Computing Depth & Height

→ let p be position of a node of a tree T.
the depth of p is # of ancestors of p, excluding p itself.
the depth of p can be recursively defined as:
- if p is root, then the depth is 0
- otherwise, depth of p is one plus the depth of parent p

```
def(self, p):
  if self.is_root(p)
    return 0
  else
    return 1 + self.depth(self.parent(p))
```

running time for T.depth(p) is $O(d_p + 1)$ where d denotes depth because algo performs constant-time recursive step for each ancestor of p

→ height of a position p in a tree T is defined recursively:
- if p is a leaf, height is zero
- otherwise, height is one more than max heights of p's children

[redacted]

→ height of a nonempty tree T is equal to the maximum of the depths of leaf positions

```
def height(self, p):
  if self.is_leaf(p)
    return 0
  else return 1 + max(self.height(c) for c in self.children(p))
```

[redacted]

recursive, progresses in a top-down fashion; eventually called once for each position of T
the root eventually invokes the recursion on each of its children, then their children, etc.

→ Let T be a tree with n positions, and let c_p denote the # of children of a position p of T. Then, summing over the positions of T, $\sum_p c_p = n - 1$
- each position of T, with exception of root, is a child of another position and thus contributes one unit to the above sum