**Trees: Introduction**

In computer science, a *tree* is a non-linear data structure in which each item can have many successors and all items, except the *root*, have exactly one predecessor.



Some real-world examples of computer-science style tree structures include oak trees, a sentence parse tree, and the organization of directories and files on your computer’s hard drive. (But not a family tree, because each child has two predecessors.) Notice that computer science trees are drawn with the root at the top.

Sentence

Noun phrase Verb phrase

Article Noun Verb Noun phrase Prepositional Phrase

*The girl hit* Article Noun Preposition Noun Phrase

*the ball with* Article Noun

*a bat*

**Definitions**

|  |  |
| --- | --- |
| Node, TreeNode | stores one item of data, a link (pointer) to its left child, and a link (pointer) its right child. |
| Root | The node at the top of the tree; it is the only node without a parent. |
| Parent | The predecessor of a node; a node in a tree can have only one parent. |
| Child | The successor of a node. |
| Left child | In a binary tree, the node linked to the left of the parent. |
| Right child | In a binary tree, the node linked to the right of the parent. |
| Siblings | In a binary tree, the children of a common parent. |
| Only child | In a binary tree, a child without a sibling. |
| Leaf | A node that has no children. |
| Edge | The link that connects a parent to its child. |
| Path | The sequence of edges connecting nodes. |
| Path length | The number of edges in a path. |
| Depth (or level) of a node | Counting the levels of the rows of nodes from the root to that node. The root starts counting as level 0. The root’s children are at level 1, etc. |
| Height of a tree | The length of the longest path from the root to the bottom-most node.  The height of a tree happens to equal the depth of the bottom-most node. |
| Width of a tree, longest path | The length of the path connecting the two bottom-most leaves in the tree. Width is the maximum of the sum of the heights to the left and to the right. Often goes through root, but not always. |
| Subtree | A node which is viewed as the root of a new, smaller tree |

**Binary Tree Definitions**

|  |  |
| --- | --- |
| Binary tree | A tree in which each node has 0, 1, or (at most) 2 children |
| Balanced binary tree | A binary tree is “balanced” when the height of the left subtree and the right subtree differ at most by 1, calculated at every node.  A completely unbalanced binary tree is linear, so that every child is to the left (or to the right) of its parent. |
| Full binary tree | A binary tree in which every leaf has the same depth and every nonleaf has two children. A full binary tree contains the maximum number of nodes for its height. |
| Complete binary tree | A binary tree in which every level except the deepest contains as many nodes as possible, and at the deepest level, all the nodes are as far left as possible. You add new nodes from the left side going across. |
| Expression Tree  (BXT) | A binary tree in which the leaves are numbers and the nonleaves are arithmetic operators (+, - ,\*, /) |
| Binary Search Tree (BST) | A binary tree with the property that all the left children of each node are less than the node and all the right children are greater than or equal to it. |
| Min-Heap | A binary tree with 1) the property that each node is less than or equal to its children and 2) is complete. Priority queues use min-heaps. |
| Max-Heap | A binary tree with 1) the property that each node is greater than or equal to its children and 2) is complete. HeapSort uses a max-heap. |

BST, balanced, linear (unbalanced), a binary tree, not a BST, heaps must be complete.  
not complete BST, not complete balanced, complete This one is a min-heap. It   
 is also full and balanced.

**Tree Traversals**

|  |  |  |
| --- | --- | --- |
| Preorder traverse | Process the root, traverse the left subtree, traverse the right subtree. | root-L-R |
| Inorder traverse | Traverse the left subtree, process the root, traverse the right subtree. | L-root-R |
| Postorder traverse | Traverse the left subtree, traverse the right subtree, process the root. | L-R-root |
| Level-order traverse | From left to right, across the tree. **DBFACE** |  |

Preorder traversal: **DBACFE** Inorder traversal: **ABCDEF** Postorder traversal: **ACBEFD**