Task 1

Sandro is a well organised person. Every day he makes a list of things which need to be done and enumerates them from 1 to n. However, some things need to be done before others. In this task you have to find out whether Sandro can solve all his duties and if so, print the correct order.

Input

In the first line you are given an integer n and m (1<=n<=10000, 1<=m<=1000000). On the next m lines there are two distinct integers x and y, (1<=x,y<=10000) describing that job x needs to be done before job y.

Output

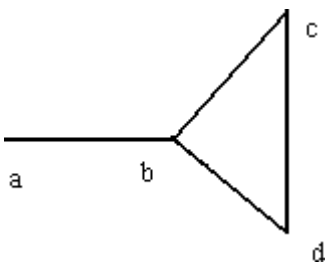
Print "Sandro fails." if Sandro cannot complete all his duties on the list. If there is a solution print the correct ordering, the jobs to be done separated by a whitespace. If there are multiple solutions print the one, whose first number is smallest, if there are still multiple solutions, print the one whose second number is smallest, and so on.

Sample Input	Sample Output
8 9 1 4 1 2 4 2 4 3 3 2 5 2 3 5 8 2 8 6 2 2 1 2 2 1	1 4 3 5 7 8 2 6 Sandro fails.

Task 2

A *graph* G is a set of point V(G), together with a set of edges E(G), where each element of E(G) is an unordered pair of distinct points of V(G).

Example 1: Let G be a graph where V(G) = $\{a, b, c, d\}$ and E(G) = $\{(a, b), (b, c), (c, d), (d, b)\}$. The figure gives a depiction of G.

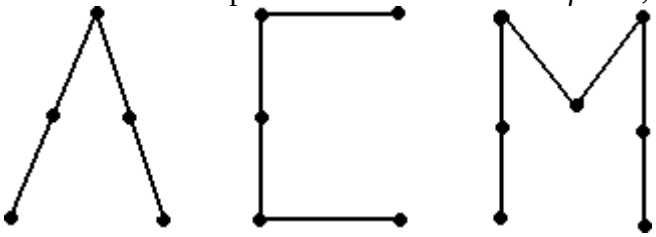


Notice that G contains the ``cycle" , $\{(b,c),(c,d),(d,b)\}$. A graph devoid of cycles is called a *tree*. A *path* in a graph G is an alternating sequence of points and edges, (beginning and ending with a point) such that all the points of the path are distinct. In the graph of example 1, $\{a,(a,b),b,(b,c),c,(c,d),d\}$ is a path.

Fact: Every two points of a tree are joined by a unique path.

A graph is called *connected* if every pair of points are joined by a path. The graph of example 1 is connected. If a graph is not connected then it is made up of ``subgraphs" which are. Each one of these subgraphs is called a *connected component* of the graph G.

A graph for which each connected component is a tree is called a *forest*, see figure below.



One extreme case worth mentioning is the case when one of the component trees has one point but no edges joined to it. This tree likes like an isolated dot. We will call this an *acorn*. We are ready to define the problem.

Problem: Given a forest you are to write a program that counts the number of trees and acorns.

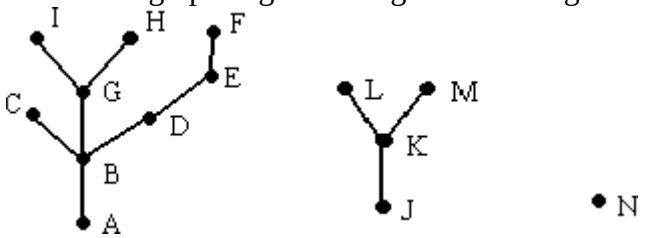
Input

The first line of the input file contains the number of test cases your program has to process. Each test case is a forest description consisting of two parts:
1. A list of edges of the tree (one per line, given as an unordered pair of capital letters delimited by a row of asterisks).
2. A list of points of the tree (these will be given on one line with a maximum of 26 corresponding to the capital letters, A - Z).

Output

For each test case your program should print the number of trees and the number of acorns, in a sentence, for example:
``There are *x* tree(s) and *y* acorn(s).", where *x* and *y* are the numbers of trees and acorns, respectively.

Example 2: Let G be a graph whose edges and points are given by the first test case in the sample input. A depiction of this graph is given in figure following.



Notes: A forest may have no trees and all acorns, all trees and no acorns, or anything inbetween, so keep your eyes open and don't miss the forest for the trees!

Sample Input	Sample Output
2 (A,B) (B,C) (B,D) (D,E) (E,F) (B,G) (G,H) (G,I) (J,K) (K,L) (K,M) **** A,B,C,D,E,F,G,H,I,J,K,L,M,N (A,B) (A,C) (C,F) ** A,B,C,D,F	There are 2 tree(s) and 1 acorn(s). There are 1 tree(s) and 1 acorn(s).