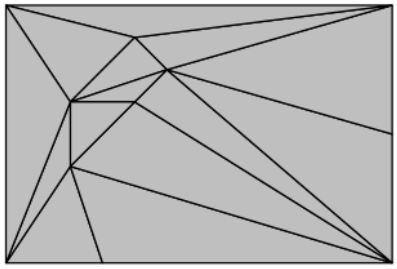
Superheroes and Shattered Glass

This problem was written by Sonny Chan (sonny.chan@ucalgary.ca) for the ACM Programming Contest back in 2011. Sonny was a CS Ph.D. student here at until a few years ago.

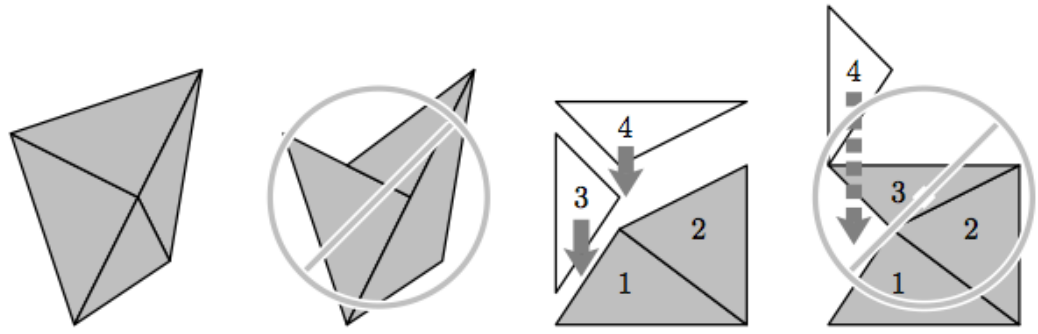
Shattered Glass

Remember the big fight where the Hulk and the Abomination threw each other through Manhattan buildings? Or the time when the Green Goblin smashed poor Spider Man through a good half a dozen brick walls? Wow, they must have shattered those walls into a million pieces!!!

It's great that we have superheroes to bring the villains to justice, but have you ever wondered who gets to repair all the collateral damage when they're done? Well actually, your job is to do exactly that! After a big fight, you must take all the broken pieces of the walls and put them back together just as they were before the fierce battle began.



A wall is a perfectly rectangular region that shatters into perfectly triangular pieces when a villain is sent through it. Through sophisticated visual analysis, you have ascertained where in the original structure every little piece came from. In essence, you have a blueprint that looks a lot like the picture above. Furthermore, you observe that wherever two broken pieces meet, they meet along the full length of the break that separates them.



(a) Breaks that can occur (b) Breaks that never occur (c) Ordering that works (d) Last piece is blocked

You have an assembly robot that can help you reconstruct a wall in place. However, the robot can only lower each piece, one at a time, straight down from the top. The robot cannot move a piece from side to side or rotate it in any way to get it where it needs to go. Thus, you must be careful about the order in which you tell the robot to reassemble the

broken pieces, lest you inadvertently block a piece from being lowered into its proper place. Can you determine an ordering of the pieces for each wall that will allow you to fully reassemble it?

Input

Data for this problem must be read in from an input file named **`shattered-glass.in`**. An integer on the first line of the input file indicates the number of walls you must reassemble. The first line for each wall has an integer, n, indicating the number of triangular pieces the wall was broken into ($2 \le n \le 1{,}000{,}000$). Then, n lines of input follow, each describing a piece with six integers, $x_1\ y_1\ x_2\ y_2\ x_3\ y_3$, that correspond to the Cartesian (x,y) coordinates of the three corners of a triangle on the original wall from which the piece came. The first line describes piece 1, the next line piece 2, and so forth. The three points will always be given in a counterclockwise winding order and form a triangle of non-zero area. All coordinates lie between 0 and 109 inclusive, with the positive y direction being the up direction. The n pieces given will cover a rectangular region exactly, with no gaps or overlaps.

Output

For each wall, print a single line with the numbers of the pieces, separated by single spaces, in an order that allows your robot to reassemble the wall. If more than one correct solution exists, any single one of them is acceptable. Publish your output both to standard out—that is, **`cout`**—and to a file named **`shattered-glass-<your-SUNet-ID>.out`**

Sample Input:

```
2
4
3 4 7 1 7 6
7 6 1 6 3 4
1 6 1 1 3 4
7 1 3 4 1 1
14
0 0 3 0 2 3
2 3 3 0 12 0
2 3 12 0 4 5
4 5 12 0 5 6
5 6 12 0 12 4
5 6 12 4 12 8
5 6 12 8 4 7
4 7 12 8 0 8
4 7 0 8 2 5
4 7 2 5 5 6
5 6 2 5 4 5
4 5 2 5 2 3
2 3 2 5 0 0
0 0 2 5 0 8
```

Sample Output:

```
4 1 3 2
1 2 13 14 3 4 12 11 5 6 10 9 7 8
```