

1. An  $m$ -bit floating-point number is a value

$$\pm 0.d_1 \cdots d_m \times 2^e,$$

where  $e$  is an integer (i.e., positive or negative), and each  $d_i$  is either 0 or 1. We call the fraction  $0.d_1 \cdots d_m$  the *significand* and  $e$  the *exponent*. If the fraction meets the condition that either  $d_1 = 1$  or all the  $d_i = 0$ , we say that the significand is *normalized*.

The  $m$ -bit *floating-point sum* of two such values is the  $m$ -bit floating-point value that is closest to the mathematical sum. In case the mathematical sum is equally close to two  $m$ -bit floating-point values, we choose the one having  $d_m = 0$ . So, in a 2-bit number,  $0.xy0\dots$  rounds to  $0.xy$ ,  $0.xy1\dots$  rounds up to  $0.xy + 0.01$  if any of the bits after the '1' are also 1,  $0.x01$  rounds to whichever  $0.x0$  and  $0.x11$  rounds to  $0.x1 + 0.01$ .

Write a program to repeatedly read in an integer,  $m$  and two normalized  $m$ -bit floating-point values, and print out the normalized  $m$ -bit sum. The format of the input will be as in the following example (for  $m = 6$ ); it is free-form, as usual.

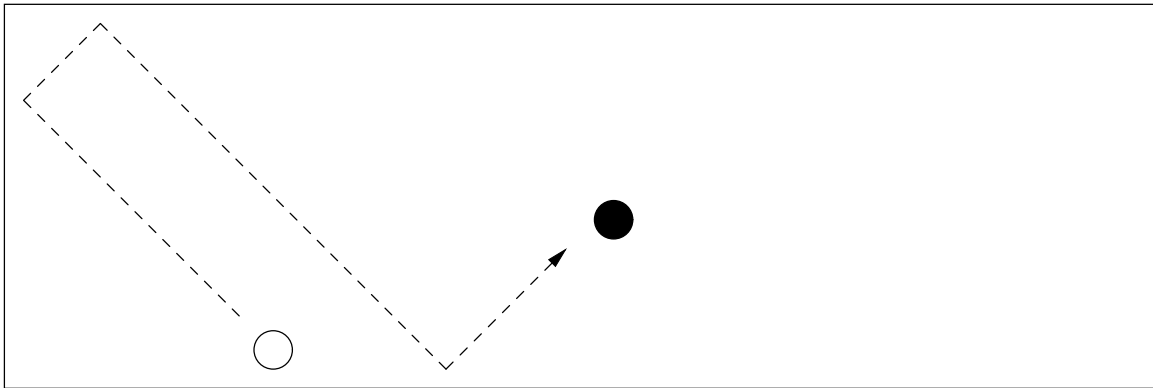
```
6 +0.110010_-1 +0.110110_0
4 -0.1001_0 +0.1000_4
```

Denoting the two problems  $0.0110010_2 + 0.110110_2$  and  $-0.1001_2 + 1000.0_2$  The results should be normalized values in the same format, one per line. For this example the results are as follows.

```
+0.101000_1
+0.1111_3
```

In this case, the mathematical results were  $1.001111_2$  and  $111.0111$ . Rounding to 6 and 4 binary digits gives us the actual results. You may assume that  $m < 256$ . Prefix all 0 values with a '+' sign and give them an exponent of 0. Input ends at the end of file.

2. A billiard ball is rolling on a frictionless, flat, rectangular table. There is another stationary ball somewhere on the table. You are to determine whether the balls collide for the first time after the first has rebounded from the cushions enclosing the table exactly three times. The situation might look like this, for example. The dashed arrow indicates the path of the center of the cue (white) ball, which banks (rebounds) three times and strikes the black ball.



The balls collide with each other when their centers are 2.0 units apart. The cue ball collides with a cushion if its center comes to within 1.0 unit of the cushion. As suggested by the diagram, the angle at which the cue ball collides with a cushion is equal to that at which it leaves the cushion. You may assume the data will not include cases where the ball hits two cushions simultaneously (as in a corner).

The input to your program consists of the following sequence of numbers separated by whitespace in free form.

- Floating-point numbers  $x_c$  and  $y_c$ , the initial position of the center of the cue ball. The point (0,0) is what appears in the diagram as the lower-left corner.
- Floating-point numbers  $x_s$  and  $y_s$ , the position of the center of the stationary ball.
- Floating-point numbers  $d_x$  and  $d_y$ , the initial direction in which the cue ball moves. That is, the cue ball initially moves so that at time  $t$ , it is at

$$(x_c + td_x, y_c + td_y).$$

At least one of  $d_x$  and  $d_y$  must be non-zero.

- Floating-point numbers  $l_x$  and  $l_y$ , the dimensions of the table in the  $x$  and  $y$  directions, respectively.

Your program should print either the message “Balls collide”, “Balls do not collide”, or “Balls collide before three bounces”, as appropriate.

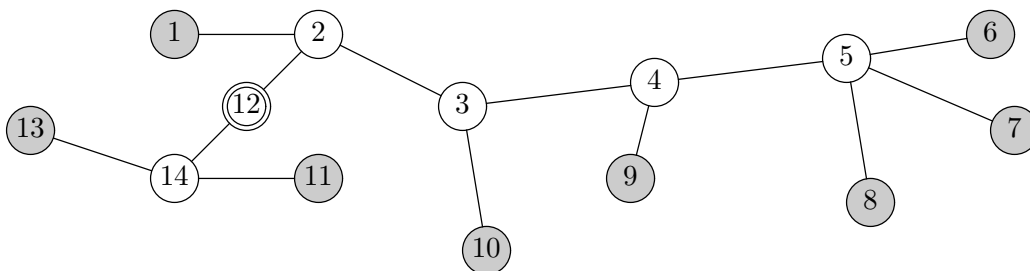
**Example:**

Input	Output
14.0 2.0 32.5 10.75	Balls collide
-2 2	
60 20	

3. [From an ACM Regional Contest in Seoul] Consider a tree network with  $n$  nodes where the internal nodes correspond to servers and the terminal nodes (those with exactly one parent) correspond to clients. The nodes are numbered from 1 to  $n$ . Among the servers, there is an original server  $S$  that provides VOD (Video On Demand) service. To ensure the quality of service for the clients, the distance from each client to the VOD server  $S$  should not exceed a certain value  $k$ . The distance from a node  $u$  to a node  $v$  in the tree is defined to be the number of edges on the path from  $u$  to  $v$ . If there is a nonempty subset  $C$  of clients such that the distance from each  $u$  in  $C$  to  $S$  is greater than  $k$ , then replicas of the VOD system have to be placed in some servers so that the distance from each client to the nearest VOD server (the original VOD system or a replica) is  $k$  or less.

Given a tree network, an original server  $S$ , and a positive integer  $k$ , find the minimum number of replicas necessary so that each client is within distance  $k$  of the original VOD system or a replica of it.

For example, consider the following tree network. The double-circled node is the original VOD server, and gray nodes are clients.



In the above tree, the set of clients is  $\{1, 6, 7, 8, 9, 10, 11, 13\}$ , the set of servers is  $\{2, 3, 4, 5, 12, 14\}$ , and the original VOD server is located at node 12.

For  $k = 2$ , the quality of service is not guaranteed with one VOD server at node 12 because the clients in  $\{6, 7, 8, 9, 10\}$  are more than 2 edges from a VOD server. Therefore, we need one or more replicas. When one replica is placed at node 4, the distance from each client to the nearest server in  $\{12, 4\}$  is less than or equal to 2. Therefore, the minimum number of the needed replicas is one for this example.

The input consists of one or more test cases on the standard input in free format. The number of test cases  $T > 0$  comes first. Each test case starts with an integer  $n$  ( $3 \leq n \leq 1000$ ), giving the number of nodes in that tree network. Next come two integers  $S$  ( $1 \leq S \leq n$ ) and  $k$  ( $k \geq 1$ ), giving the location of the original VOD server and the maximum distance value for ensuring the desired quality of service. Last come  $n - 1$  pairs of integer node numbers, representing the edges in the tree network.

The output consists of one line per test case containing the minimum number of the needed replicas (a non-negative integer).

**Example:**

Input	Output
2	1
14 12 2	0
1 2 2 3 3 4	
4 5 5 6 7 5	
8 5 4 9 10 3	
2 12 12 14 13 14	
14 11	
14 3 4	
1 2 2 3 3 4	
4 5 5 6 7 5	
8 5 4 9 10 3	
2 12 12 14 13 14	
14 11	