**Trees**

In computer science, a *tree* is a widely used data structure that emulates a tree structure with a set of linked nodes. Trees are used popularly in computer programming. They can be used for improving database search times (binary search trees, AVL trees, red– black trees), in game programming (minmax trees, decision trees, path fi nding trees), 3D graphics programming (binary trees, quadtrees, octrees), arithmetic scripting languages (arithmetic precedence trees), data compression (Huffman trees), and even file systems (btrees, sparse indexed trees, trie trees). Let us learn about trees in this chapter.

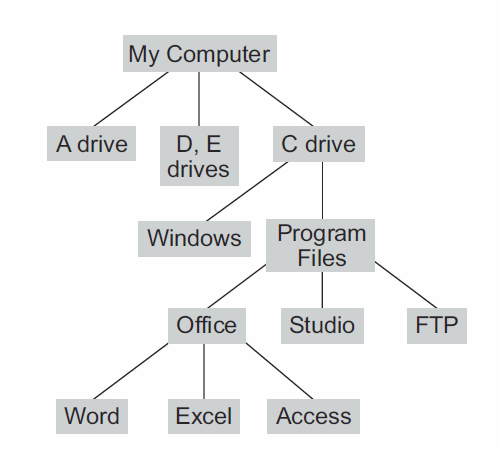
A data structure is said to be *linear* if its elements form a sequence or a linear list. In a linear data structure, every data element has a unique successor and a unique predecessor.

In non-linear data structures, every data element may have more than one predecessor as well as successor. Elements do not form any particular linear sequence.

Well-known examples of such structures are family trees, hierarchy of positions in organization, and so on. *Tree*, a non-linear data structure, is a means to maintain and manipulate data in many applications. Consider the following example:

The operating system of a computer system organizes files into directories and subdirectories.

Directories are also referred to as *folders*. The operating system organizes folders and files using a tree structure as in



The relationships in a tree are ***hierarchical***, with some objects being “above” and some “below” others. Actually, the main terminology for tree data structures comes from family trees, with the terms “parent,” “child,” “ancestor,” and “descendant” being the most common words used to describe relationships.

**Tree Definitions and Properties**

A ***tree*** is an abstract data type that stores elements hierarchically. With the exception of the top element, each element in a tree has a ***parent*** element and zero or more ***children*** elements. We typically call the top element the ***root*** of the tree, but it is drawn as the highest element, with the other elements being connected below (just the opposite of a botanical tree).

**Formal Tree Definition**

Formally, we define ***tree*** *T* to be a set of ***nodes*** storing elements in a ***parent-child*** relationship with the following properties:

• If *T* is nonempty, it has a special node, called the ***root*** of *T*, that has no parent.

• Each node *v* of *T* different from the root has a unique ***parent*** node *w*; every node with parent *w* is a ***child*** of *w*.

Note that according to our definition, a tree can be empty, meaning that it doesn’t have any nodes. This convention also allows us to define a tree recursively, such that a tree *T* is either empty or consists of a node *r*, called the root of *T*, and a (possibly empty) set of trees whose roots are the children of *r*.

Two nodes that are children of the same parent are ***siblings***.

A node *v* is ***external*** if *v* has no children.

A node *v* is ***internal*** if it has one or more children.

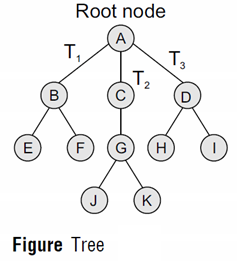
External nodes are also known as ***leaves***.

An ***edge*** of tree *T* is a pair of nodes (*u*,*v*) such that *u* is the parent of *v*, or vice

versa.

A ***path*** of *T* is a sequence of nodes such that any two consecutive nodes in

the sequence form an edge.



***Ancestor node*** An ancestor of a node is any predecessor node on the path from root to that node. The root node does not have any ancestors. In the tree given in Fig. 9.1, nodes A, C, and G are the ancestors of node K.

***Descendant node*** A descendant node is any successor node on any path from the node to a leaf node. Leaf nodes do not have any descendants. In the tree given in Figure above, nodes C, G, J, and K are the descendants of node A.

***Level number*** Every node in the tree is assigned a *level number* in such a way that the root node is at level 0, children of the root node are at level number 1. Thus, every node is at one level higher than its parent. So, all child nodes have a level number given by parent’s level number + 1.

***Degree*** Degree of a node is equal to the number of children that a node has. The degree of a leaf node is zero.

***In-degree*** In-degree of a node is the number of edges arriving at that node.

***Out-degree*** Out-degree of a node is the number of edges leaving that node.

**Binary Trees**

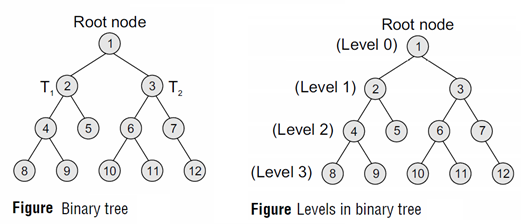
A binary tree is a data structure that is defined as a collection of elements called nodes. In a binary tree, the topmost element is called the root node, and each node has 0, 1, or at the most 2 children.

**A binary tree**

1. is either an empty tree or

2. consists of a node, called *root*, and two children, *left* and *right*, each of which is itself a binary tree.

The definition is recursive as we have defined a binary tree in terms of itself. All the internal nodes of a binary tree are themselves the roots of smaller binary trees



A node that has zero children is called a leaf node or a terminal node. Every node contains a data element, a left pointer which points to the left child, and a right pointer which points to the right child. The root element is pointed by a 'root' pointer. If root = NULL, then it means the tree is empty.

Figure Binary Tree shows a binary tree. In the figure, R (value of node is 1) is the root node and the two trees T1 and T2 are called the left and right sub-trees of R. T1 is said to be the left successor of R. Likewise, T2 is called

the right successor of R.

Note that the left sub-tree of the root node consists of the nodes: 2, 4, 5, 8, and 9. Similarly, the right sub-tree of the root node consists of nodes: 3, 6, 7, 10, 11, and 12.

In the tree, root node 1 has two successors: 2 and 3. Node 2 has two successor nodes: 4 and 5.

Node 4 has two successors: 8 and 9. Node 5 has no successor. Node 3 has two successor nodes:

6 and 7. Node 6 has two successors: 10 and 11. Finally, node 7 has only one successor: 12.

A binary tree is recursive by definition as every node in the tree contains a left sub-tree and a right sub-tree. Even the terminal nodes contain an empty left sub-tree and an empty right sub-tree. In the same figure , nodes 5, 8, 9, 10, 11, and 12 have no successors and thus said to have empty sub-trees.

***Terminology***

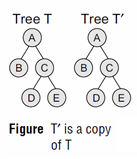
***Parent*** If N is any node in T that has *left successor* S1 and *right successor* S2, then N is called the *parent* of S1 and S2. Correspondingly, S1 and S2 are called the left child and the right child of N. Every node other than the root node has a parent.

***Level number*** Every node in the binary tree is assigned a *level number* (refer Fig. Levels). The root node is defined to be at level 0. The left and the right child of the root node have a level number 1. Similarly, every node is at one level higher than its parents. So all child nodes are defined to have level number as parent's level number + 1.

***Degree of a node*** It is equal to the number of children that a node has. The degree of a leaf node is zero. For example, in the tree, degree of node 4 is 2, degree of node 5 is zero and degree of node 7 is 1.

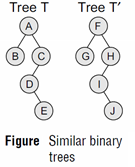
***Sibling*** All nodes that are at the same level and share the same parent are called *siblings* (brothers). For example, nodes 2 and 3; nodes 4 and 5; nodes 6 and 7; nodes 8 and 9; and nodes 10 and 11 are siblings.

***Leaf node*** A node that has no children is called a leaf node or a terminal node. The leaf nodes in the tree are: 8, 9, 5, 10, 11, and 12.

******

***Copies*** Two binary trees T and T’ are said to be *copies* if they have similar structure and if they have same content at the corresponding nodes. Figure below shows that T’ is a copy of T.

***Similar binary trees*** Two binary trees T and T’ are said to be similar if both these trees have the same structure. Figure below: shows two *similar binary trees*.



***Edge*** It is the line connecting a node N to any of its successors. A binary tree of n nodes has exactly n – 1 edges because every node except the root node is connected to its parent via an edge.

***Path*** A sequence of consecutive edges*.* For example, in Fig. , the path from the root node to the node 8 is given as: 1, 2, 4, and 8.

***Depth*** The *depth* of a node N is given as the length of the path from the root R to the node N. The depth of the root node is zero.

***Height of a tree*** It is the total number of nodes on the path from the root node to the deepest node in the tree. A tree with only a root node has a height of 1. A binary tree of height h has at least h nodes and at most 2h – 1 nodes. This is because every level will have at least one node and can have at most 2 nodes. So, if every level has two nodes then a tree with height h will have at the most 2h – 1 nodes as at level 0, there is only one element called the root. The height of a binary tree with n nodes is at least log2(n+1) and at most n.

***In-degree/out-degree of a node*** It is the number of edges arriving at a node. The root node is the only node that has an in-degree equal to zero. Similarly, *out-degree* of a node is the number of edges leaving that node.

Binary trees are commonly used to implement binary search trees, expression trees, tournament trees, and binary heaps.

A binary tree is ***proper*** if each node has either zero or two children. Some people also refer to such trees as being ***full*** binary trees.

Thus, in a proper binary tree, every internal node has exactly two children. A binary tree that is not proper is ***improper***.

***Representation of Binary Trees in the Memory***

See PowerPoint Slides

***Linked representation of binary***

See PowerPoint Slides