FORMULE

Consider a simpler variant of the problem with only one variable (say 0).

There are three cases:

1) The variable is not present in the formulas, or it can be deduced that 0 = 0
2) It can be deduced that 0 is a basic formula
3) It can be deduced that 0 is a function, but not a basic formula

The solution for the first case is trivial, for example 0 = a.

In the second case, since 0 is a basic formula, the problem comes down to simple string manipulation.

The third case is specific because no solution exists. Since the problem statements guarantees that a solution will exist, we can ignore this case.

To solve the original two-variable problem we need to find an identity variable=formula. Because a solution must exist, the formula on the right side of the identity will not contain the variable on the left side. We can thus substitute the variable from the identity into the given formulas and reduce the problem to the single variable variant we've already solved.

LUBENICA

The cities in the problem form a tree. We say that node A is a descendant of node B if A is in the subtree rooted in B. B is then called an ancestor of A. For simplicity, consider each node to also be its own ancestor and descendant.

Any two nodes A and B have at least one common ancestor. If there are more than one, there is a single one that is farthest from the root – it is called the least common ancestor of A and B.

In general, the path from A to B can be split in two parts, the path from A to their least common ancestor and then to B.

For each node A, let us keep track of its 2^i-th ancestor, for all i between 0 and 16. Call this *ancestor*(A, i). Other than the label of the ancestor, we also keep track of the shortest and longest roads on the path from A to it.

The following identity holds:

ancestor(A, i+1) = *ancestor*(*ancestor*(A, i), i)

If the nodes are not at the same depth, we raise the deeper one as fast as possible until it is. Once they are at the same depth, we raise them in parallel as fast as possible, but so that we delay reaching one of their common ancestors as long as possible. This way we'll surely end up in the least common ancestor of A and B.

By always choosing the largest possible power of two, the complexity becomes O(log N).