function `TREE-CSP-SOLVER(csp)` returns a solution, or failure
inputs: `csp`, a CSP with components `X, D, C`

```
function TREE-CSP-SOLVER(csp) returns a solution, or failure
inputs: csp, a CSP with components X, D, C
n number of variables in X
assignment 4 any empty assignment
root any variable in X
X TOPOLOGICALSORT(X, root)
for j = n down to 2 do
    MAKE-ARC-CONSISTENT(PARENT(Xj), Xj)
    if it cannot be made consistent then return failure
for i = 1 to n do
    assignment[i][Xj] any consistent value from Di
    if there is no consistent value then return failure
return assignment
```

Figure 6.11 The `TREE-CSP-SOLVER` algorithm for solving tree-structured CSPs. If the CSP has a solution, we will find it in linear time; if not, we will detect a contradiction.

deleting from the domains of the other variables any values that are inconsistent with the value chosen for SA.

Now, any solution for the CSP after SA and its constraints are removed will be consistent with the value chosen for SA. (This works for binary CSPs; the situation is more complicated with higher-order constraints.) Therefore, we can solve the remaining tree with the algorithm given above and thus solve the whole problem. Of course, in the general case (as opposed to map coloring), the value chosen for SA could be the wrong one, so we would need to try each possible value. The general algorithm is as follows:
Section 6.5. The Structure of Problems

1. Choose a subset $S$ of the CSP's variables such that the constraint graph becomes a tree after removal of $S$. $S$ is called a cycle cutset.

2. For each possible assignment to the variables in $S$ that satisfies all constraints on $S$,
   
   (a) remove from the domains of the remaining variables any values that are inconsistent with the assignment for $S$, and
   
   (b) If the remaining CSP has a solution, return it together with the assignment for $S$.  

If the cycle cutset has size $c$, then the total run time is $O(d^n - (c + 1)d^c)$; we have to try each of the $d^c$ combinations of values for the variables in $S$, and for each combination we must solve a tree problem of size $n - c$. If the graph is "nearly a tree," then $c$ will be small and the savings over straight backtracking will be huge. In the worst case, however, $c$ can be as large as $(n - 2)$. Finding the smallest cycle cutset is NP-hard, but several efficient approximation algorithms are known. The overall algorithmic approach is called cutset conditioning; it comes up again in Chapter 14, where it is used for reasoning about probabilities.

The second approach is based on constructing a tree decomposition of the constraint graph into a set of connected subproblems. Each subproblem is solved independently, and the resulting solutions are then combined. Like most divide-and-conquer algorithms, this works well if no subproblem is too large. Figure 6.13 shows a tree decomposition of the map-coloring problem into five subproblems. A tree decomposition must satisfy the following three requirements:

- Every variable in the original problem appears in at least one of the subproblems.
- If two variables are connected by a constraint in the original problem, they must appear together (along with the constraint) in at least one of the subproblems.
- If a variable appears in two subproblems in the tree, it must appear in every subproblem along the path connecting those subproblems.

The first two conditions ensure that all the variables and constraints are represented in the decomposition. The third condition seems rather technical, but simply reflects the constraint that any given variable must have the same value in every subproblem in which it appears; the links joining subproblems in the tree enforce this constraint. For example, 5.4 appears in all four of the connected subproblems in Figure 6.13. You can verify from Figure 6.12 that this decomposition makes sense.

We solve each subproblem independently; if any one has no solution, we know the entire problem has no solution. If we can solve all the subproblems, then we attempt to construct a global solution as follows. First, we view each subproblem as a "mega-variable" whose domain is the set of all solutions for the subproblem. For example, the leftmost subproblem in Figure 6.13 is a map-coloring problem with three variables and hence has six solutions—one is $\{WA = \text{red}, SA = \text{blue}, NT = \text{green}\}$. Then, we solve the constraints connecting the subproblems, using the efficient algorithm for trees given earlier. The constraints between subproblems simply insist that the subproblem solutions agree on their shared variables. For example, given the solution $\{WA = \text{red}, SA = \text{blue}, NT = \text{green}\}$ for the first subproblem, the only consistent solution for the next subproblem is $\{SA = \text{blue}, NT = \text{green}, Q = \text{red}\}$.

A given constraint graph admits many tree decompositions; in choosing a decomposition, the aim is to make the subproblems as small as possible. The tree width of a tree...