5 Forests and binary trees

5.1 Forests

A *forest* is a finite set V, together with a *partial* function parent : $V \nrightarrow V$. Elements of V are called *nodes*. There is also a crucial *acyclicity condition*, given below.

A node whose parent is undefined is called a *root*.

The *ancestors* of a node u are, informally,

$$u, \text{parent}(u), \text{parent}(\text{parent}(u)) \ldots$$

Formally: The ancestors of u form the smallest set X of nodes such that

- $u \in X$, and
- If $v \in X$ and parent(v) is defined then parent$(v) \in X$.

The *proper* ancestors of u are the ancestors of parent(u), if u has a parent. If not, then u is a root, and has no proper ancestor.

There is a very important *acyclicity* condition:

No node is a *proper* ancestor of itself.

5.2 Tree, ancestor, child, descendant, depth

A *tree* is a forest which is either empty or contains exactly one root.

(5.1) Lemma *Let u, v, w be nodes in a forest where both v and w are ancestors of u. Then either v is an ancestor of w or w is an ancestor of v.*

Proof. Write out the list of ancestors of u:

$$u, \text{parent}(u), \ldots,$$

Either v appears before w in this list or v appears after w in this list or $v = w$. In the first case, w is an ancestor of v. In the second, v is an ancestor of w. In the third, each is an ancestor of the other. ∎

(5.2) Lemma *Let u be a node in a forest. Then exactly one ancestor of u is a root node.*

Proof. If no ancestor of u is a root, then there would be an infinite list (with repetitions) of ancestors of u; There would be at least one node w which occurs more than once in this list, and w would be a proper ancestor of itself. This contradiction shows that some ancestor of u is a root node. It cannot have two such ancestors, since one would be a proper ancestor of the other, which is impossible. ∎

(5.3) Definition *The* depth *of a node u is the number of* proper *ancestors possessed by u. In particular, a root node has depth zero.*

A child *of a node u is any node v such that* $\text{parent}(v) = u$.

A leaf *is a node with no children.*

A descendant *of a node u is any node v such that u is an ancestor of v.*

(5.4) Lemma

$$\text{depth}(u) = \begin{cases} 0 & \text{if } u \text{ is a root} \\ 1 + \text{depth}(\text{parent}(u)) & \text{otherwise.} \end{cases}$$

(Easy). ∎

5.3 Binary trees

A *binary tree* T consists of a finite set V of nodes with three *partial* functions, parent, lchild, rchild : $V \nrightarrow V$ from nodes to nodes, such that

- V + parent function is a tree (forest which is empty or has just one root)
- If u is a node and v=lchild(u) is defined, then $u = \text{parent}(v)$.
- If u is a node and v=rchild(u) is defined, then $u = \text{parent}(v)$.
- If both children are defined, $v = \text{lchild}(u)$ and $w = \text{rchild}(u)$, then $v \neq w$.

5.4 Size and depth inequalities in binary trees

(5.5) Lemma *Let T be a binary tree and $r = 0, 1, \ldots$. Then T contains at most 2^r nodes at depth r.*

Proof. There is at most one node at depth 0, i.e., at most one root. Assume by induction that there are at most 2^r nodes at depth r.

The nodes at depth $r + 1$ are all children of nodes at depth r, and every node has at most two children, so there are at most 2×2^r nodes at depth $r + 1$. ∎

(5.6) Definition *The depth of a tree is the maximum depth of all its nodes (defaulting to -1 for empty trees).*

(5.7) Lemma *Let W be a set of nodes in a binary tree T. Let d be the maximum depth of all nodes in W (default -1 if $W = \emptyset$). Then*

$$|W| \leq 2^{d+1} - 1.$$

Proof. For $0 \leq r \leq d$, W contains at most 2^r nodes at depth r. Adding, we get the result. ∎

(5.8) Corollary *Let W be a set of nodes, and let d be the maximum depth of nodes in W. Then*

$$d \geq \log_2(1 + |W|) - 1. \quad ∎$$

We shall use that fact — later — to show that $O(n \log n)$ is the best performance one can expect for sorting algorithms.

(5.9) Corollary *A tree of depth d has at most*

$$2^{d+1} - 1$$

nodes overall. $\blacksquare$