-1 lrn2play
Protoss make colossus. I am not surprised.
Random player 6 pool. Can't play past the 5 minute mark.
stay bronze n00b
2
random
PrOToSs
```

Sample Output

```
ur eZ
Random player 6 pool. Can’t play past the 5 minute mark.
stay bronze n00b
ur eZ
Protoss make colossus. I am not surprised.
stay bronze n00b
```

Problem C - Ant Squashing

My friend John hates ants. Actually I lied; I don't have any friends. But let's suppose that John hates ants regardless – and being a sensible person, John plans to squish the ants! Sadly for John, it's illegal to squish more than 2 ants. What a silly story this is becoming!

Ants are modelled by a series of dots on the x-axis. It turns out that ants tend to stand very still, which makes it easier for John to squish them. Furthermore, ants like to sit in large groups with regular spacing between ants in the same group, and all ants sit on lattice points. As an example, one group of ants might sit on the points {3, 5, 7, 9, 11}. For convenience, we will denote this group by the triple *3 11 2*, meaning that the group of ants starts at 3, ends at 11, and has regular spacing of 2. There will be several groups of ants in a problem, but the intervals never overlap (the interval [3,11] is completely disjoint from every other interval of ants). An example of two overlapping triples might be *3 11 2* and *1 19 3*. This is invalid input and will not appear.

For each test case, there will be several groups of ants (no more than 50,000 groups). Your job is to find the distance between the closest two ants, to minimize John's effort.

Input

There will be several test cases. Each test case begins with an integer **N** (1 ≤ **N** ≤ 50,000) on its own line. On the next line are 3***N** integers, denoting the **N** groups of ants as triples **A B C** as above, such that **A** < **B**, **C** > 0, **C** divides (**B**-**A**). No group has a member with a location greater than 10^9 in absolute value.

Input ends on **N**=0. Do not process this case.

Output

Output a single number for each test case – the minimum distance between 2 ants.

Sample Input

```
3
1 4 3 5 10 5 14 21 7
2
1 10 9 16 31 5
0
```

Sample Output

```
1
5
```

Problem D - Throw-Down Sticks

After repeated failed attempts to get his hyperactive nephew to play the cerebral game of Pick-Up Sticks, Bob has decided to make a more appropriate game: Throw-Down Sticks!

The game is simple. First, draw a circle on the ground. Then, take a bunch of sticks, and throw them on the ground. For every pair of sticks that overlaps, you get a point if they overlap on or inside the circle.

Well, now the sticks are down, and it's time to determine the score. If we treat the sticks as lines in a plane, and if we say that the circle is centered at (0, 0) with radius 10, how many points have you scored?

Input

The first line of input contains an integer **T** ($1 \le$ **T** ≤ 100), the number of test cases. Each test case begins with an integer **N** ($1 \le$ **N** ≤ 100), the number of sticks. Then follow **N** lines, each with two real numbers **M** and **B** ($-20 \le$ **M**, **B** ≤ 20). We'll treat each stick as the line y = **M**x + **B** in the plane. No two lines will have the same **M** and **B**.

Output

For each test case, output your score: the number of pairs of sticks that intersect on or inside the circle.

Sample Input

```
3
3
1 0
1 1
0 5
4
5 20
-5 20
6 -8
-6 -8
5
0 0
1 0
-1.5 0
2 0
-2.2 0
```

Sample Output

```
2
3
10
```

Problem E - Don’t Kill Me, I’m Running

Epic Mafia is an online version of the classic Mafia card game. The details of the game itself are not too important. What’s important is winning!

The goal of an Epic Mafia competition is to get the most points. For simplicity, we’ll say that every game you win is worth 60 points. We’re near the end of the round, and I’ve played all of my games. So far I’m in 1st, but we’ve got some hotshots who think they can beat me. I want to know the chance that I’m going to win the current competition. I mean a *real* win. Not some kind of wimpy tie for 1st place.

Input

The first line of input contains an integer **T** (1 ≤ **T** ≤ 1,000), the number of test cases. Each test case begins with an integer **N** (1 ≤ **N** ≤ 100), the number of competitors. Then follow **N** lines that contain, in order: the name of a competitor (a string of at most 60 alphanumeric characters), the number of games that person has left to play (an integer 0 ≤ **G** ≤ 10), the number of points so far (an integer 0 ≤ **P** ≤ 2500), and the win rate of that player so far (a real number 0 ≤ **W** ≤ 1).

The competitors will be listed in non-ascending order of points. Because I’m winning, I’ll always be the first person, and I will always have 0 games left to play.

Output

Output a single real number, the chance that I win the competition, rounded to three decimal places. Assume that each player’s observed win rate is actually the chance that they’ll win each of their future games, and that none of the competitors play against each other, so all wins/losses are independent.

Sample Input

```
2
2
wjomlex 0 1337 1.000
lordklotski 1 1336 0.337
3
vulpesx 0 1234 0.567
sourspinach 2 1159 0.765
renard 10 1001 0.500
```

Sample Output

```
0.663
0.071
```

Problem F - Robo Rally

In the game of Robo Rally, the object is to collect a series of flags before your opponents. Each turn, you get some cards and you must use them to program your robot. In this (mildly) simplified version, there are no opponents, only one flag, and you will only consider a single turn.

Each card has one of 7 instructions on it: Move 1, Move 2, Move 3, Rotate Right, Rotate Left, Back Up, U-Turn. These cards will be signified in the input by the characters **1 2 3 R L B U** respectively. A Move X card moves your robot X spaces in the direction it is facing. Rotate Right or Left rotates your robot in-place by 90 degrees clockwise or counter-clockwise respectively. A Back Up card moves your robot 1 space in the direction opposite to the direction your robot is facing. A U-Turn rotates your robot in-place 180 degrees.

For each map in the input, you will be given 9 cards, and must choose the best 5 to execute. "Best" means that your sequence should get your robot to the flag, if possible. If it is impossible to get the flag, then your sequence must get your robot to end up as close to the flag as possible. Distance is measured as the number of orthogonal movements necessary to reach the flag, ignoring obstacles (that's the L1 or Manhattan norm for you math people). However, the land of Robo Rally is full of treachery. Here are the different map tiles you will encounter:

S – Robot starting location

F – Flag

. – Empty space

– Wall

L, **R** – Left and Right gears (respectively)

^, **v**, **<**, **>** – Up, Down, Left, and Right conveyors (respectively)

X – Bottomless pit

There will be exactly one starting location, and exactly one flag. Your robot always begins facing upwards. A turn proceeds in 5 phases, as follows: On the *i*th phase, your robot executes its *i*th card. Then, if it is on a conveyor, it is moved one space in the direction of the conveyor. Then, if it is on a gear, it is rotated 90 degrees in the direction of the gear. To illustrate:

```
L<
.<
.S
```

In this situation, if the robot executes Move 2 facing up (North), then it will land on a left conveyor. It will be pushed one space left, onto the left gear. Then, it will rotate counter-clockwise and end up facing left (West).

If a robot would move into a wall, whether by its own volition, or because of a conveyor, it moves as far as possible without going into the wall's space. So a robot facing a wall with one empty space in between moves only one space forward when executing a Move 3 (see sample input). If a robot would move off of the map, or over a space containing a pit, it falls to its doom. To collect the flag, you must be in the flag's space at the end of a phase. Simply moving over the flag's space is not sufficient. It is acceptable for the robot to fall after collecting the flag, but if the robot falls before collecting the flag, the sequence is not a best sequence, even if the robot falls very close to flag.

Input Format

The first line of input is an integer **T** (1 ≤ **T** ≤ 100). **T** test cases follow. Each test case consists of 6 lines with 6 characters in each, describing the map, then a line with 9 characters describing the cards you have to work with. Valid characters for each part are as described above.

Output Format

For each case, print a single line with 5 characters, the lexicographically earliest of all possible "best" sequences as described above. If all possible sequences end with the robot falling to its untimely demise, then print "gg robot :(" instead (without quotes).

Sample Input

```
3
F<#...
..<#..
...<#.
....<#
.....<
.....S
L123R1L31
......
..X...
.XSX..
..X...
....F.
......
123123123
#S#...
.>RR..
......
......
...v..
...L.F
B1U3ULR1R
```

Sample Output

```
11231
gg robot :(
B131L
```