Problem 13.6.1

First sort the intervals based on the left boundary, and add another interval to the right of all the others with weight 0.

Let $f(i)$ be the minimum weight of a dominating set that includes the ith interval, and covers all previous intervals.

To calculate $f(i)$, note that if all previous intervals intersect with the ith one, then we can set $f(i) = w[i]$. Otherwise, let q be the highest index of an interval that does not intersect with i. It is easy to see that in a dominating set containing interval i, the interval before must either have index $x \geqslant q$, or intersect with the qth interval. Hence

$$f(i) = w[i] + \min_{1 \leqslant x < i} (f(x) \text{ if } x \geqslant q \text{ or the } x\text{th and } q\text{th intervals intersect})$$

The answer will just be $f(n+1)$, and the time complexity is $O(n^2)$ because there are $O(n)$ values of $f(i)$ to compute, and each one requires $O(n)$ time to determine q as well as scan for a minimum.

Problem 13.6.2

Root the tree on the first vertex. For each i, let $f(i)$, $g(i)$, and $h(i)$ represent the size of the smallest subset of the subtree of i, denoted as S, such that $S \cup N(S)$ includes all **descendants** of i, where

- $f(i)$ has no additional restriction,
- $g(i)$ requires $i \in S \cup N(S)$ as well, and
- $h(i)$ requires $i \in S$.

Then

$$\begin{aligned} h(i) &= w[i] + \sum_c f(c) \\ g(i) &= \min(h(i), \min_c(h(c) - g(c)) + \sum_c g(c)), \quad \text{and} \\ f(i) &= \min(g(i), h(i), \sum_c g(c)) \end{aligned}$$

(If i is a leaf then $g(i) = h(i) = w[i]$).

The equations for h and f are self-evident. To reach the case g when i is not in the set, the children must be either $g(c)$ or $h(c)$, but there needs to be at least one $h(c)$. As $h(c) \geqslant g(c)$ is more restrictive, we should start with a baseline of $\sum_c g(c)$ and

then add the minimum possible $h(c) - g(c)$.

By processing the vertices in reverse DFS order, we can get the answer, $g(1)$, in $O(n)$ time. I wrote some Python code to sanity check my equations:

```
n = int(input())
w = [int(t) for t in input().split()]
edges = [[] for _ in range(n)]
for _ in range(n-1):
    a,b = [int(t)-1 for t in input().split()]
    edges[a].append(b)
    edges[b].append(a)

#dfs
par = [0]*n
stack = [0]
order = []
while stack:
    a = stack.pop()
    p = par[a]
    order.append(a)
    for b in edges[a]:
        if b == p: continue
        par[b] = a
        stack.append(b)

#dynamic programming
f = [0]*n
g = [0]*n
h = [0]*n
for a in reversed(order):
    if a and len(edges[a]) == 1:
        f[a] = 0
        g[a] = h[a] = w[a]
    else:
        p = par[a]
        h[a] = w[a] + sum(f[b] for b in edges[a] if b^p)

        gsum = sum(g[b] for b in edges[a] if b^p)
        g[a] = min(h[a], min(h[b]-g[b] for b in edges[a] if b^p) + gsum )
        f[a] = min(g[a],h[a],gsum)
print(g[0])
```