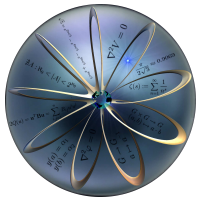


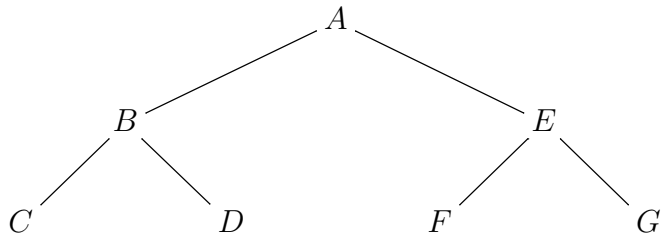
planetmath.org

Math for the people, by the people.

binary tree

Canonical name	BinaryTree
Date of creation	2013-03-22 12:29:08
Last modified on	2013-03-22 12:29:08
Owner	Daume (40)
Last modified by	Daume (40)
Numerical id	7
Author	Daume (40)
Entry type	Data Structure
Classification	msc 05C05
Classification	msc 68P05
Related topic	Tree
Related topic	BalancedTree
Related topic	Heap
Related topic	ExtendedBinaryTree
Related topic	LowerBoundForSorting
Related topic	GraphTheory
Defines	balanced binary tree
Defines	left child
Defines	right child
Defines	left descendent
Defines	right descendent
Defines	left subtree
Defines	right subtree

A *binary tree* is an ordered rooted tree where every node has two or fewer children. A *balanced binary tree* is a binary tree that is also a balanced tree. For example,



is a balanced binary tree.

The two (potential) children of a node in a binary tree are often called the *left* and *right children* of that node. The left child of some node X and all that child's descendents are the *left descendents* of X. A similar definition applies to X's *right descendents*. The *left subtree* of X is X's left descendents, and the *right subtree* of X is its right descendents.

Since we know the maximum number of children a binary tree node can have, we can make some statements regarding minimum and maximum depth of a binary tree as it relates to the total number of nodes. The maximum depth of a binary tree of n nodes is $n-1$ (every non-leaf node has exactly one child). The minimum depth of a binary tree of n nodes ($n > 0$) is $\lceil \log_2 n \rceil$ (every non-leaf node has exactly two children, that is, the tree is balanced).

A binary tree can be implicitly stored as an array, if we designate a constant, maximum depth for the tree. We begin by storing the root node at index 0 in the array. We then store its left child at index 1 and its right child at index 2. The children of the node at index 1 are stored at indices 3 and 4, and the chldren of the node at index 2 are stored at indices 5 and 6. This can be generalized as: if a node is stored at index k, then its left child is located at index $2k + 1$ and its right child at $2k + 2$. This form of implicit storage thus eliminates all overhead of the tree structure, but is only really advantageous for trees that tend to be balanced. For example, here is the implicit array representation of the tree shown above.

A	B	E	C	D	F	G

Many data structures are binary trees. For instance, heaps and binary search trees are binary trees with particular properties.