**Trees**

Abstractly speaking, a **tree** is a connected undirected graph $T$ with no cycles. Equivalently, $T$ is a connected graph with $n$ nodes and $n - 1$ edges, for some (finite) number $n$.

A **rooted tree** distinguishes one node $r$ as being the **root**. The root $r$ is usually portrayed as being uppermost, i.e., trees "grow down"---like with genealogical trees. Then edges are regarded as radiating away from the root, so that they become directed edges after all. (But, often we implement trees with links going both ways---so don't insist on calling them directed graphs.) Here is a "general tree":

- Every node $u$ other than the root has exactly one edge to a node $v$ that is closer to $r$. (In the case of $u$ shown, $v$ equals $r$ itself.) Then $v$ is called the **parent** of $u$.
- A node with no edges going away from $r$ (in the directed view we say: having no out-edges), is a **leaf**. Note that $u$ is a leaf, even though it is closer to $r$ than many other nodes.
- Every non-leaf node, including the root, is called an **interior node**.
- Every interior node $w$ has one or more **children**, meaning nodes connected by edges going away from the root. The node $w$ shown has just one child (which happens to be an interior node), while $x$ has three children (which all happen to be leaves). Fellow children of the same node are called **siblings**, and the terms **descendant** and **ancestor** have their familiar meanings from genealogy.
- The **valence** of a node is its number of children. (The word **degree** is often used interchangeably with valence, but technically they are equal only if the tree is regarded as a directed graph.)
- A **binary tree** is one in which every interior node---including the root---has valence 2. A **unary-binary** tree allows nodes of valence 1 too. **Binary search trees (BSTs)** are in fact unary-binary trees.
- The **depth** of a node $u$ is the number of edges on the path from $u$ to the root. Nodes of equal depth, whether siblings or not, are on the same **level**.
- The **height** of a node $u$ can be reckoned two ways: as the maximum distance to a leaf **below** $u$,
or (as the text says), the difference between the maximum depth in the tree and the depth of \( u \).

- The height \( h \) of the whole tree, however, is always the same as the maximum depth of a leaf. The height of the above tree is 3. The height of \( x \) is 1 above its own leaves, but 2 overall. You can call \( u \) in the above tree a "high leaf".

Now for a notion that (IMHO strangely) does not have a standard name and which the text takes for granted on page 494: A tree is "sequenced" if the children of every internal node are listed in sequence. The sequence is usually written left-to-right. This specifies a unique layout for the tree in two dimensions, as in the above diagram. The sequencing should not be confused with the different traversal orders defined below. The sequencing determines those orders, but is not the same as them.

A path is a special case of a tree where there is one leaf at the end, the root at the beginning, and every other node has one parent edge toward the root and one edge away from it. This is the graph of a linked list. If you have a singly linked list, then each edge is directed away from the root. If you have a doubly linked list, then it's an undirected path. (If you have a circularly linked list, then the graph is a cycle, which is not a tree.)

Implementing Trees

As with lists, trees can be implemented via direct recursion without needing a separate concept of Node. Immutable trees are done that way. But we will mainly use standard mutable trees with nodes. Among them there are several choices:

1. Every node in the tree uses the same Node class, which has:
   (a) a list (or other Sequence) of children; leaves have the empty list.
   (b) optionally, a parent link.
   (c) usually, a data item. Sometimes data is only in the leaves, or sometimes internal nodes have a different kind of item (such as an operator) from the leaves.
   (d) Associating data with edges, say in the form of a weight, is rarer, so unlike with general
graphs there isn't usually a separate Edge class.

2. Leaves can use a separate Leaf class.

3. A binary tree usually specifies the children of a node as left and right rather than as a list. The text puts null pointers in those fields for leaves, and for one of those fields in the case of a unary node in a unary-binary tree. An alternative is to use a sentinel end node.

4. The "Circularly Linked Tree" or "CLR tree", used in the famous "MIT White Book" by Cormen, Leiserson, and Rivest (and now Stein) and by the C++ Standard Template Library, makes the root and the sentinel end node be parents of each other, too.

Traversals

- Level order, which is what BFS does.
- Preorder
- Postorder --- not the same as mirror-image of preorder, still L to R.
- Inorder (well-defined for binary and unary/binary trees).

Visualizations: [Geeks for Geeks](https://www.programiz.com/dsa/tree-traversal)
[https://www.programiz.com/dsa/tree-traversal](https://www.programiz.com/dsa/tree-traversal)
[http://ceadserv1.nku.edu/longa/classes/mat385_resources/docs/traversal.htm](http://ceadserv1.nku.edu/longa/classes/mat385_resources/docs/traversal.htm)