Exercise 13.6. Recall the dominating set problem[†] from section 13.3. Here we will consider the weighted version where the vertices are given positive weights, and the goal is to compute the minimum weight dominating set. For each of the following problems, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

[†]A set of vertices $S \subseteq V$ is a *dominating set* if every vertex $v \in V$ is either in S or the neighbor of a vertex in S. The minimum dominating set problem is to compute the minimum cardinality dominating set.

Exercise 13.6.1. The minimum weight dominating set problem for intervals, *with the additional assumption that* no two intervals are nested. To state it more precisely: the input consists of n weighted intervals $\mathcal{I}$. The non-nested assumptions means that for any two intervals $I, J \in \mathcal{I}$, we never have I contained in J or J contained in I.

The goal is to compute the minimum weight subset $S \subseteq \mathcal{I}$ of intervals such that every interval in $\mathcal{I}$ is either in S or overlaps some interval in S.

- (For 1 pt. extra credit) Extend your algorithm to general intervals.[8]

[8]Of course, anyone who has already solved the general case automatically solves the special case where no two intervals are nested.

Solution. We claim that the following algorithm suffices as a polynomial time solution.

Preprocessing. Sorting the intervals $\mathcal{I}$ by left endpoint $\in \mathcal{O}(n \log n)$ ■

Algorithm. $\underline{\texttt{MWDS}(\mathcal{I} = \{I_1, I_2 ..., I_n\}, w = \mathcal{I} \to R_{>0}):}$

/ given n non-nested weighted intervals in $\mathcal{I}$ sorted by first endpoint, computes the minimum weight subset $S \subseteq \mathcal{I}$ such that every interval in $\mathcal{I}$ is in either S or overlaps some interval in S */*

1. If $n = 0$ then return $\emptyset, 0$.
2. Let $J_1 = \{I_j \in \mathcal{I} : I_j \neq I_1 \text{ and } I_j \text{ overlaps } I_1\}$
3. If $\mathcal{I} \setminus (J_1 \cup I_1) == \emptyset$:
   A. return min-weight $\begin{cases} I_1, w(I_1) \\ \underline{\texttt{MWDS}(\mathcal{I} \setminus I_1, w)} \end{cases}$
4. Else
   A. $S_{rem}, w_{rem} \leftarrow \underline{\texttt{MWDS}(\mathcal{I} \setminus (J_1 \cup I_1), w)}$
   B. $S_{inc}, w_{inc} \leftarrow \{I_1\} \cup S_{rem}, w(I_1) + w_{rem}$
   C. If $J_1 \neq \emptyset$:
      1. return min-weight $\begin{cases} S_{inc}, w_{inc} \\ \underline{\texttt{MWDS}(\mathcal{I} \setminus I_1, w)} \end{cases}$
   D. Else return S_{inc}, w_{inc}

■

Runtime and Caching. The above algorithm can be implemented by using caching the resulting subset and weight for every subproblem on an interval $\underline{\texttt{MWDS}(\mathcal{I}, w)}$ to avoid recomputation.
The number of subproblems due to non-nested intervals property is O(n) and time per subproblem is O(n) to find all the overlapping intervals with I_1. Hence, the runtime for $\underline{\texttt{MWDS}(\mathcal{I}, w)}$ is

$$(\#\text{ of subproblems})(\#\text{ time per subproblem}) = \mathcal{O}(n)\mathcal{O}(n) = \mathcal{O}(n^2)$$

Making our **Total Runtime** $\mathcal{O}(n \log n) + \mathcal{O}(n^2) \in \mathcal{O}(n^2)$

■

How to use the algorithm. We need to first sort our intervals $\mathcal{I}$ by left endpoint and then call `MWDS(`$\mathcal{I}, w$`)`

■

- *Solution.* Algorithm for general intervals... a polynomial time algorithm for minimum weight dominating set for general intervals would imply a polynomial for SAT, since the number of subrpoblems can grow exponentially as an interval can be contained by another and increasing the recursion stack? □

□

Exercise 13.6.2. The minimum weight dominating set problem in trees.

Solution. We claim that the following algorithm suffices as a polynomial time solution.

MWDST$(T, v, exclude-root)$:

/* *given a tree $T = (V, E)$, vertex $v \in V$, weights $w : V \to \mathbb{R}$ and boolean exclude-root indicating whether v is excluded from the dominating set, computes the minimum weighted dominating subset in T* */

1. If v is a leaf:
   A. If exclude-root return $\emptyset, \infty$
   B. Else return $\{v\}, w(v)$
2. If exclude-root:
   A. return union of subsets and sum of weights MWDST$(T, c, False)$ for all children c of v
3. Else
   A. $S_{children}, w_{children} \to$ union and sum of MWDST$(T, c, True)$ for all children c of v
   B. $S_{inc} \leftarrow \{v\} \cup S_{children}, w_{inc} \leftarrow w(v) + w_{children}$
   C. $S_{exc}, w_{exc} \leftarrow$ Union and sum of MWDST$(T, c, False)$ for all children c of v
   D. return return min-weight $\begin{cases} S_{inc}, w_{inc} \\ S_{exc}, w_{exc} \end{cases}$

Runtime and Caching. The above algorithm for trees can be implemented by caching the resulting subset and weight for every subproblem on a tree MWDST$(T, v, exclude-root)$ to avoid recomputation.

The number of subproblems is O(n) since there are n vertices in the tree, and each vertex can be in two states (excluded or not excluded). Hence, there are at most 2n distinct subproblems.

The time per subproblem is to process all children of vertex v.

Hence, the runtime for MWDST$(T, v, exclude-root)$ with memoization is

$$\begin{gathered}(\#\text{ of subproblems})(\#\text{ time per subproblem}) \\ \textstyle\sum_v(\text{time spent on MWDST}(T, v..)) = O\left(\sum_v \#\text{ children of } v\right) = O(n)\end{gathered}$$

Making our **Total Runtime** $\mathcal{O}(n)$ We also mention that the space usage is O(n) because there are O(n) subproblems, each requiring constant space.

■

How to use the algorithm. We need to first choose an arbitrary vertex as the root of the tree (if the tree is not already rooted). Then, we call MWDST$(T, root, False)$ to compute the minimum weight dominating set for the entire tree.

The algorithm will return the minimum weight dominating set and its corresponding weight. If we want to allow the possibility that the root is excluded, we can take the minimum of MWDST$(T, root, False)$ and MWDST$(T, root, True)$. ■

□