A High Scores

With new game development being expensive your company has decided to re-release a collection of small games as a larger title. To the dismay of your play-testers, this means that the individual scoreboards can not provide an overall ranking! Your task is to combine the scores from the separate games into one scoreboard, so the game can ship on time.

Because some of the separate games generate points in different scales, each score will be converted to a percentage of maximum possible score, as integer between 0 and 100 (inclusive, rounded down). Unfortunately, you were never told how many points each game can award.

After trying to get an answer from your boss for a week (currently on vacation in Prague), you decided that the high scorer in any game with a non-zero score will simply be granted 100 points. All of the other scores for that game will be scaled relative to the leader. For example, if the leader scored 1023511 points in Atomic Crush Man, a score of 53432 would be worth 5 points on the overall scoreboard. If no one every played the game, or the leader scored zero points, zero overall points will be awarded for that game. Obviously a player who has never played a game will get zero overall points for it.

The names in the overall scoreboard will be sorted based on the sums of their scaled points. If two players have the same scaled points, they will be sorted by name. No two players will use the same name, and only the highest score of each player will appear on a scoreboard.

Input

The input will consist of a number of scoreboards, one for each of the individual games. Each scoreboard will start with a number that specifies the number of lines for that board. A zero will be used to indicate the end of input. No more than 200 scoreboards will be present in an input.

Each scoreboard line contains a player name followed by a space and the player's score for that game. The player name will consist of between one and ten (inclusive) upper-case letters (A through Z). The score will be a non-negative integer no greater than 10000000. Any player that does not appear on the scoreboard has not yet played that game. The lines for each scoreboard are sorted by points and name as required for the output.

Output

The output will contain the combined scoreboard, in the same format as the game scoreboards, sorted by points and name as specified above. Even players with zero points should be displayed on this scoreboard. The scoreboard will never be more than 500 lines long.

Sample Input

```
4
WINNAR 100
TWOLEET 15
ONELEET 5
EYELEET 4
4
TWOLEET 5000
WINNAR 500
ONELEET 50
EYELEET 0
0
```

Sample Output

```
TWOLEET 115
WINNAR 110
ONELEET 6
EYELEET 4
```

B Play By Mail

Once upon a time, before the Internet, before even BBSes, games were played long distance by a technique known as Play by Mail. Each player would set up an identical copy of the game board. Then the first player would make a move and mail it to the second player. The second player would make a move and mail both to the third. This continued until the last player made a move and mailed all of the moves back to the first player.

This system obviously requires a high level of trust between players. If a mistake is made in relaying the moves, either deliberately or accidentally, the entire game would be worthless. Your task is to monitor the mailings sent between players during a game to determine who, if anyone, is cheating.

The game board format consists of 5 rows of 5 characters. Each character will be a digit, representing the player that owns the piece, or a period ('.') to represent an empty space. No space will be placed between columns or rows.

During a move a player must move a single piece that player owns to an adjacent space on the board along N, S, E or W. If there was previously a piece in the target space, it is captured and removed from play (even if it is also owned by the player).

Input

The first line of input will contain only a number, specifying the number of players ($2 <= N < 10$).

You will then be given a number of mailings, each consisting of a valid game board for N players, and separated by a line containing only "===". The input will be terminated by a line containing only "======". There will always be at least one, and no more than 1000 mailings.

The first mailing, which will always be present specifies the initial board position. The second mailing contains the board after the first move of player 1, the third after the first move of player 2, and etc.

Output

If one or more players are cheating, print a line containing "The cheating player(s) is/are: " followed by a comma-separated list of the cheating players and a period, as shown in the sample output. If no players are cheating, print a line containing only "Nobody cheated!" (without the quotation marks).

Sample Input 1

```
2
1111.
111.2
11.22
1.222
.2222
===
111.1
111.2
11.22
1.222
.2222
===
111.2
111..
11.22
1.222
.2222
======
```

Sample Output 1

```
Nobody cheated!
```

Sample Input 2

```
3
1111.
111.2
11.22
1.222
32222
===
11111
111.2
11.22
1.222
32222
===
11112
111..
11.22
1.222
32222
===
11112
111..
11.22
3.222
32222
======
```

Sample Output 2

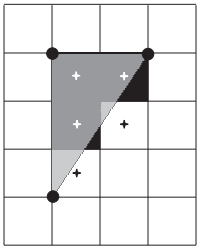
```
The cheating player(s) is/are: 1, 3.
```

C Triangle Rasterizer

You are developing the next super Frogger game, but your boss wants to add some extra spice. He decides that since all the new games are going 3D Frogger should as well. The only problem is your target platform has no 3D acceleration. Solution, draw the triangles using software!

In this problem, you will be given the 3 vertices of the triangle to draw. The vertices will be floating point values, and you must draw the filled triangle. The filled triangle will be drawn on a grid of pixels (or ASCII characters in our case). Pixels that are defined to be within the triangle are drawn with a '*', while pixels outside the triangle will be drawn with a '.' (single quotes for clarity only).

A pixel is defined to be within a triangle if the center of the pixel is within the triangle. All pixels are one wide, by one high. Don't worry about numerical accuracy (for example, if the triangle edge passes right through the center of a pixel, you may either draw that pixel, or not). For example, the triangle (1.0, 1.0), (4.0, 1.0), (1.0, 3.0) is drawn as follows:



Input

One line, containing T, the number of test cases for you to draw. Then there will be T sections, each containing the following information.

One line, containing N, the number of triangles for you to draw. N will never be larger than 50.

Then come N lines, each containing 6 floating point values. The first two represent the (x,y) co-ordinates of the first vertex, the next two the (x,y) of the second vertex and the last two the (x,y) co-ordinate of the last vertex. $0 <= (x,y) < 30$ for all triangles.

The floating point values will always have 2 digits of precision.

Each triangle should be drawn into the same buffer; see the sample output for an example.

Output

T sections, with each section containing the following,

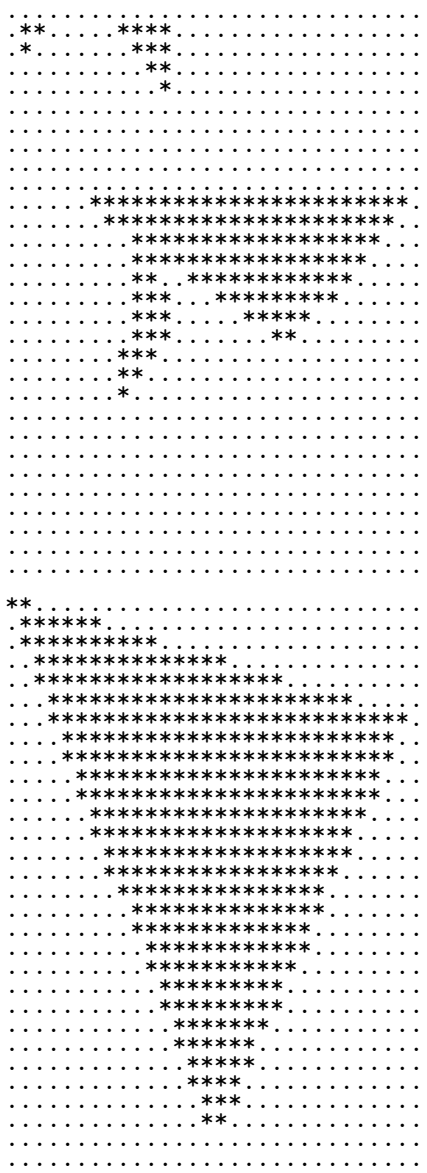
30 lines, each with 30 characters per line. The first line represents row 0 of the screen, the second row 1, etc. The first character in each line represents column 0 of the screen, the second character, column 1, etc. Print a blank line after each section.

Sample Input

```
2
4
1.00 1.00 4.00 1.00 1.00 3.00
4.96 10.53 29.96 9.53 20.53 18.34
8.13 20.87 10.00 8.00 12.15 17.43
8.00 1.00 12.00 1.00 12.00 5.00
1
0.00 0.00 29.00 6.5 15.0 29.0
```

Sample Output

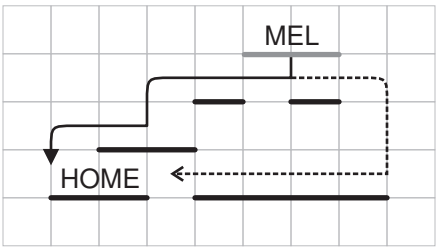
This output represents only one possible result, yours does not need to match it exactly to be judged correct.



D Pathing

The Amazingly Complicated Mechanisms game allows the player to place a variety of objects on a 2D board. One of these objects, Mel, has a mind of his own. All Mel wants to do is return to his house by as short a path as possible, to avoid being eaten by alligators. Your task is to determine the shortest distance path (including falling) from Mel's initial position to his house, if there is one.

Each object, including Mel and his house, will be specified as a horizontal platform. Whenever part of Mel overlaps another object for at least one unit of width, he can move either East or West until he has no more support. If no part of his base is overlapping, he will fall directly downward until he hits another object. The instant he overlaps his house for at least one unit of width, he will go inside and be safe. Falling off the bottom of the board will leave Mel in the alligator pit, which is bad.



This example is shown below in the Sample Input and Output. The correct path is shown by the solid line, while a longer (and therefore incorrect) path is shown as a dashed line. Note that Mel is exactly one unit wider than the gap between the two top platforms, so can not fall between them and he does not make a decision as he passes the gap.

Input

The input will start with a single number on a line indicating the number of boards to follow (1 <= N <= 20).

Each board consists of a list of objects, one per line. Each object is provided as three non-negative integers specifying left (minimum X position), right (maximum X position) and top (Y position). No object will ever have the same minimum and maximum X coordinate. All values will be between 0 and 1000000, inclusive. The first object specified is Mel, the second object is his house. There will always be at least two objects (Mel and his house) and no more than 1000 objects. No two objects specified in the input will overlap, though their ends may touch.

The end of a board is indicated by an object with left, right and top all set to -1. Note that gravity pulls toward the negative y direction.

Output

For each board, print a line containing the directions that Mel must move to reach his home in the shortest distance. There must be one (and only one) direction specified for every time Mel falls onto a non-home platform. If Mel falls onto his home immediately, simply print a blank line.

Each direction will be either "W" for West (toward negative x) or "E" for East (toward positive x). Do not place spaces between the letters. If there is no way for Mel to get home print a line containing "GATOR BAIT" (with no quotation marks). If there is more than one shortest route, generate the first route that would appear in an alphabetical ordering of all shortest routes.

Sample Input

```
2
4 6 3
0 2 0
3 7 0
1 3 1
3 4 2
5 6 2
-1 -1 -1
4 6 3
0 2 0
1 3 1
3 4 2
-1 -1 -1
```

Sample Output

```
WW
GATOR BAIT
```

E Frogger

A little frog named Frogger has decided that he wants to go from his current pond to the next pond over. This isn't as easy as it should be because there are quite a few obstacles in his path. The obstacles take the form of 1 dimensional creatures that are always perpendicular to Frogger. These obstacles always move at a constant speed of one step for every step Frogger takes, and always move before Frogger does. The obstacles always start on the far left of their path, and upon reaching the right hand side of their path, will start moving in the other direction, then upon reaching the far left, will reverse directions again, etc.

Your goal is to get Frogger to the pond in the minimum amount of time possible. Frogger can not be in the same square as an obstacle, or move into a square that contains an obstacle, otherwise something very bad will happen (did I hear a squish somewhere?). Frogger may only move in the four cardinal directions, north, south, east and west. Frogger may also not move outside his grid (otherwise he will become badly lost).

It takes Frogger one unit of time to move one step, and Frogger must make a move. There will always be a way for Frogger to reach the pond.

The exact sequence that a move proceeds in is as follows. 1. Check if old Frogger is colliding with any old obstacles. 2. Move obstacles, check if new obstacles collide with old Frogger. 3. Move Frogger, check if new Frogger collides with new obstacles.

See the Sample Input for a board description which will clarify matters.

Input

An integer, n, denoting the number of test cases to follow. Then for every test case, there will be the following input.

Two integers, w and h, denoting the width and height of the grid that you need to take Frogger over. Width and height will be at most 50.

Then there will be h lines, each with w characters on it. The valid characters are: 'F', denotes Frogger's starting position, '.', denotes an empty position, '-', denotes a path for an obstacle, 'P', Frogger's destination, the pond.

There will only ever be one obstacle per row. Frogger's starting position and pond may appear anywhere on the map. The pond will always be a connected group of 'P's, and there will only be one pond.

Output

One line per input. An integer denoting the minimum amount of time it takes Frogger to cross from his start position to the end position.

Sample Input

```
2
5 5
.PPP.
---..
..---
.--..
.F.-.
6 4
------
F----P
-----P
------
```

In this sample input, there are 3 obstacles, here are how the positions change over the first 4 time units. The obstacle positions are the '*'s.

```
T0     T1     T2     T3     etc..
.PPP.  .PPP.  .PPP.  .PPP.
*--..  -*-..  --*..  -*-..
..*--  ..-*-  ..--*  ..-*-
.*-..  .-*..  .*-..  .-*..
.F.*.  .F.*.  .F.*.  .F.*.
```

Sample Output

```
8
7
```