1. A set $S \subseteq V$ is a dominating set in a graph G if each vertex $v \in V$ is either in S or has a neighbor in S. We consider the case that G is a tree.

(a) (4 points) Give a recursive formulation to determine the size of the minimum dominating set.

Solution:

- the function $OPT_{in}(u)$ will be used to return the minimum dominating set including u,
- the function $OPT_{out\_not\_dominated}(u)$ will be used to return the minimum dominating set where u is not in the dominating set and still needs to be dominated, and
- the function $OPT_{out\_dominated}(u)$ will be used to return the minimum dominating set where u is not in the dominating set but is already dominated.

Each of these functions is defined as follows:

- $OPT_{in}(u) = 1 + \sum_{v \in child(u)} \min \{OPT_{in}(v), OPT_{out\_dominated}(v)\}$
- $OPT_{out\_not\_dominated}(u)$
$= \min_{v \in child(u)} \left\{OPT_{in}(v) + \sum_{w \in child(u), w \neq v} \min \{OPT_{in}(w), OPT_{out\_not\_dominated}(w)\}\right\}$
- $OPT_{out\_dominated}(u) = \sum_{w \in child(u)} \min \{OPT_{in}(w), OPT_{out\_not\_dominated}(w)\}$

In the above we take $\min(\emptyset) = \infty$, so if u is a leaf node then $OPT_{out\_not\_dominated}(u) = \infty$.

The answer of a tree rooted in r is given by $\min \{OPT_{in}(r), OPT_{out\_not\_dominated}(r)\}$.

================

Alternatively, with a bit different notation, we could define the third function to still have the option of the root being included:

- $OPT_{in}(u)$ denotes the minimum weight of the dominating set for the subtree of u where u is included in S,
- $OPT_{dom}(u)$ denotes the minimum weight of the dominating set for the subtree of u where u is not included in S and needs to be dominated by its children, and
- $OPT_{un}(u)$ denotes the minimum weight of the dominating set for the subtree of u where u is already dominated by its parent.

For leaf nodes u, $OPT_{in}(u) = w_u$ and $OPT_{dom}(u) = \infty$ and $OPT_{un}(u) = 0$. For other nodes, these functions are defined as follows:

- if u is included, include its weight; the children are then dominated: $OPT_{in}(u) = w_u + \sum_{v \in child(u)} OPT_{un}(v)$

- if u still needs to be dominated, it should be done by one of its children; the others still need to be dominated: $OPT_{dom}(u) = \min_{v \in child(u)} \left(OPT_{in}(v) + \sum_{w \in child(u) \setminus \{v\}} OPT_{dom}(w) \right)$
- if u is already dominated, either include it anyway, or make sure all children will be dominated: $OPT_{un}(u) = \min \left\{ OPT_{in}(u), \sum_{w \in child(u)} OPT_{dom}(w) \right\}$

The answer for the root of the tree is then given by $\min \{OPT_{in}(r), OPT_{dom}(r)\}$.

(b) (2 points) Give an analysis of a tight upper bound on the runtime of a dynamic programming implementation of this function.

Solution: We store solutions to $2 \cdot n$ subproblems. Computing them costs amortized constant time, so $O(n)$.