Introduction to Trees

# You have already studied the data structure called a *singly linked list*. We present a diagram of a linked list to the right. This particular linked list has a header, with d pointing to it.

# Another way to depict this linked list in minimal fashion, without a header, is shown to the right, with the values in the circles. It is an implementation of the list (6, 7*,* 3).

# But if each node can have 0 or more successors, as shown to the right, we have what we call a *tree*. Here the first node, with value 6, has three successors, with values 2, 7, and 8. The first node’s successor with value 2 has no successors. The first node’s successor with value 7 has one successor, and the third node’s successor has 2 successors.

# Thus, a linked list is a special form of a tree.

# To the right, we draw a tree as computer scientists generally draw trees. No arrow heads are used; it is assumed that all lines point down. And, think of the letters not as values but just as names of the nodes. Node A at the top is called the *root* of the tree, and nodes B, H, I, and J, which have no successors, are called *leaves*. Computer scientists generally draw trees upside down!

At this point, it’s important to realize that the last tree we drew is not a *data structure*. Remember: a *data structure* is an organization or format for storing and managing data, usually to make some operations efficient. Here, we’re not showing how to implement trees, or how to store them in a data structure. We’re just introducing the type *tree*.

**Uses of trees**

Trees are used all over the place. Several sorting algorithms use trees, and they are used in implementing a *priority queue*.

Parse trees and syntax tree are used in compilers and interpreters.

A GUI (graphical user interface) on your screen is basically a tree: there is one item, and it contains others, each of which may contain more.

A program that plays a game like chess or checkers may create a game tree of possible successive moves by the two players and prune it to find an optimal move.

The file system on your hard drive is structured as a tree: The root directory contain files (leaves) and other directories, each of which can contain ….

The management structure at a company is generally a tree.

On the next page, we introduce terminology used in discussing trees.

**Tree terminology**

The tree below describes the notion of a *parent* of a node, a *child* of a node, and a *sibling* of a node. Based on that, the notions of *ancestor* and *descendent* are obvious.



A is **parent** of B, C, D

I and H are **siblings**

J is **child** of D

**ancestor**: parent, parent’s parent, parent’s parent, etc.

**descendent** or **descendant**: child, child’s child, child’s child’s child, etc.

J’s **ancestors** are D and A.

C’s **descendents** are F, H, and I

With those definitions, we define the root of a tree, the leaves of a tree, and the internal nodes of a tree:

**internal node**: has a child

internal nodes are A, B, C, D, F



**root** of tree (no parent)

**leaf** of tree (no child)

A path in a tree: A

***Edge***

The connection between one node and another.

***Path***

A sequence of nodes and edges connecting a node with a descendant.

***Level***

The level of a node is defined as: 1 + the number of edges between the node and the root.

***Height of node***

The height of a node is the number of edges on the longest path between that node and a leaf.

***Height of tree***

The height of a tree is the height of its root node.

***Depth***

The depth of a node is the number of edges from the tree's root node to the node.

***Forest***

A forest is a set of n ≥ 0 disjoint trees.