(a) Let us root the tree at an arbitrary node r, and define subtrees T_u as we have done in Chapter 10.2 when solving the Weighted Independent Set Problem. There we defined two subproblems corresponding to each subtree depending on whether or not we include the root u in the set. We will use the same subproblems: $OPT_{in}(u)$ denotes the maximum weight of an independent set of T_u that includes u, and $OPT_{out}(u)$ denotes the maximum weight of an independent set of T_u that does not include u. Now the optimum we are looking for is $\min(OPT_{in}(r), OPT_{out}(r))$. It helps us to define a third subproblem for each subtree: $OPT_{un}(u)$ denotes the maximum weight of an independent set of T_u that does not have to dominate u. When u's parent is included in the dominating set, u is already dominated by its parent, hence the set selected in the subtree T_u does not have to dominate u.

Now that we have our sub-problems, it is not hard to see how to compute these values recursively. For a leaf $u \neq r$ we have that $OPT_{out}(u) = \infty$, $OPT_{in}(u) = c(u)$, and $OPT_{un}(u) = 0$. For all other nodes u we get a recurrence that defines $OPT_{out}(u)$, $OPT_{in}(u)$, and $OPT_{un}(v)$ using the values for u's children.

(1) *For a node u, the following recurrence defines the values of the sub-problems:*

- $OPT_{in}(u) = c(u) + \sum_{v \in children(u)} OPT_{un}(v)$
- $OPT_{un}(u) = \sum_{v \in children(u)} \min(OPT_{in}(v), OPT_{out}(v))$
- $OPT_{out}(u) = \min_{v \in children(u)} (OPT_{in}(v) + \sum_{w \in children(u), w \neq v} \min(OPT_{out}(v), OPT_{in}(v)))$.

Using this recurrence, we get a dynamic programming algorithm by building up the optimal solutions over larger and larger sub-trees. We define arrays $Mo[u]$, $Mi[u]$ and $Mu[u]$, which hold the values $OPT_{out}(u)$, $OPT_{in}(u)$ and $OPT_{un}(u)$ respectively. For building up solutions, we need to process all the children of a node before we process the node itself.

```
To find a minimum-weight dominating set of a tree T:
    Root the tree at a node r.
    For all nodes u of T in post-order
        If u is a leaf then set the values:
            Mo[u] = ∞
            Mi[u] = c(u)
            Mu[u] = 0
        Else set the values:
            Mi[u] = c(u) + Σ_{v∈children(u)} Mu[v].
            Mu[u] = Σ_{v∈children(u)} min(Mi[v], Mo[v]).
```

[1] ex573.411.14

$$Mo[u] = \min_{v \in children(u)} Mi[v] + \sum_{w \in children(u), w \neq v} \min(Mo[v], Mi[v])$$

```
    Endif
Endfor
Return max(Mo[r], Mi[r]).
```

The algorithms clearly runs in polynomial time, as there are $3n$ subproblems for an n node tree, and each value associated with a subproblem of u of degree n_u can be determined in $O(n_u)$ time. So the total time is $O(\sum_u n_u) = O(n)$.

(b) We extend the algorithm for the case of bounded tree-width by having subproblems associated with the nodes of the tree-decomposition. For each node t of the tree-decomposition, let V_t be the subset of nodes corresponding to tree-node t, T_t the subtree rooted at t, and G_t the corresponding subgraph. Now for each disjoint sets $U, W \subset V_t$ we will define a subproblem and have $OPT(t, U, W)$ the minimum weight of a set in G_t that contains exactly the nodes U in V_t and covers all nodes of G_t except possibly not dominating the a subset of W. Recall that in the case of a tree, the recurrence for $OPT_{out}(u)$ was a little awkward, as we needed to select a child of u that is covering u. Here we would need to do this for each node in $V_t - (U \cup W)$. To make this simpler to write, we will define more subproblems. Let t be a node of the tree decomposition, and let $t_1, \ldots, t_d$ be its children, then we define subproblems $OPT(t, i, U, W)$ for each $0 \leq i \leq d$ to be the minimum weight of a set in the graph corresponding to the union of subtrees $T_{t_1}, \ldots, T_{t_i}$ and the set V_t that contains exactly the nodes U in V_t and dominates all nodes of this subgraph except possibly not dominating the a subset of W.

So if d_r is the degree of the root, then the optimum value we are looking for is $\min_{U \subset V_r} OPT(r, d_r, U, \emptyset)$ For any node t we have $OPT(t, 0, U, W) = \sum_{u \in U} c(u)$ if U dominates the nodes $V_t - W$, and ∞ otherwise. This defines the values at the leafs.

Given the subproblems, we will get the value at a node t using the values for smaller i, and the values at the children of t as follows. For a set U we will use the notation $c(U) = \sum_{u \in U} c(u)$, and $\delta(U)$ is the set dominated by U. Let t be a node of the tree decomposition and $t_1, \ldots, t_d$ its children, let n_i be the degree of child i.

(2) *The value of $OPT(t, i, U, W)$ for $i \geq 1$ is given by the following recurrence:*

$$\begin{aligned} OPT(t, i, U, W) &= c(U) + \min_{U_i \subseteq V_{t_i} : U_i \cap V_t = U \cap V_{t_i}} (OPT(t, i-1, U, W \cup \delta(U_i)) \\ &\quad + OPT(t_i, n_i, U_i, (\delta(U) \cup W) \cap V_{t_i}) - c(U \cap V_{t_i})) \end{aligned}$$

For a tree-decomposition of width k there are 3^{k+1} subproblems associated with a (t, i) pair, and there are n such pairs if the tree is of size n. Computing each value takes only $O(1)$ time, so the total time required is $O(3^k n)$.