

Find the weight of the minimum weight dominating set of size at least  $k$ .

First, we define an algorithm to find the minimum weight dominating set of size exactly  $k$ .

Define relations:

$$\text{OPT}_0(r, k) = \min \begin{cases} A + w(r) \\ \min_{v \in N(r)} B \end{cases}$$

$$\text{OPT}_1(r, k) = C + w(r)$$

$$\text{OPT}_2(r, k) = \min \begin{cases} D + w(r) \\ E \end{cases}$$

$$n = |N(r)|$$

$$A = \min_{k_1, \dots, k_n} \left[ \sum_{x=1}^n \text{OPT}_2(N(r)_x, k_x) \right], \sum_{i=1}^n k_i = k - 1$$

$$B = \min_{k_1, \dots, k_n} \left[ \text{OPT}_1(v, k_1) + \sum_{x=2}^n \text{OPT}_0((N(r) \setminus \{v\})_x, k_x) \right], \sum_{i=1}^n k_i = k$$

$$C = \min_{k_1, \dots, k_n} \left[ \sum_{x=1}^n \text{OPT}_2(N(r)_x, k_x) \right], \sum_{i=1}^n k_i = k - 1$$

$$D = \min_{k_1, \dots, k_n} \left[ \sum_{x=1}^n \text{OPT}_2(N(r)_x, k_x) \right], \sum_{i=1}^n k_i = k - 1$$

$$E = \min_{k_1, \dots, k_n} \left[ \sum_{x=1}^n \text{OPT}_0(N(r)_x, k_x) \right], \sum_{i=1}^n k_i = k$$

Note that the values of these expressions  $A$  to  $E$  can be computed by the algorithm from problem 1. Input the target sum of the  $k_i$ s as the target sum  $k$  from problem 1. For the set of  $T$  functions, there will be a  $T$  for each neighboring vertex and for a neighboring vertex  $v$ ,  $T_v(x)$  will be  $\text{OPT}_t(v, x)$ , with  $t$  being chosen depending on which  $\text{OPT}$  is being used for  $v$  in one of  $A$  to  $E$ .

To implement this as a dynamic programming solution, we have to examine the dependency graph. When looking at the vertex parameter, since any vertex only depends on  $\text{OPT}$  values of its children, we can compute the sets of values for the vertices in post-order, which ensures that any vertex will have the  $\text{OPT}$  values of its children available when it is being computed.

For the  $k$  parameter, since for any  $\text{OPT}_a(v, b)$  is never dependent on  $\text{OPT}_c(v, d)$  for any  $(a, b, c, d)$ , but only on values less than  $v$ , the entries of a row corresponding to any vertex can be computed in any order.

The base cases for the recurrence are the leaves of the tree. For all leaves  $v$ ,

$$\text{OPT}_0(v, 0) = \text{OPT}_1(v, 0) = \infty$$

$$\begin{aligned}
\text{OPT}_0(v, 1) &= \text{OPT}_1(v, 1) = w(v) \\
\text{OPT}_0(v, x > 1) &= \text{OPT}_1(v, x > 1) = \infty \\
\text{OPT}_2(v, 0) &= 0 \\
\text{OPT}_2(v, 1) &= w(v) \\
\text{OPT}_2(v, x > 1) &= \infty
\end{aligned}$$

The solution to the minimum dominating set of size exactly  $k$  is  $\text{OPT}_0(r, k)$ , where  $r$  is the root node.

**Time complexity analysis:**

First, we compute a post-order traversal order for the tree, which takes  $O(n)$  time.

Next, we fill out 3  $n$  by  $k$  tables, where  $n = |V|$ . For each set of 3 corresponding entries in the three OPT tables, we perform the algorithm from problem 1. For A, C, D, and E, this is done once per vertex. For B, this is done  $|N(v)|$  times per vertex  $v$ . However, there are only  $n - 1$  total neighbors, so the total number of times B is also only  $O(nk)$ . The overall runtime for this section is thus  $O(nk * (nk^2)) = O(n^2k^3)$ .

We then repeat the entire algorithm for target sum values of  $k$  up to  $n$ , or  $n - k$  times, to find the minimum weight dominating set of size at least  $k$ . This yields a total runtime of  $O(n^2k^3(n - k))$ .