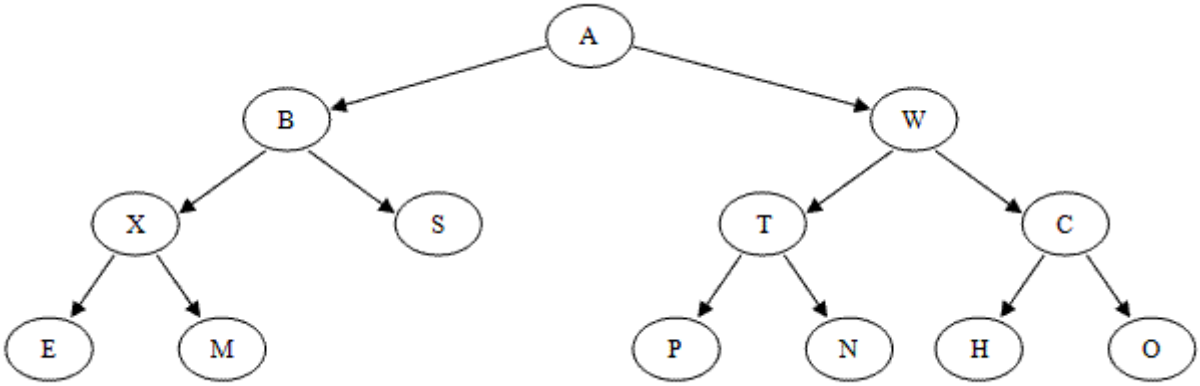
Binary Tree Overview

Formal Definition of a Binary T ree

A *binary tree* consists of a finite set of nodes that is either empty, or consists of one specially designated node called the *root* of the binary tree, and the elements of two disjoint binary trees called the *left subtree* and *right subtree* of the root.

Note that the definition above is recursive: we have defined a binary tree in terms of binary trees. This is appropriate since recursion is an innate characteristic of tree structures.

Diagram 1: A binary tree



Binary Tree Terminology

Tree terminology is generally derived from the terminology of family trees (specifically, the type of family tree called a *lineal chart*).

- Each root is said to be the *parent* of the roots of its subtrees.
- Two nodes with the same parent are said to be *siblings*; they are the *children* of their parent.
- The root node has no parent.
- A great deal of tree processing takes advantage of the relationship between a parent and its children, and we commonly say a *directed edge* (or simply an *edge*) extends from a parent to its children. Thus edges connect a root with the roots of each subtree. An *undirected edge* extends in both directions between a parent and a child.
- *Grandparent* and *grandchild* relations can be defined in a similar manner; we could also extend this terminology further if we wished (designating nodes as cousins, as an uncle or aunt, etc.).

Other Tree Terms

- The number of subtrees of a node is called the *degree* of the node. In a binary tree, all nodes have degree 0, 1, or 2.
- A node of degree zero is called a *terminal node* or *leaf node*.
- A non-leaf node is often called a *branch node*.
- The *degree of a tree* is the maximum degree of a node in the tree. A binary tree is degree 2.
- A *directed path* from node n_1 to n_k is defined as a sequence of nodes n_1, n_2, ..., n_k such that n_i is the parent of n_{i+1} for $1 <= i < k$. An *undirected path* is a similar sequence of undirected edges. The length of this path is the number of edges on the path, namely $k - 1$ (i.e., the number of nodes – 1). There is a path of length zero from every node to itself. Notice that in a binary tree there is exactly one path from the root to each node.

- The *level* or *depth* of a node with respect to a tree is defined recursively: the level of the root is zero; and the level of any other node is one higher than that of its parent. Or to put it another way, the level or depth of a node n_i is the length of the unique path from the root to n_i.
- The *height* of n_i is the length of the longest path from n_i to a leaf. Thus all leaves in the tree are at height 0.
- The *height of a tree* is equal to the height of the root. The *depth of a tree* is equal to the level or depth of the deepest leaf; this is always equal to the height of the tree.
- If there is a directed path from n_1 to n_2, then n_1 is an ancestor of n_2 and n_2 is a descendant of n_1.

Special Forms of Binary T rees

There are a few special forms of binary tree worth mentioning.

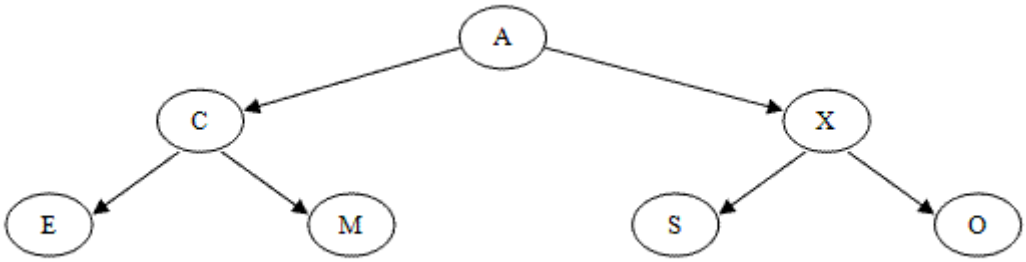
If every non-leaf node in a binary tree has nonempty left and right subtrees, the tree is termed a *strictly binary tree*. Or, to put it another way, all of the nodes in a strictly binary tree are of degree zero or two, never degree one. A strictly binary tree with N leaves always contains $2N - 1$ nodes.

Some texts call this a "full" binary tree.

A *complete binary tree* of depth d is the strictly binary tree all of whose leaves are at level d.

The total number of nodes in a complete binary tree of depth d equals $2^{d+1} - 1$. Since all leaves in such a tree are at level d, the tree contains 2^d leaves and, therefore, 2^d - 1 internal nodes.

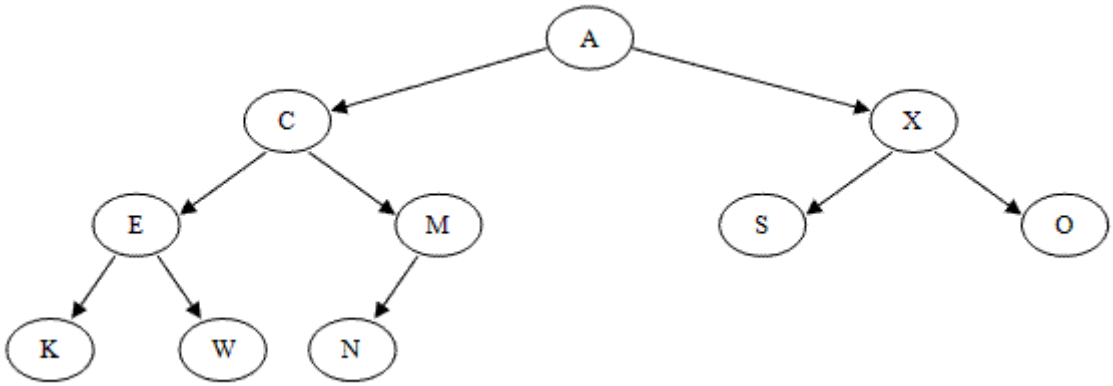
Diagram 2: A complete binary tree



A binary tree of depth d is an *almost complete binary tree* if:

- Each leaf in the tree is either at level d or at level $d - 1$.
- For any node n_d in the tree with a right descendant at level d, all the left descendants of n_d that are leaves are also at level d.

Diagram 3: An almost complete binary tree



An almost complete strictly binary tree with N leaves has $2N - 1$ nodes (as does any other strictly binary tree). An almost complete binary tree with N leaves that is not strictly binary has $2N$ nodes. There are two distinct almost complete binary trees with N leaves, one of which is strictly binary and one of which is not.

There is only a single almost complete binary tree with N nodes. This tree is strictly binary if and only if N is odd.

Representing Binary Trees in Memory

Array Representation

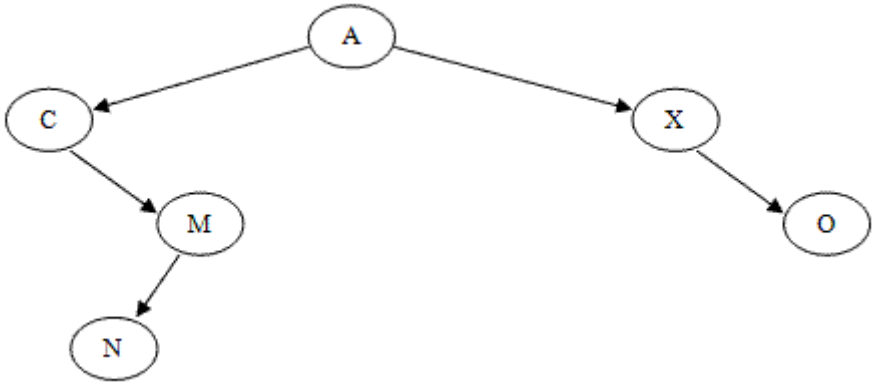
For a complete or almost complete binary tree, storing the binary tree as an array may be a good choice.

One way to do this is to store the root of the tree in the first element of the array. Then, for each node in the tree that is stored at subscript k, the node's left child can be stored at subscript $2k+1$ and the right child can be stored at subscript $2k+2$. For example, the almost complete binary tree shown in **Diagram 2** can be stored in an array like so:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
A	C	X	E	M	S	O	K	W	N

However, if this scheme is used to store a binary tree that is not complete or almost complete, we can end up with a great deal of wasted space in the array.

For example, the following binary tree

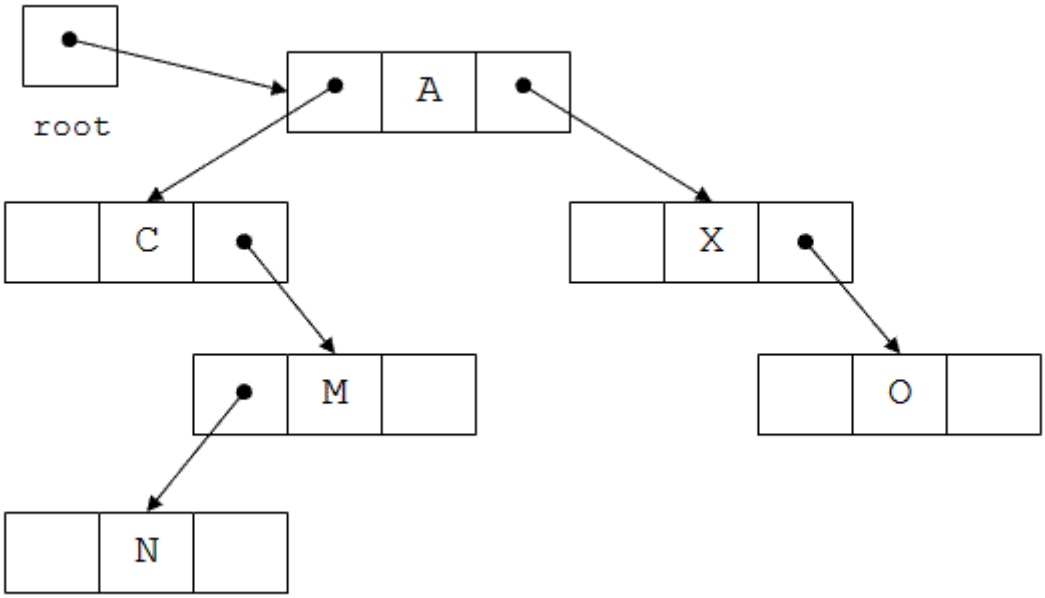


would be stored using this techinque like so:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
A	C	X		M		O			N

Linked Representation

If a binary tree is not complete or almost complete, a better choice for storing it is to use a linked representation similar to the linked list structures covered earlier in the semester:



Each tree node has two pointers (usually named `left` and `right`). The tree class has a pointer to the root node of the tree (labeled `root` in the diagram above).

Any pointer in the tree structure that does not point to a node will normally contain the value `NULL`. A linked tree with *N* nodes will always contain *N* + 1 null links.