11 Introduction to Trees

This chapter provides the conceptual framework for working with the hierarchical collections known as trees. We begin with an overview of the terminology used to talk about trees and examine representations of the two most common, binary trees and general trees. To explore the representation of binary trees, we then develop a prototype binary tree class. A Case Study illustrates the use of a variant of a binary tree called an expression tree for evaluating arithmetic expressions. We then show how to add a tree collection that represents general trees to the Java collection framework. Chapter 12 continues our examination of trees by looking at some special-purpose variations, such as heaps, binary search trees, and other kinds of search trees.

11.1 Overview of Trees

To this point, we have been discussing linear collections; however, many applications require collections in which there is a hierarchical relationship between items. In computer science, we call these collections trees. They are characterized by the fact that each item can have multiple successors and all items, except a privileged item called the *root,* have exactly one predecessor.

For example, consider Figure 11.1, which is the parse tree for the sentence, "The girl hit the ball with a bat." In this, and in all diagrams of trees, the items are called *nodes*. Trees are drawn with the root at the top. Immediately below a node and connected to it by lines are its successors, or ***children.*** A node without children (in italics in the figure) is called a *leaf.* Immediately above a node is its predecessor, or *parent.* Thus, the root node "Sentence" has two children but no parent. In contrast,

the leaf node "ball" has a parent but no children. A node, such as "Noun phrase," that has children is called an ***interior node.***

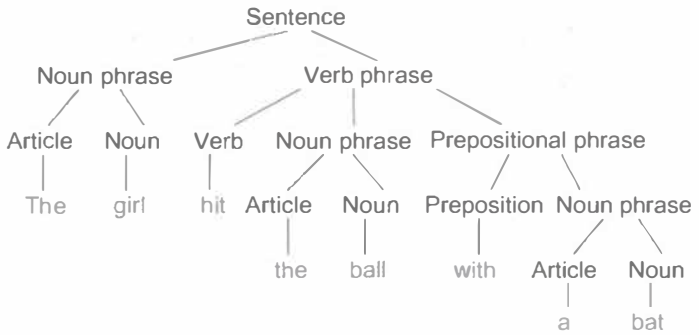


Figure 11.1 Parse tree for a sentence

A book's table of contents is another familiar example of a hierarchical collection or tree. Here, however, each node is on a single line, and indentation indicates parent-child relationships (see Figure 11.2). The ellipses (. . .) indicate omitted entries.

Figure 11.2 Partial table of contents for *The Albatross Book of Living Verse,* published by W. Collins Sons and Co. Ltd., London

In addition, trees are used to organize a computer's file system (see Figure 11.3) as we are reminded every time we view the files on our hard drive.

Example XML

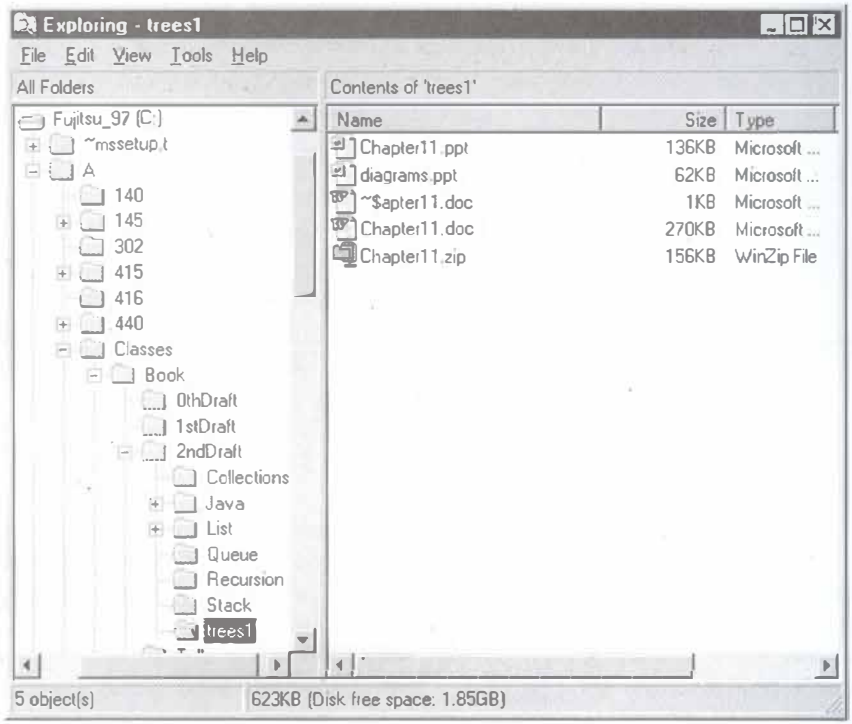


Figure 11.3 A partial directory tree of the files on the disorganized home computer of one of the authors

These examples barely scratch the surface because the applications of trees are truly multitudinous. In this and the next chapter, we encounter a few more: expression trees, search trees, and heaps.

11.1.1 Talking About Trees

We have already introduced some to the terminology associated with trees, but there is more. The terminology is a peculiar mix of biological, genealogical, and geometric terms. Table 11.1 provides a quick summary with the earlier terms repeated.

Table 11.1

A Summary of Terms Used to Describe Trees

Term	Definition
Node	An item stored in a tree.
Root	The topmost node in a tree. It is the only node without a predecessor.
Child	A successor of a node. A node can have more than one child, and its children are viewed as organized in left to right order. The leftmost child is called the first child, and the rightmost is the last child.

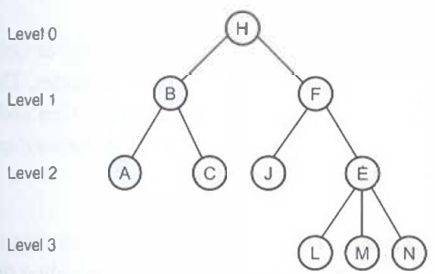
Continues

Table 11.1 *(Continued)*

A Summary of Terms Used to Describe Trees

Term	Definition
Parent	The predecessor of a node. A node can have only one parent.
Siblings	The children of a common parent.
Edge/Branch	The line that connects a parent to its child.
Descendant	A node's descendants include its children, its children's children, and so on, down to the leaves.
Ancestor	A node's ancestors include its parent, its parent's parent, etc., up to the root.
Path	The sequence of edges that connects a node and one of its descendants.
Path length	The number of edges in a path.
Leaf	A node that has no children.
Interior node	A node that has at least one child.
Depth or level	The depth or level of a node equals the length of the path connecting it to the root. Thus, the root depth or level of the root is 0. Its children are at level 1, and so on.
Height	The height of a tree equals the length of the longest path in the tree, or put differently, the maximum level number among leaves in the tree.
Subtree	The tree formed by considering a node and all its descendants. We exclude the root when forming subtrees.

Figure 11.4 shows a tree and some of its properties.



Property	Value
Number of nodes	10
Height	4
Root node	H
Leaves	A, C, J, L, M, N
Interior nodes	H, B, F, E
Nodes at level 2	A, C, J, E
Ancestors of E	F, H
Descendants of F	J, E, L, M, N
Nodes in right subtree of F	E, L, M, N

Figure 11.4 A tree and some of its properties

The trees we have been discussing are sometimes called ***general trees*** to distinguish them from a special category called ***binary trees***. In a binary tree, each node has at most two children, referred to as the ***left child*** and the ***right child***. In a binary tree, when a node has only one child, we distinguish between it being a left child and a right child. Thus, the two trees shown in Figure 11.5 are not the same when considered as binary trees, although they are the same when considered as general trees.

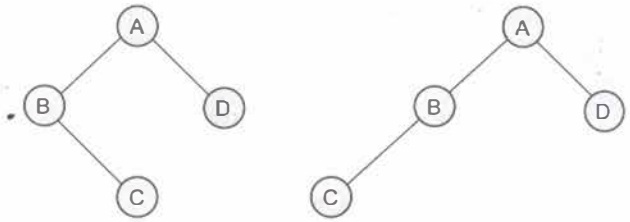


Figure 11.5 Two unequal binary trees that just happen to have equal sets of nodes

11.1.2 Formal Definitions of General and Binary Trees

Although at this point you probably have a clear understanding of what a tree is, we now give a more formal definition. As is often the case, one cannot understand the formal definition without an intuitive grasp of the concept being defined; however, the formal definition is important because it provides an unambiguous and precise basis for further discussion. For the sake of comparison, we present recursive and nonrecursive definitions. The recursive definitions are particularly important because they suggest the basic pattern for the many algorithms that process trees recursively.

Nonrecursive definition of a general tree: A ***general tree*** is either empty or consists of a finite set of ***nodes*** and a finite set of ***edges*** that connect pairs of nodes. The node at one end of an edge is called the ***parent*** and at the other the ***child.*** One node is distinguished from all others and is called the root. All nodes except the root are connected by an edge to exactly one parent.

Recursive definition of a general tree: A ***general tree*** is either empty or consists of a finite set of nodes T. One node r is distinguished from all others and is called the ***root.*** In addition, the set $T - \{r\}$ is partitioned into disjoint subsets, each of which is a general tree.

Recursive definition of a binary tree: A ***binary tree*** is either empty or consists of a root plus a ***left subtree*** and a ***right subtree,*** each of which are binary trees.

11.2 Representations of Trees

As we have grown to expect from previous chapters, there are two general approaches to representing trees, one using linked nodes and the other using arrays; however, the use of arrays is usually limited to complete binary trees (we define this term shortly).

11.2.1 Linked Representations

To begin, Figure 11.6 shows a rather obvious linked representation of a general tree. Here each node contains a data item and pointers to the node's children. The child pointers are usually stored in a list, which grows and shrinks as children are added and removed. A list also accommodates the fact that not all nodes have the same number of children. Occasionally, pointers are added to each node. These can include a pointer to a node's parent and sometimes to its left and right siblings. The extra pointers can speed certain operations on trees, but at the cost of increasing the size of each node.

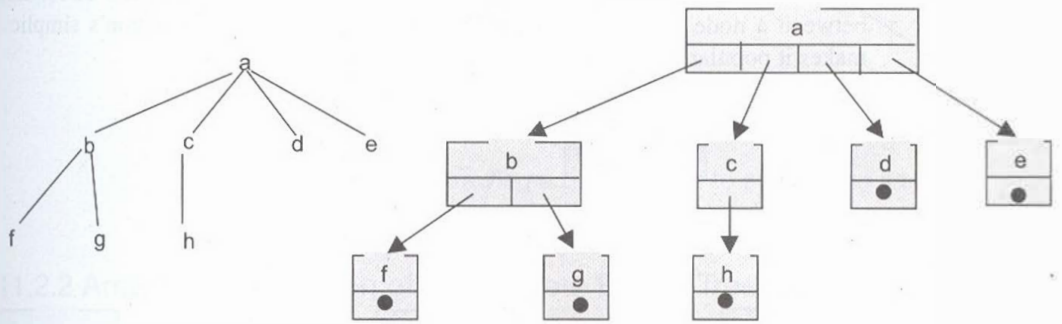


Figure 11.6 A general tree and its linked representation

The linked representation of a binary tree is similar to that of a general tree except that each node always has exactly two pointers, one to each child (see Figure 11.7). Here there is no need to store the pointers in a list.