

# Trees

## Chapter 23



# Contents

- Tree Concepts
  - Hierarchical Organizations
  - Tree Terminology
- Traversals of a Tree
  - Traversals of a Binary Tree
  - Traversals of a General Tree
- Java Interfaces for Trees
  - Interfaces for All Trees
  - An Interface for Binary Trees

# Contents

- Examples of Binary Trees
  - Expression Trees
  - Decision Trees
  - Binary Search Trees
  - Heaps
- Examples of General Trees
  - Parse Trees
  - Game Trees

# Objectives

- Describe binary trees, general trees, using standard terminology
- Traverse tree in one of four ways: preorder, postorder, inorder, level order
- Give examples of binary trees: expression trees, decision trees, binary search trees, and heaps
- Give examples of general trees: including parse trees, game trees

# Trees vs Hashtables

- <http://stackoverflow.com/questions/4128546/advantages-of-binary-search-trees-over-hash-tables>
- Hash tables in general have better cache behavior requiring less memory reads compared to a binary tree. For a hash table you normally only incur a single read before you have access to a reference holding your data ( $O(1)$ ). The binary tree, if it is a balanced variant, requires something in the order of  $O(k * \log(n))$  memory reads for some constant  $k$ .
- On the other hand, if an enemy knows your hash-function the enemy can force your hash table to make collisions, greatly hampering its performance. The workaround is to choose the hash-function randomly from a family, but a BST does not have this disadvantage. Also, when the hash table pressure grows too much, you often tend to enlarge and reallocate the hash table which may be an expensive operation. The BST has simpler behavior here and does not tend to suddenly allocate a lot of data and do a rehashing operation.
- Trees tend to be the ultimate average data structure. They can act as lists, can easily be split for parallel operation, have fast removal, insertion and lookup on the order of  $O(\lg n)$ . They do nothing *particularly* well, but they don't have any excessively bad behavior either.
- Finally, BSTs are much easier to implement in (pure) functional languages compared to hash-tables and they do not require destructive updates to be implemented (the *persistence* argument by Pascal above).

# Tree Concepts

- A way to organize data
  - Consider a family tree
- Hierarchical organization
  - Data items have ancestors, descendants
  - Data items appear at various levels
- Contrast with previous linearly organized structures



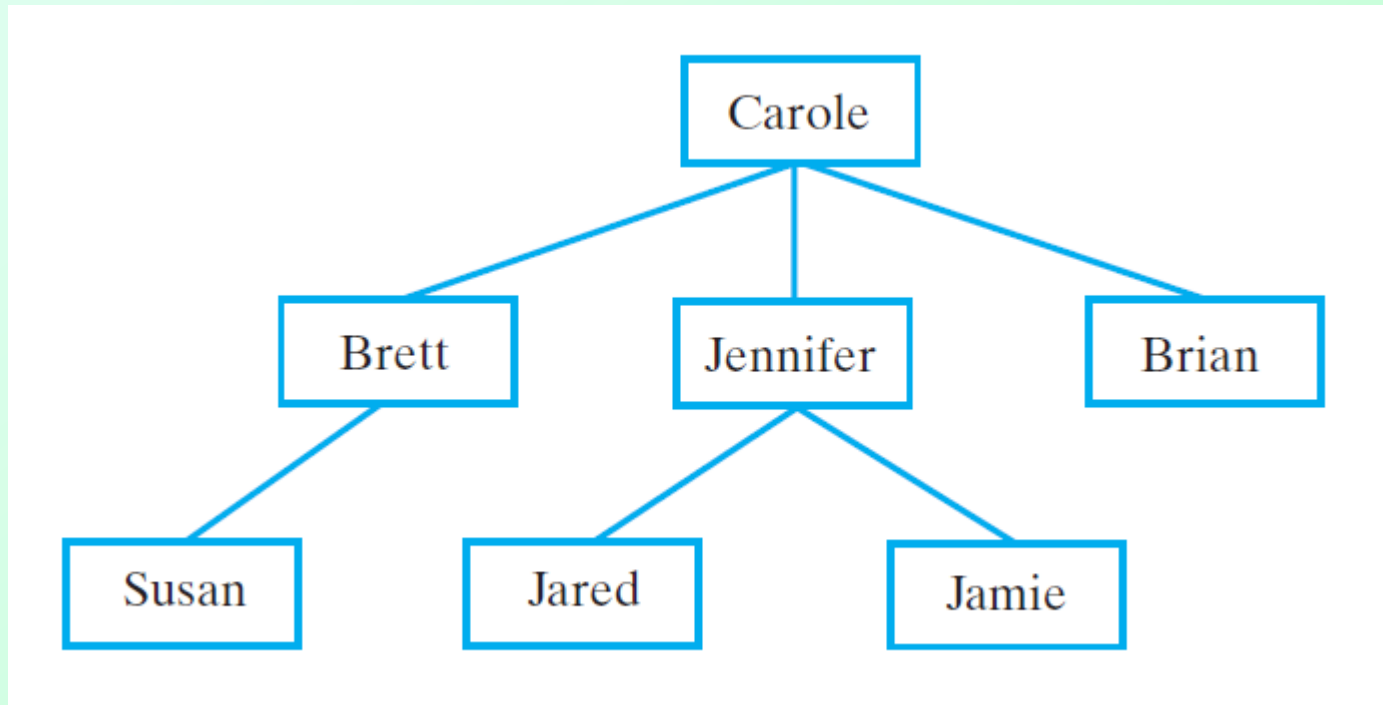


Figure 23-1 Carole's children and grandchildren

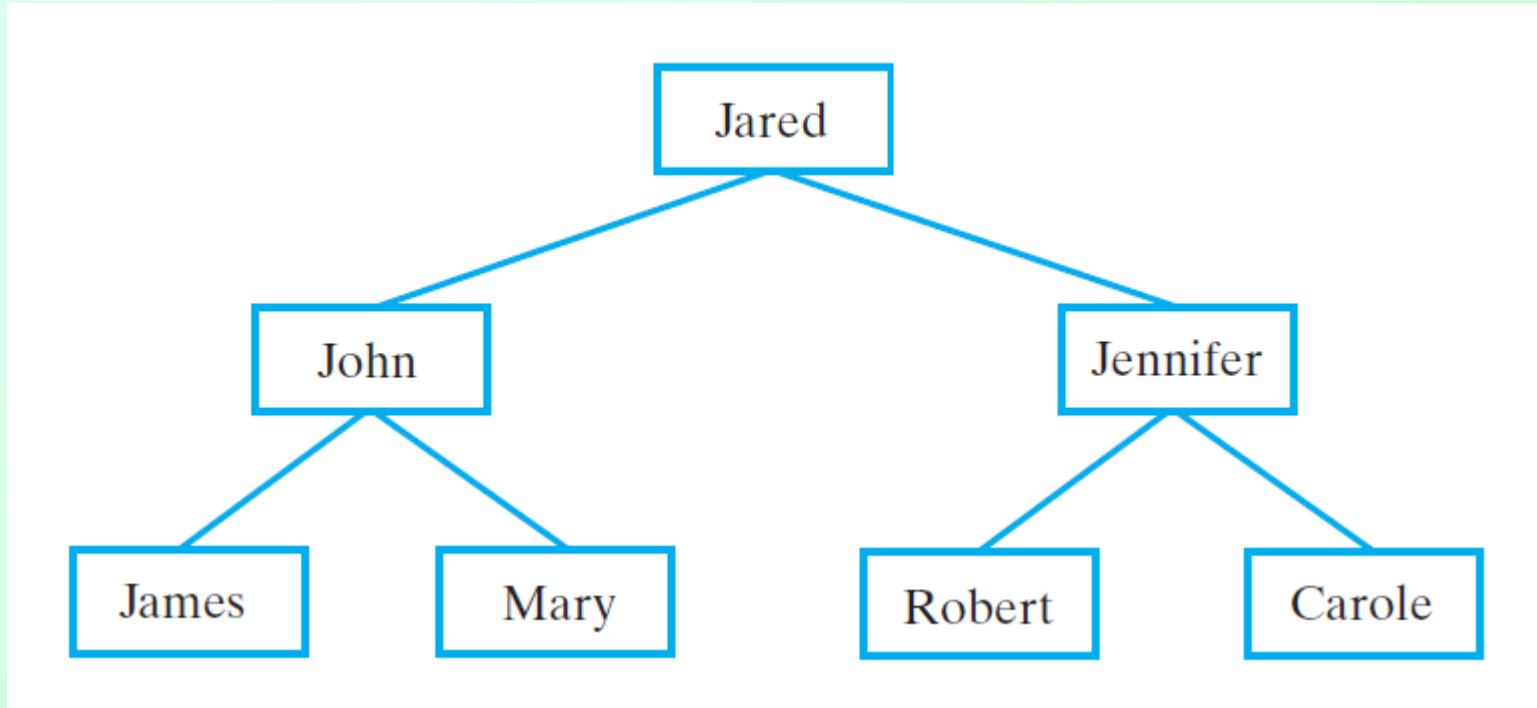


Figure 23-2 Jared's parents and grandparents



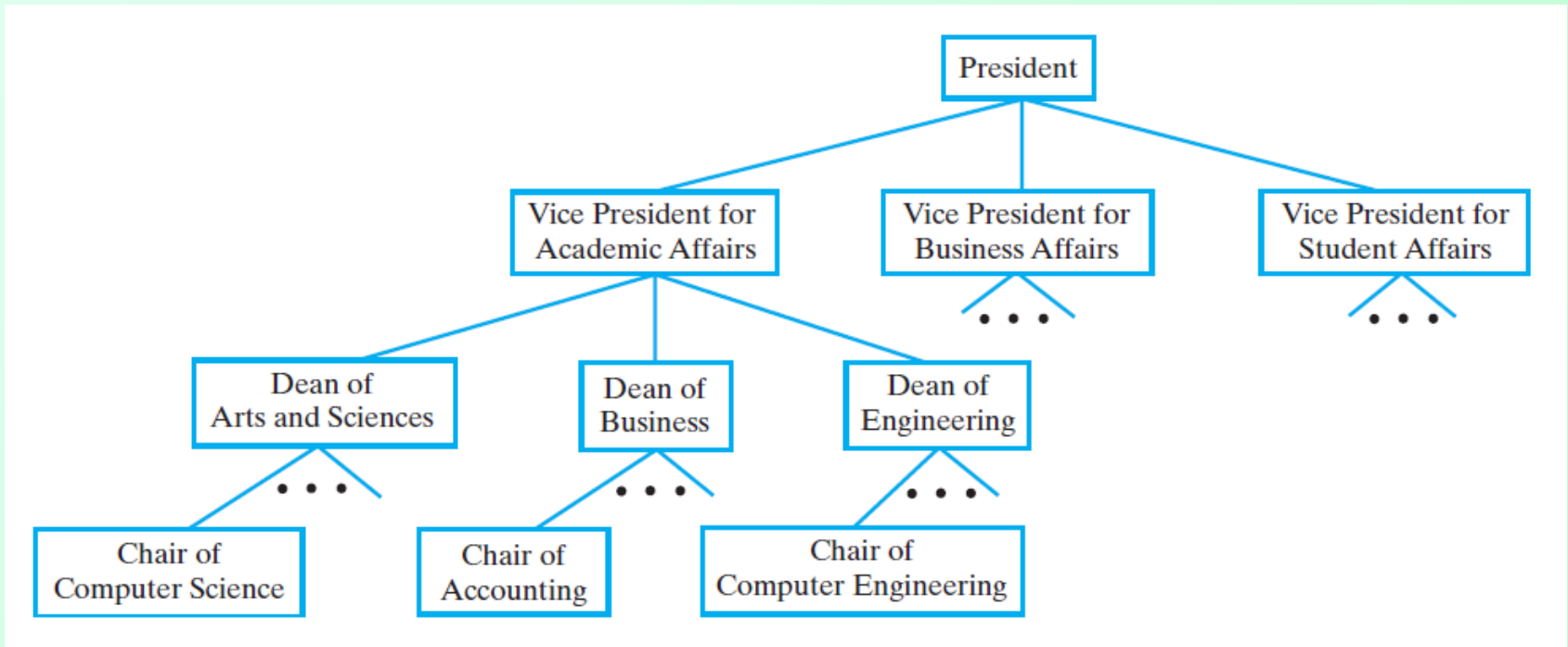


Figure 23-3 A portion of a university's administrative structure

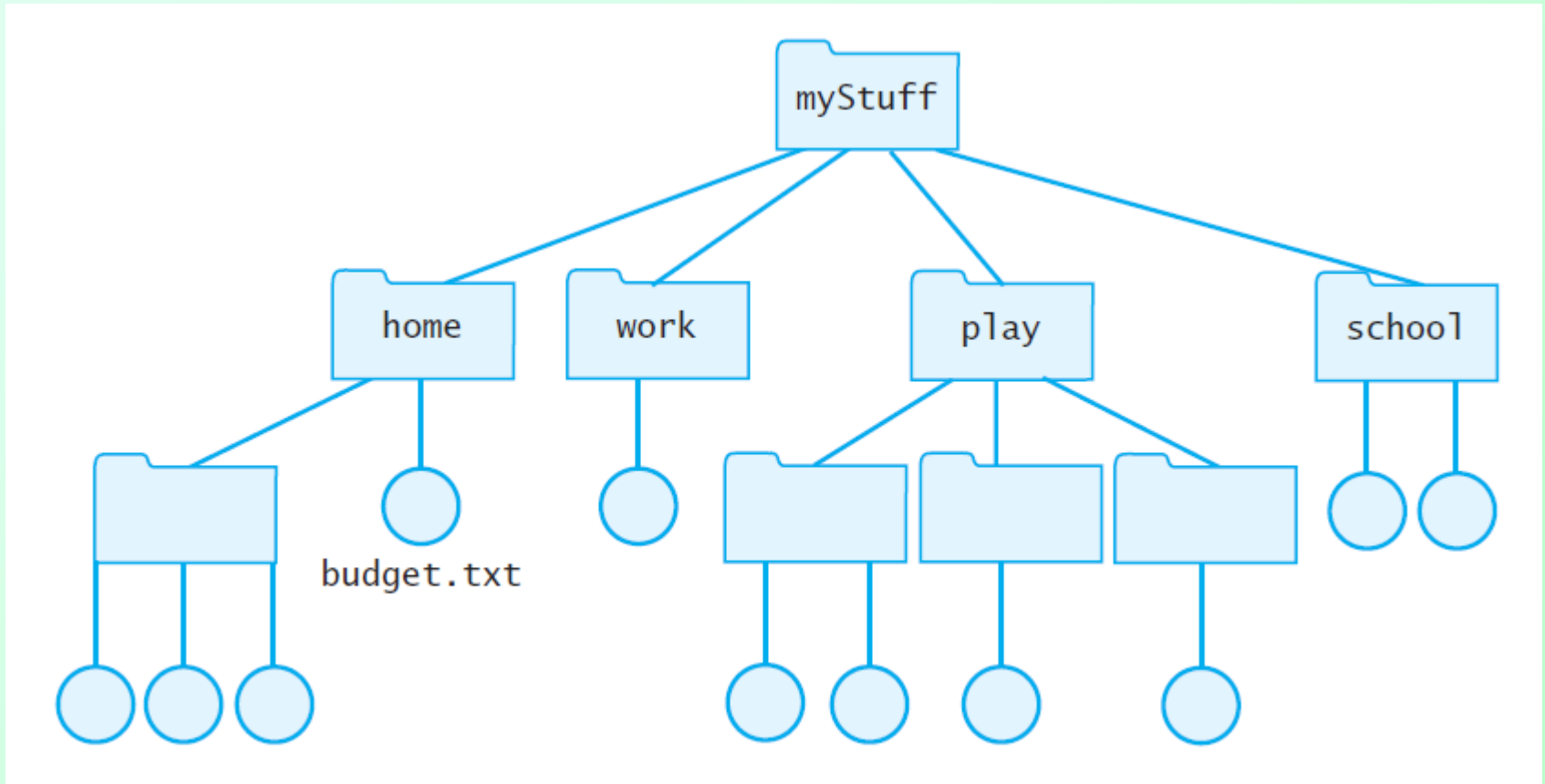


Figure 23-4 Computer files organized into folders

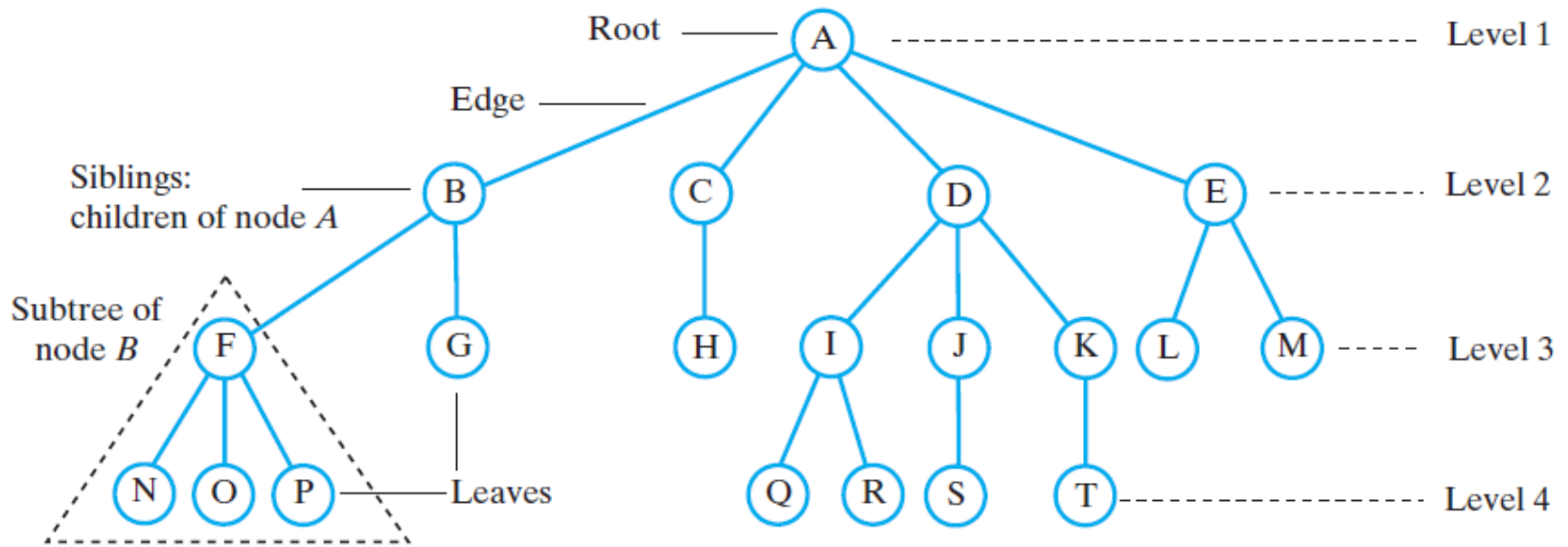


Figure 23-5 A tree equivalent to the tree in Figure 23-4

# Tree Concepts

- *Root* of an ADT tree is at tree's top
  - Only node with no parent
  - All other nodes have one parent each
- Each node can have *children*
  - A node with children is a *parent*
  - A node without children is a *leaf*

# Tree Concepts

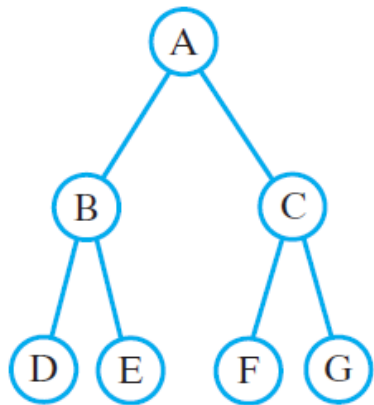
- *General tree*
  - Node can any number of children
- *N-ary tree*
  - Node has max  $n$  children
  - Binary tree node has max 2 children
- Node and its descendants form a *subtree*
- Subtree of a node
  - Tree rooted at a child of that node

# Tree Concepts

- Subtree of a tree
  - Subtree of the tree's root
- Height of a tree
  - Number of levels in the tree
- Path between a tree's root and any other node is unique.

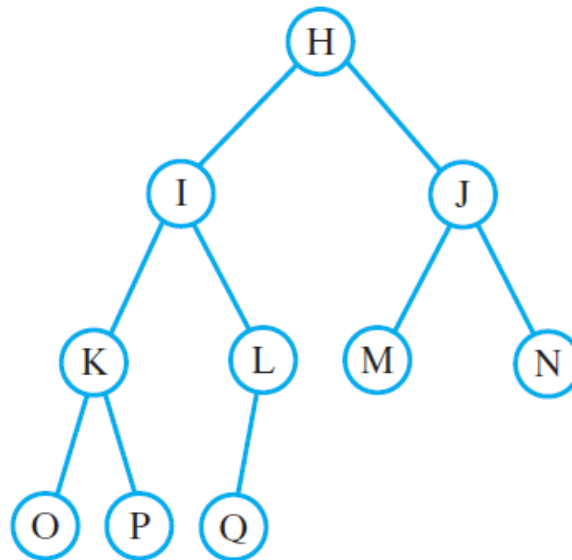


(a) Full tree



Left children: B, D, F  
Right children: C, E, G

(b) Complete tree



(c) Tree that is not full and not complete

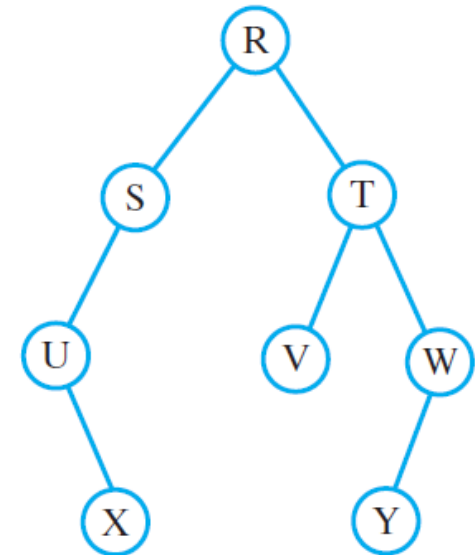


Figure 23-6 Three binary trees



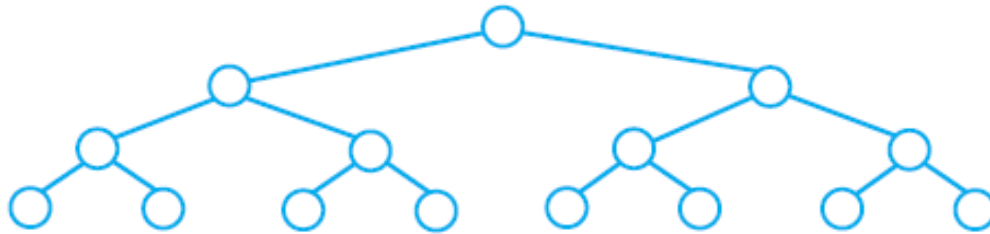
Full Tree	Height	Number of Nodes
	1	$1 = 2^1 - 1$
	2	$3 = 2^2 - 1$

Figure 23-7 The number of nodes in a full binary tree as a function of the tree's height



3

$$7 = 2^3 - 1$$



4

$$15 = 2^4 - 1$$

Figure 23-7 The number of nodes in a full binary tree as a function of the tree's height

The height of a binary tree with  $n$  nodes that is either complete or full is  $\log_2 (n + 1)$  rounded up.

Number of nodes per level

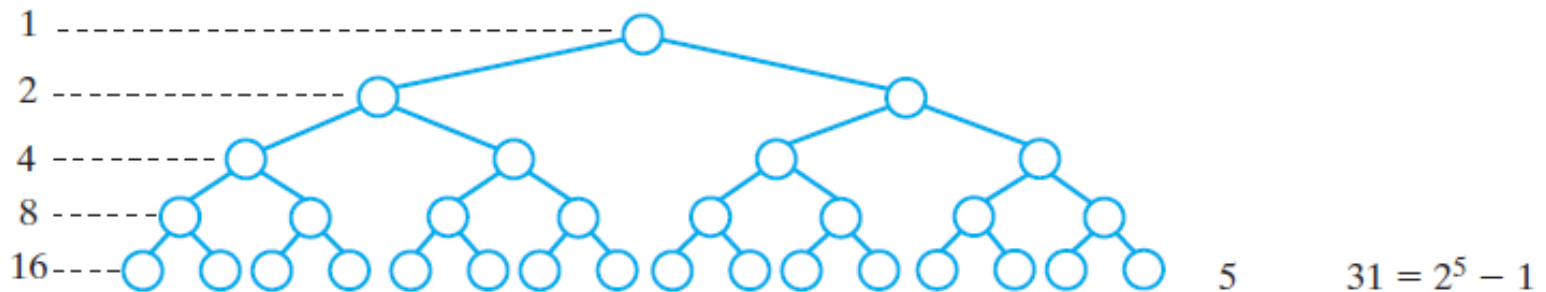


Figure 23-7 The number of nodes in a full binary tree as a function of the tree's height

# Traversals of a Tree

- Must visit/process each data item exactly once
- Nodes can be visited in different orders
- For a binary tree
  - Visit the root
  - Visit all nodes in root's left subtree
  - Visit all nodes in root's right subtree
- Could visit root before, between, or after subtrees

Visit root *before* visiting root's subtrees

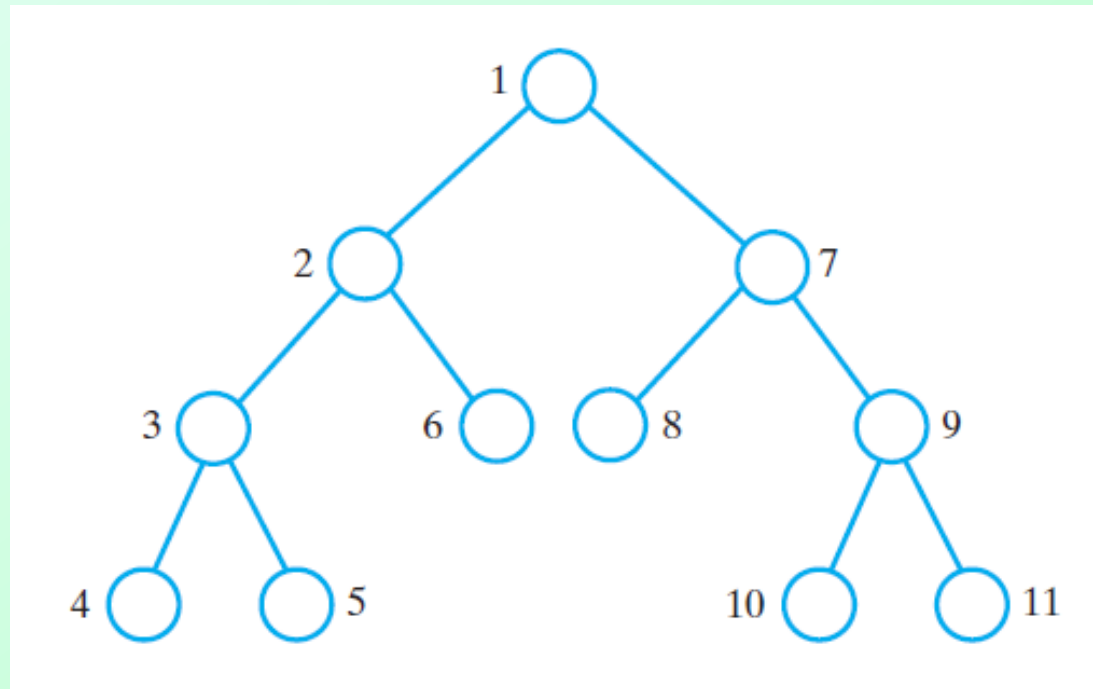


Figure 23-8 The visitation order of a **preorder** traversal



Visit root *between* visiting root's subtrees

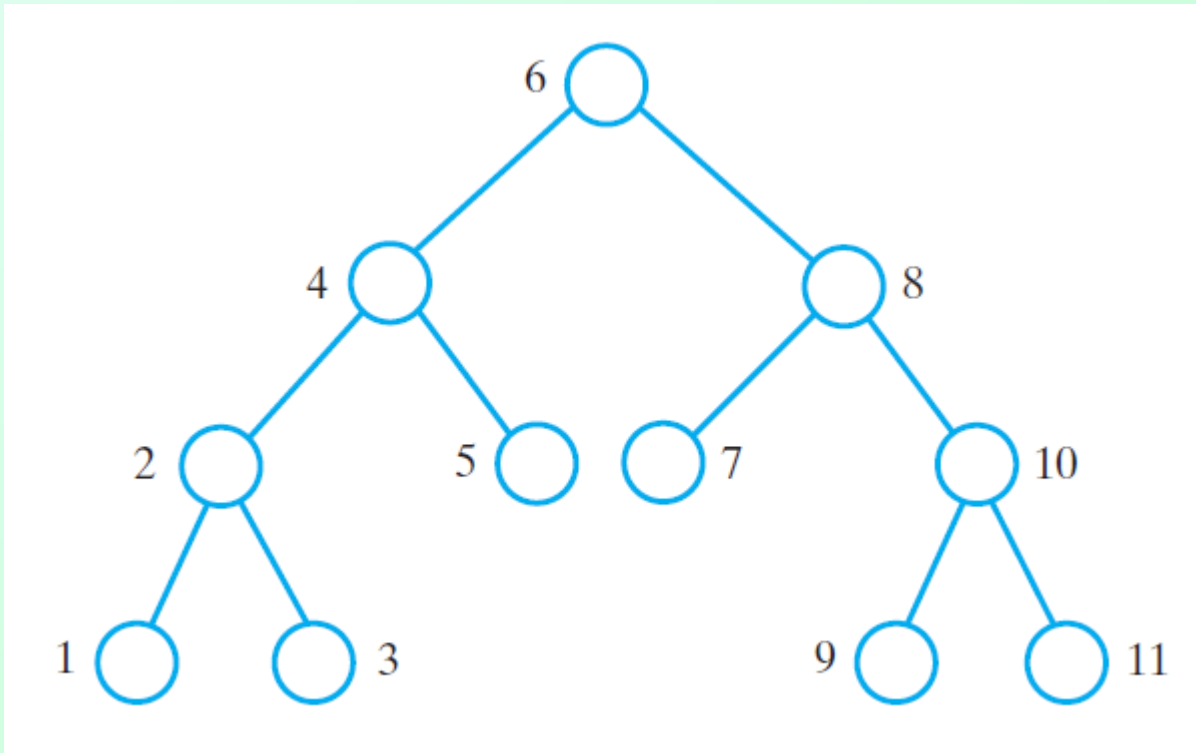


Figure 23-9 The visitation order of an **inorder** traversal

Visit root *after* visiting root's subtrees

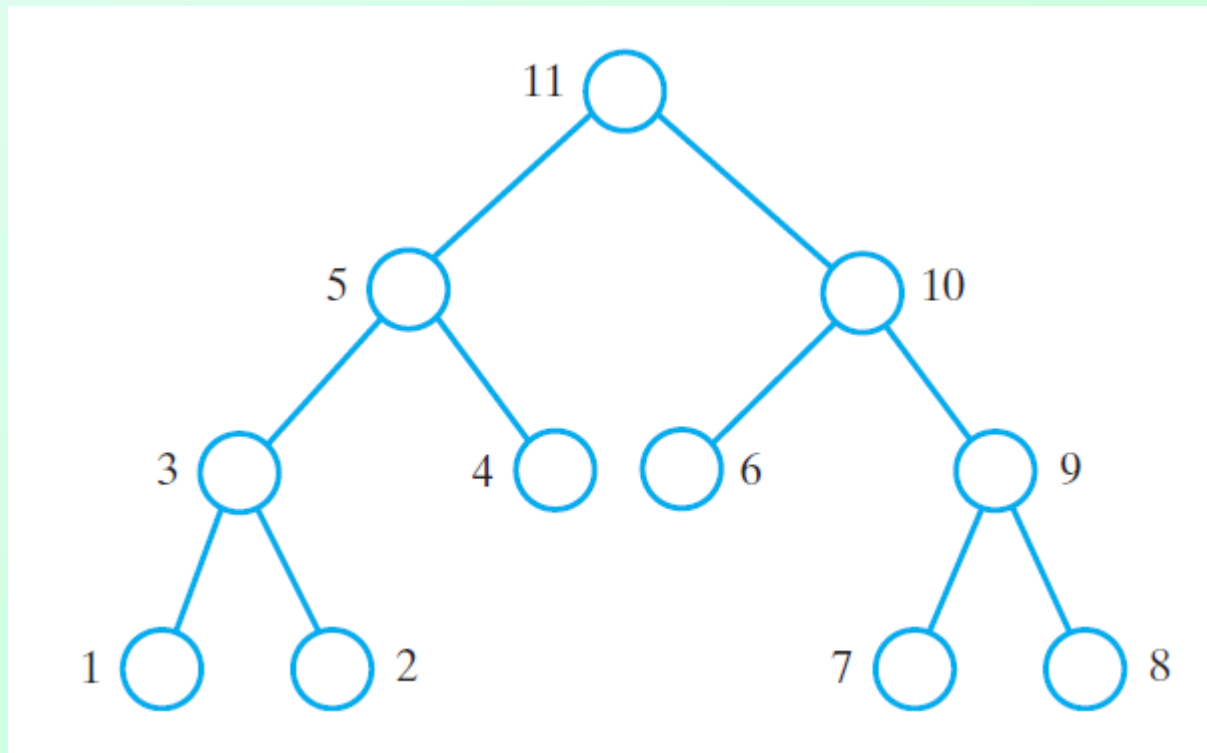


Figure 23-10 The visitation order of a **postorder** traversal

# Traversals of a Tree

- Level-order traversal
  - Example of *breadth-first* traversal
- Pre-order traversal
  - Example of *depth-first* traversal
- For a general tree (not a binary)
  - In-order traversal not well defined
  - Can do level-order, pre-order, post-order

Begins at root, visits nodes one level at a time

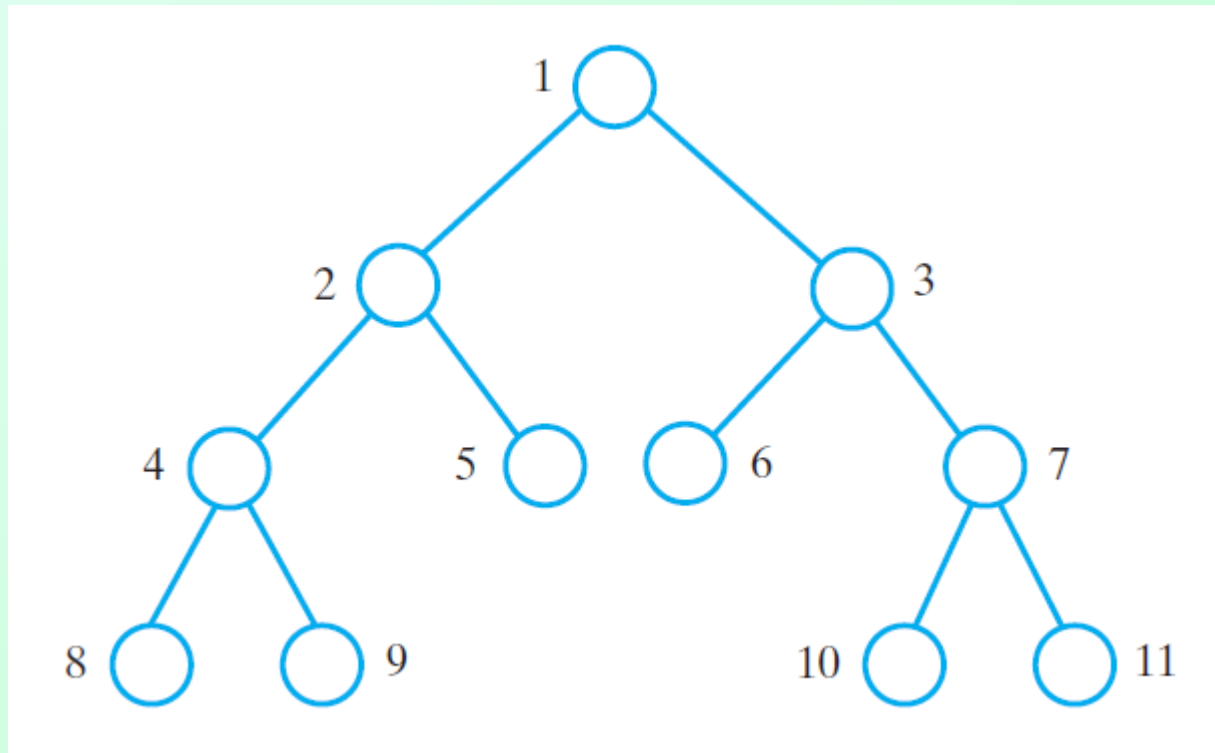


Figure 23-11 The visitation order of a **level-order** traversal

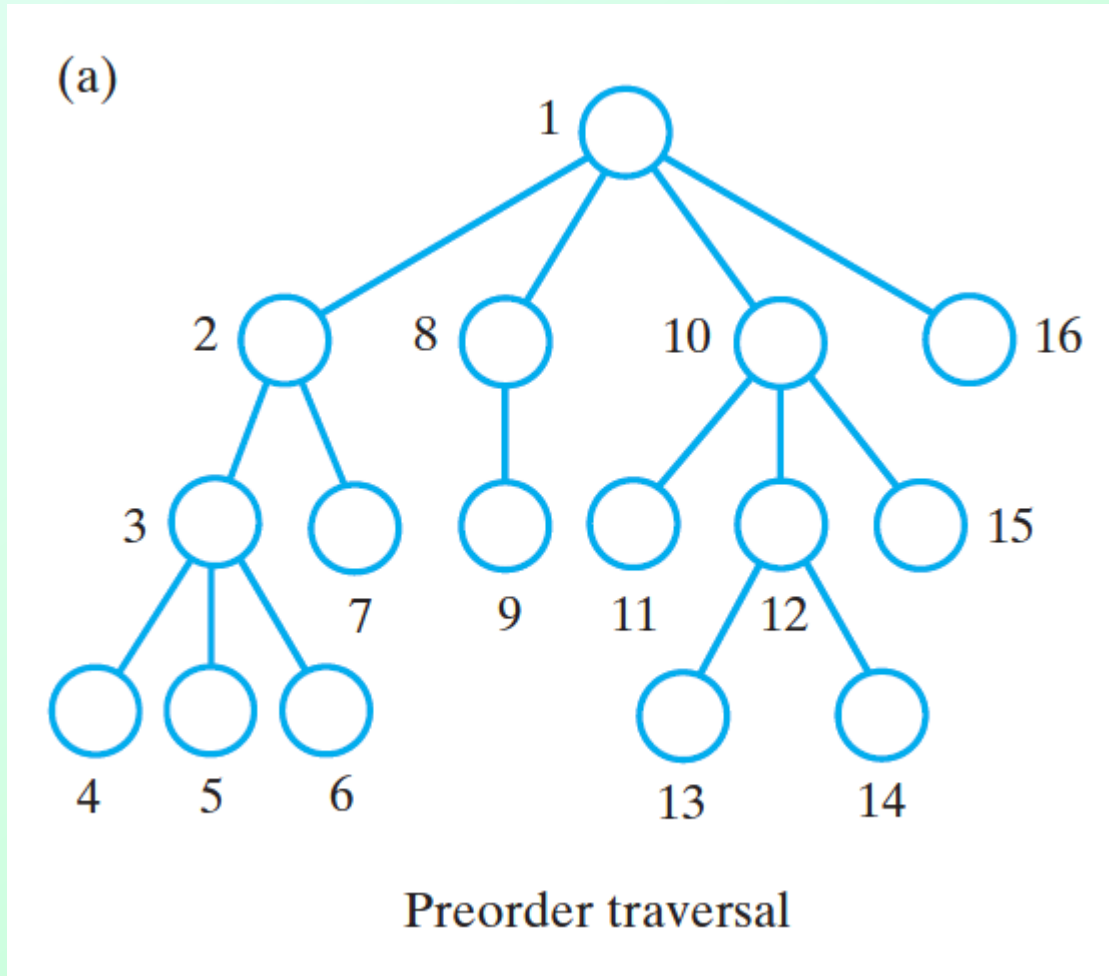


FIGURE 23-12 The visitation order of two traversals of a general tree: (a) preorder;

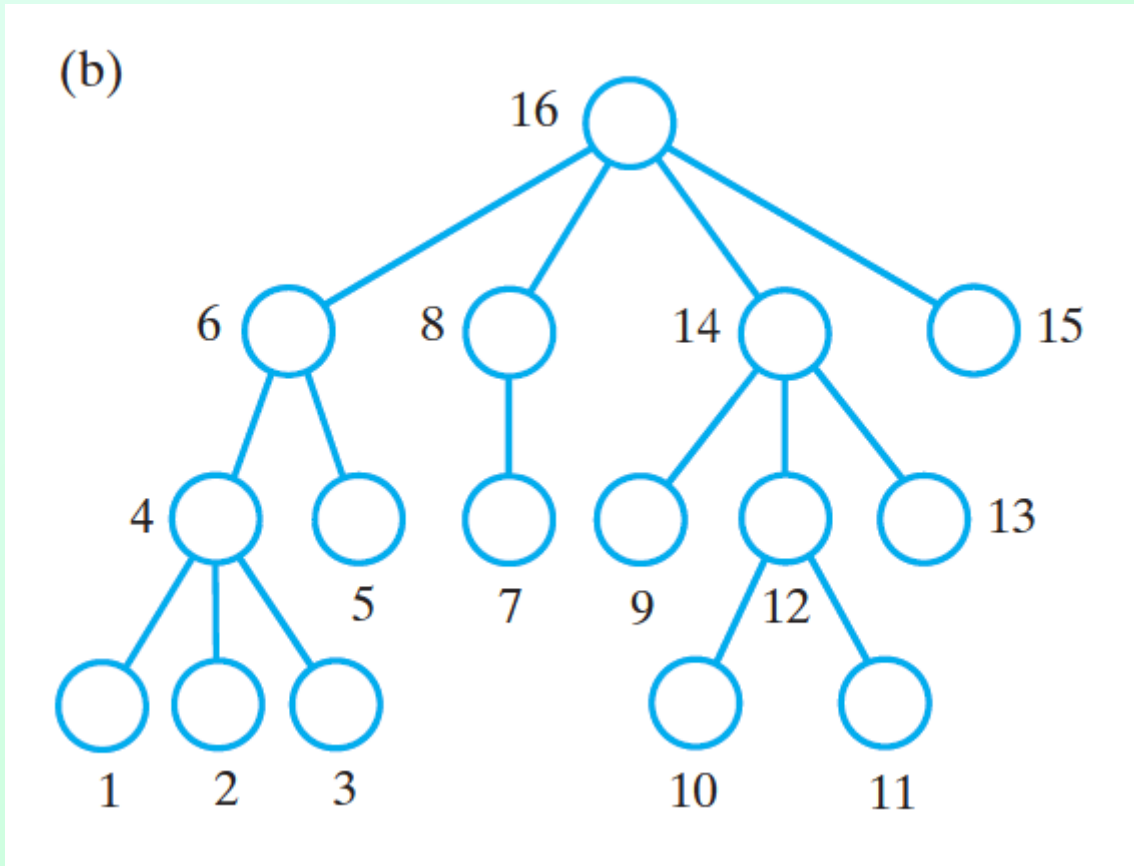


FIGURE 23-12 The visitation order of two traversals of a general tree: (a) postorder;



# Java Interfaces for Trees

- Interfaces for all trees
  - Interface which includes fundamental operations, [Listing 23-1](#)
  - Interface for traversals, [Listing 23-2](#)
  - Interface for binary trees, [Listing 23-3](#)

Note: Code listing files must be in same folder as PowerPoint files for links to work

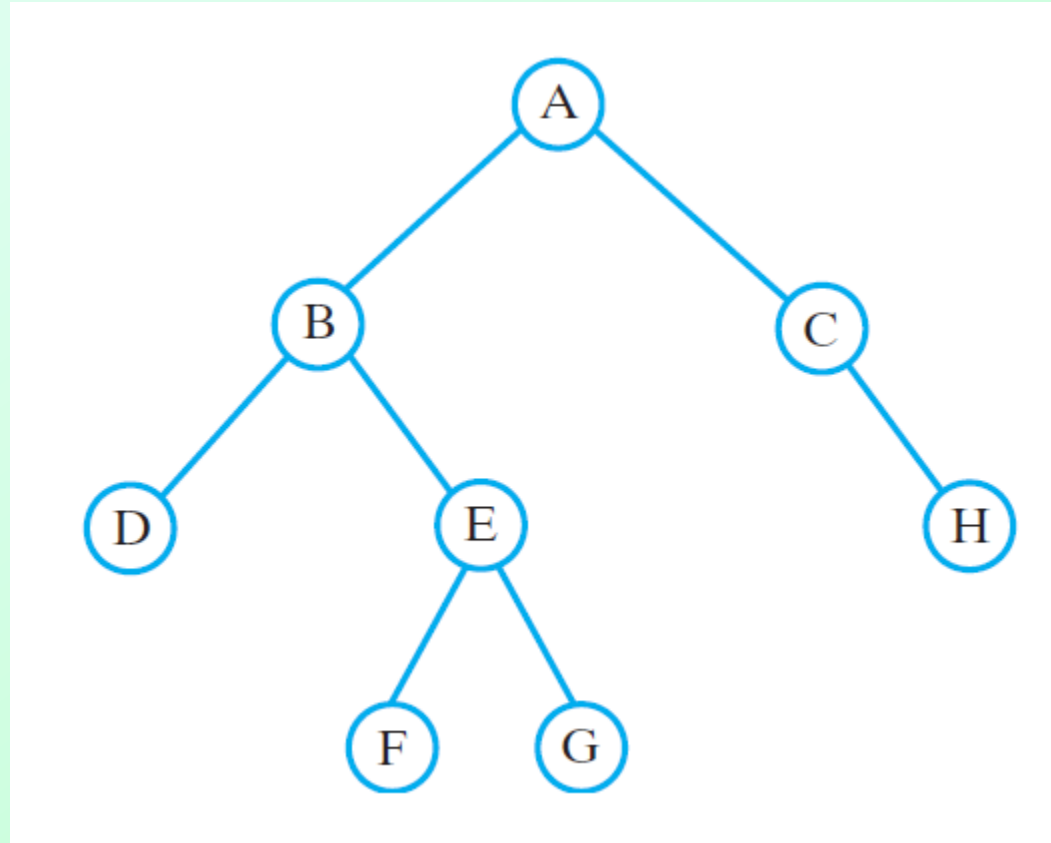
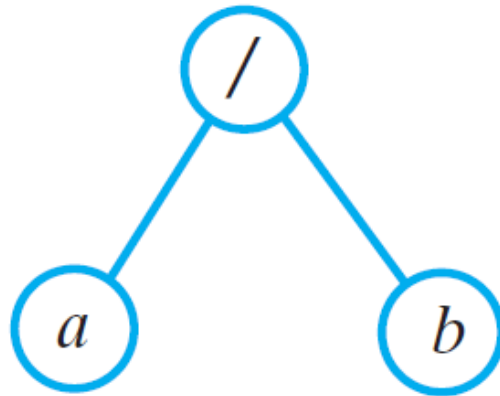


Figure 23-13 A binary tree whose nodes contain one-letter strings

# Expression Trees

- Use binary tree to represent expressions
  - Two operands
  - One binary operator
  - The operator is the root
- Can be used to evaluate an expression
  - Post order traversal
  - Each operand, then the operator

(a)  $a / b$



(b)  $a * b + c$

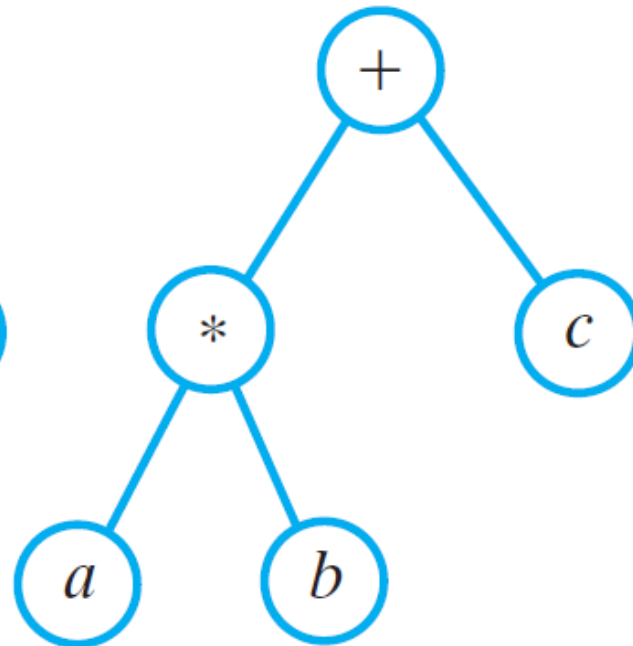


FIGURE 23-14 Expression trees for four algebraic expressions

(c)  $a * (b + c)$

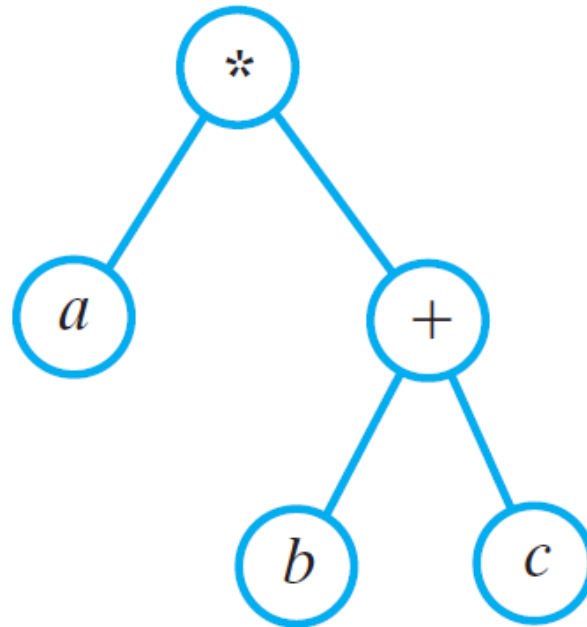


FIGURE 23-14 Expression trees for four algebraic expressions

(d)  $a * (b + c * d) / e$

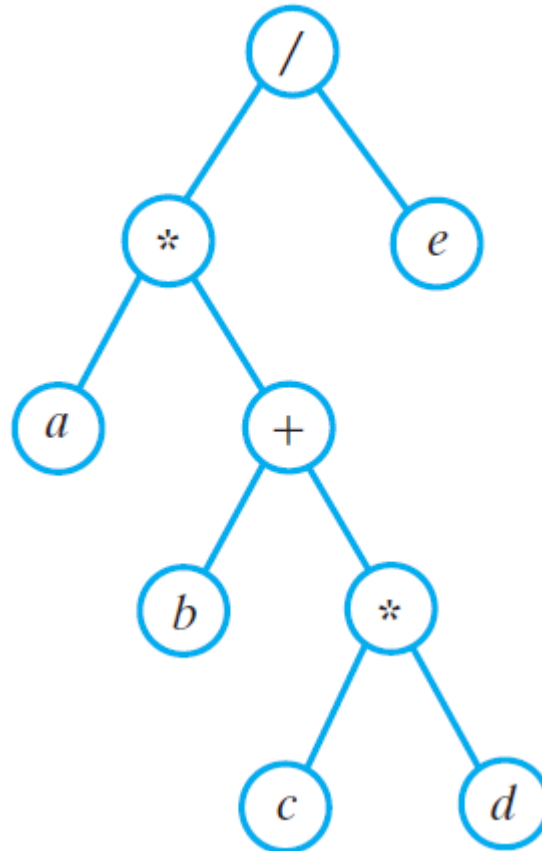


FIGURE 23-14 Expression trees for four algebraic expressions



# Decision Trees

- Used for expert systems
  - Helps users solve problems
  - Parent node asks question
  - Child nodes provide conclusion or further question
- Decision trees are generally n-ary
  - Expert system application often binary
- Note interface for decision tree, [Listing 23-4](#)

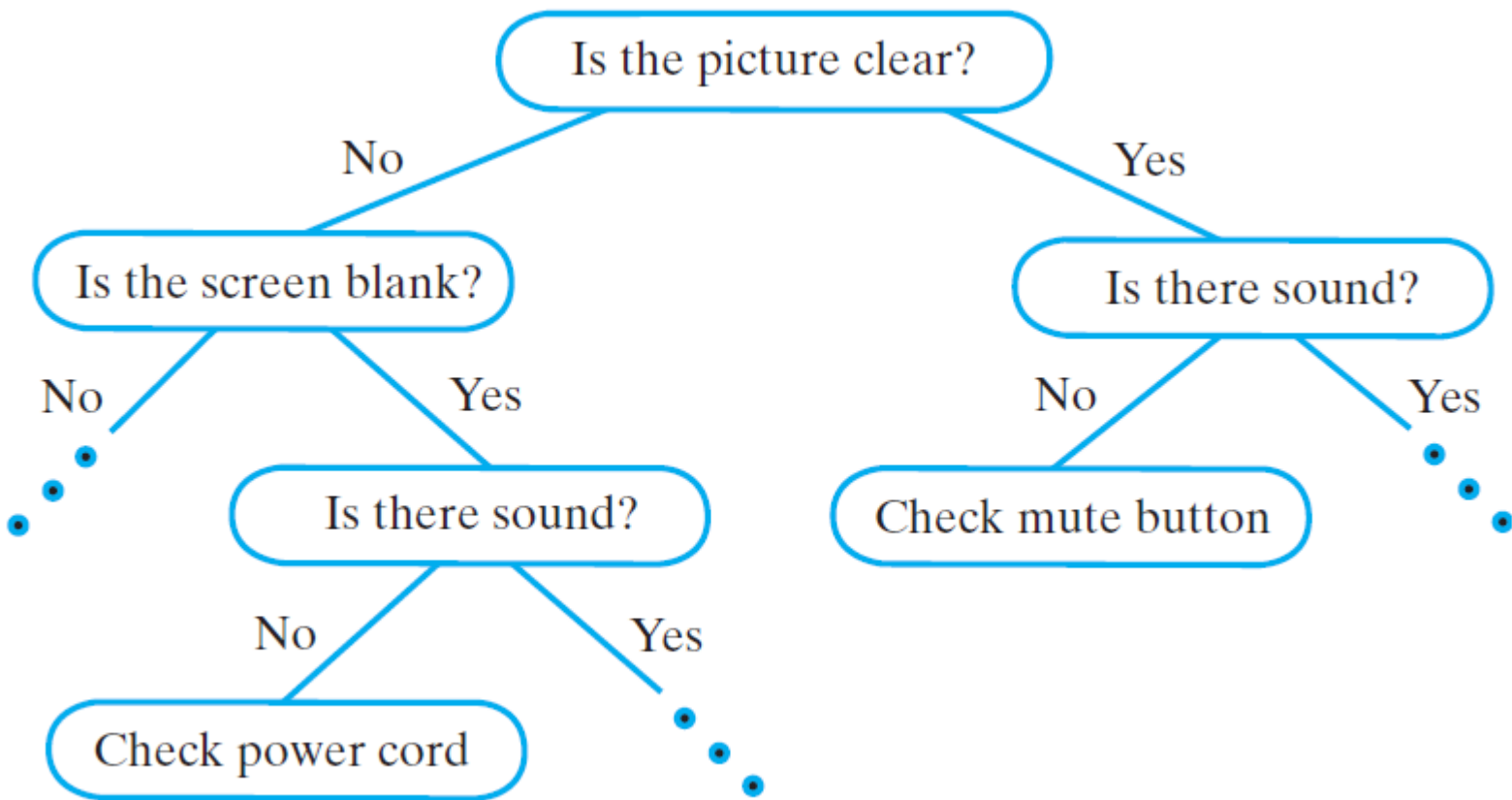


Figure 23-15 A portion of a binary decision tree

# Decision Trees

- Consider a guessing game
  - Program asks yes/no questions
  - Adds to its own decision tree as game progresses

- Example

```
Is it in North America?  
Yes  
My guess is U. S. A. Am I right?  
Yes  
I win.  
Play again?
```

- View class

**GuessingGame**, [Listing 23-5](#)

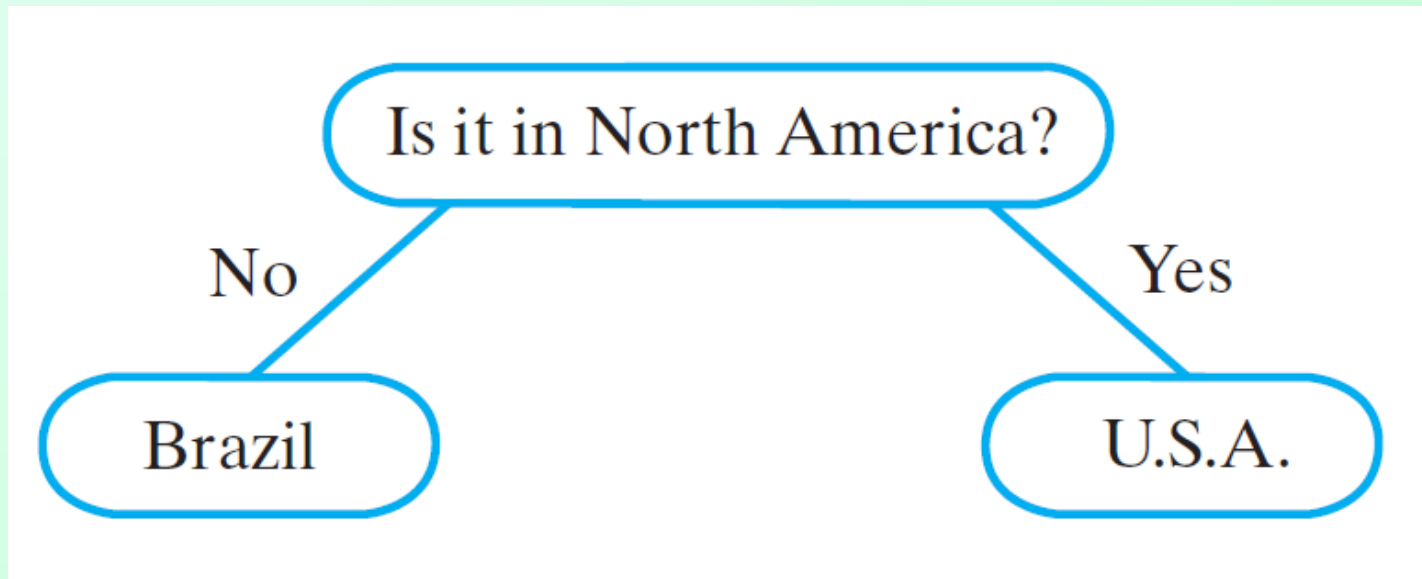


Figure 23-16 An initial decision tree for a guessing game

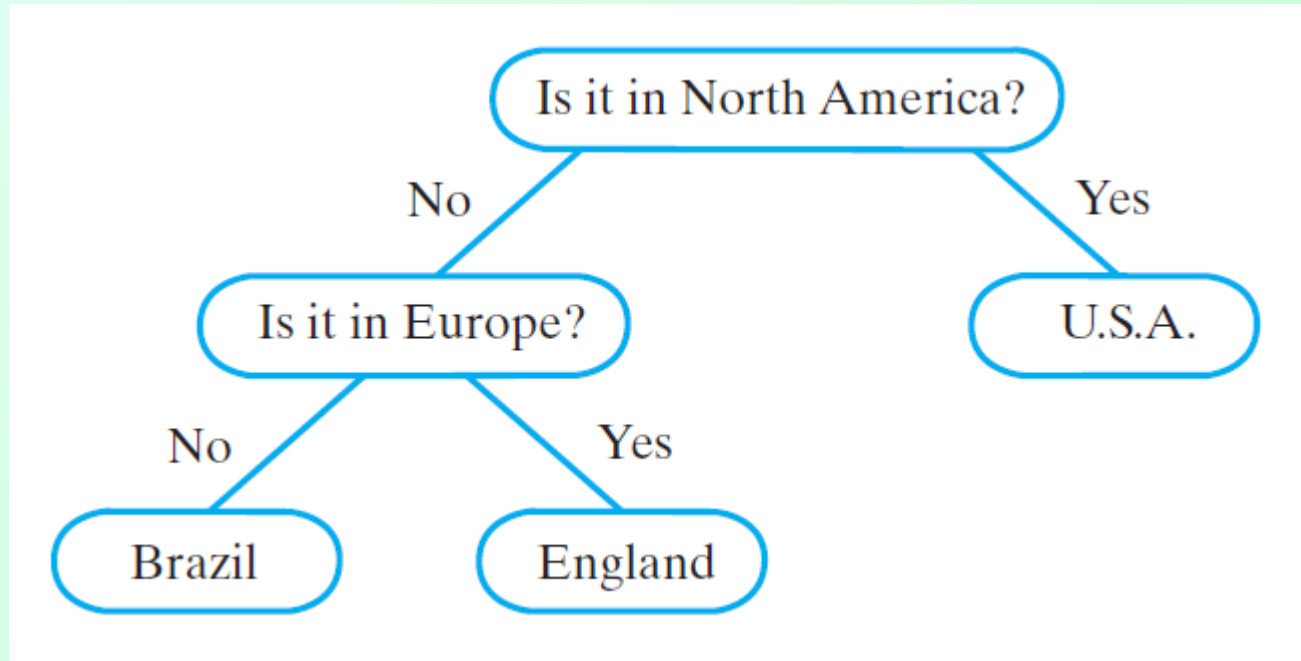


Figure 23-17 The decision tree for a guessing game after acquiring another fact

# Binary Search Trees

- Nodes contain **Comparable** objects
- For each node in a search tree:
  - Node's data greater than all data in node's left subtree
  - Node's data less than all data in node's right subtree

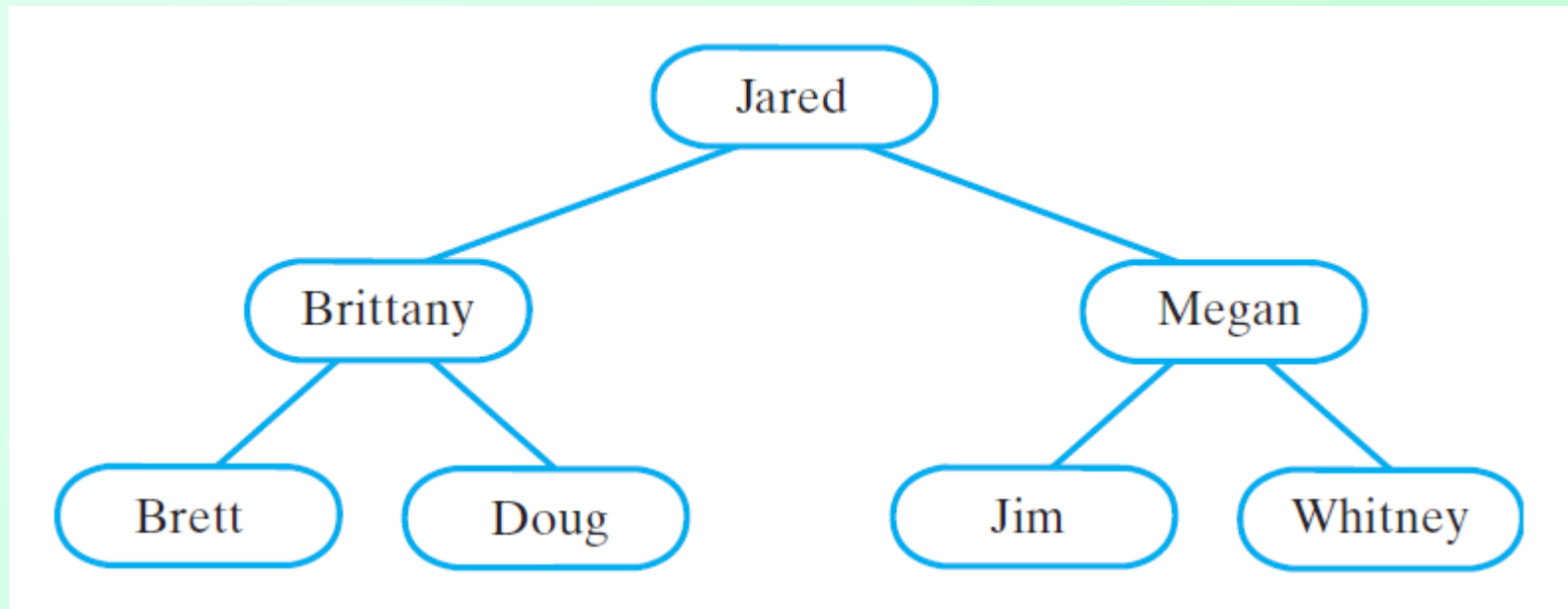
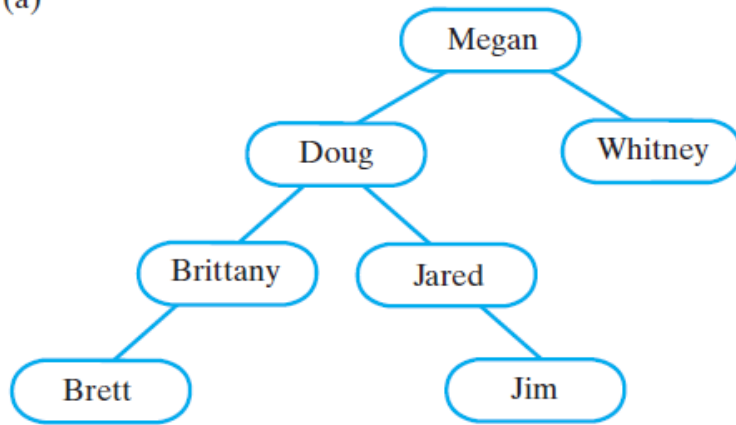


Figure 23-18 A binary search tree of names



(a)



(b)

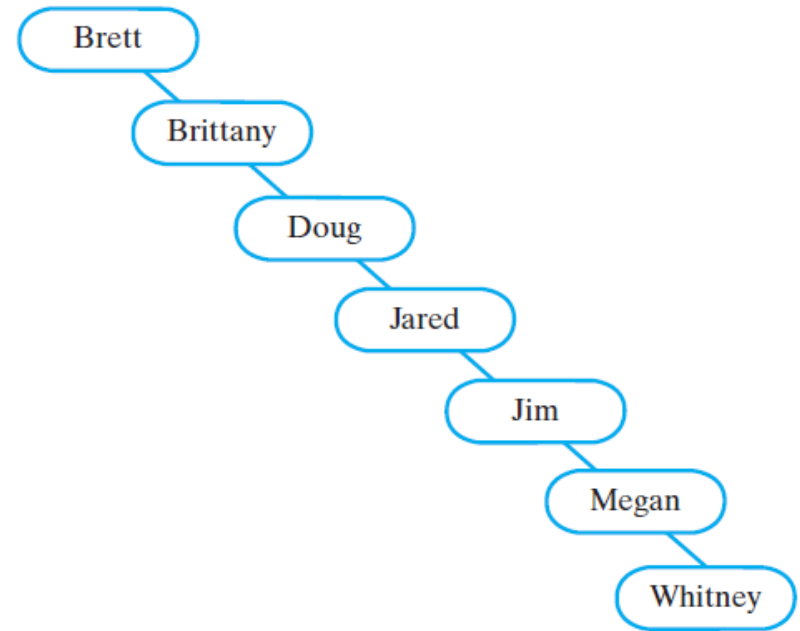


Figure 23-19 Two binary search trees containing the same data as the tree in Figure 23-18

# Heaps

- Complete binary tree: nodes contain **Comparable** objects
- Organization
  - Each node contains object no smaller (or no larger) than objects in descendants
  - Maxheap, object in node greater than or equal to descendant objects
  - Minheap, object in node less than or equal to descendant objects
- Interface, [Listing 23-6](#)

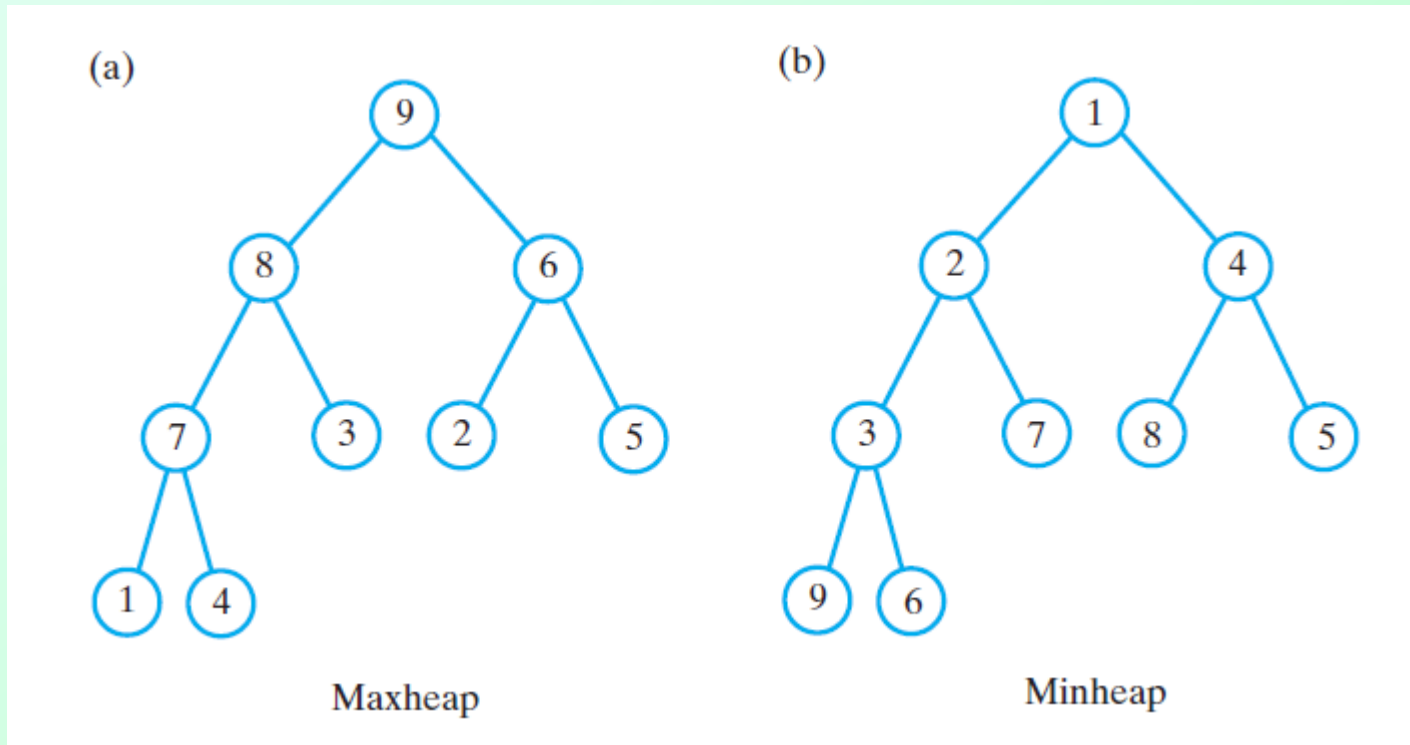


FIGURE 23-20 (a) A maxheap and (b) a minheap that contain the same values

# Priority Queues

- Use a heap to implement the ADT priority queue
- Assume class **MaxHeap** implements **MaxHeapInterface**
- View class **PriorityQueue**, [Listing 23-7](#)

# Examples of General Trees

- Parse trees
  - Use grammar rules for algebraic expression
  - Apply to elements of a string
  - Expression is root
  - Variables, operators are the leaves
- Must be general
  - To accommodate any expression

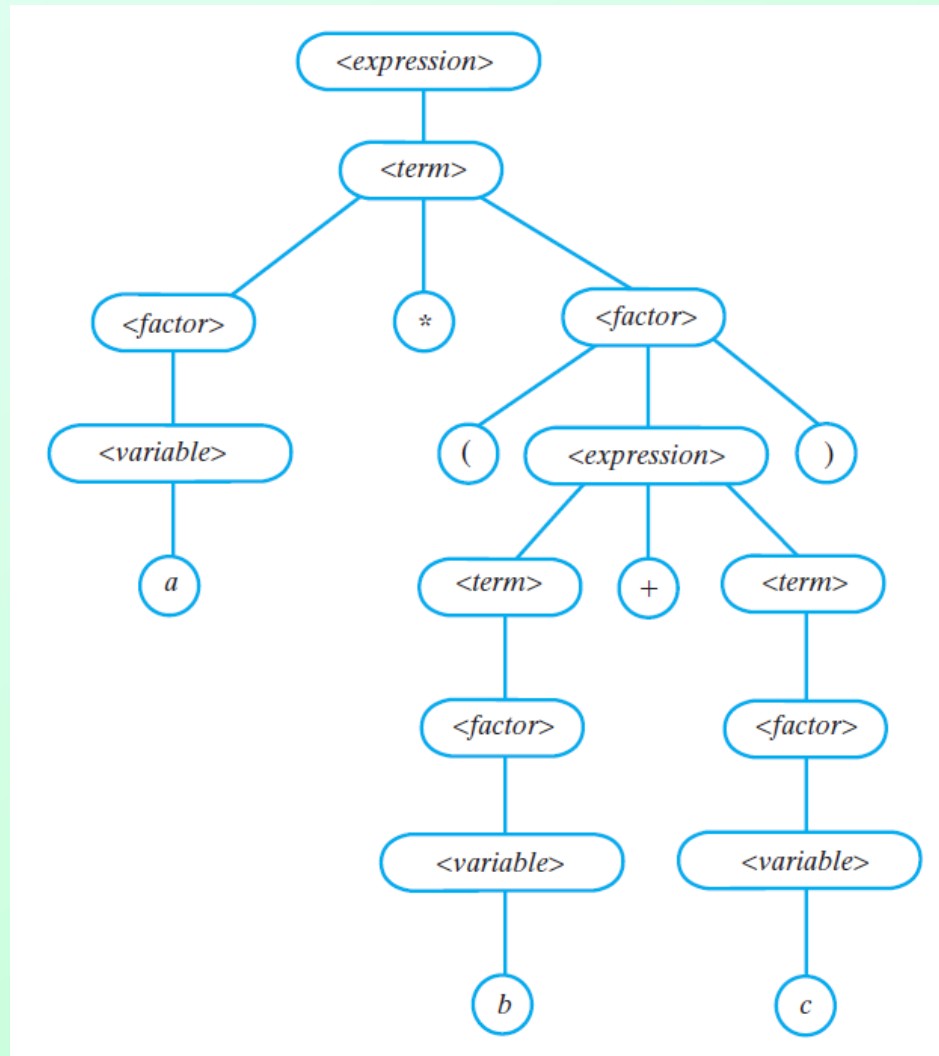


Figure 23-21 A parse tree for the algebraic expression  $a * (b + c)$

# Game Trees

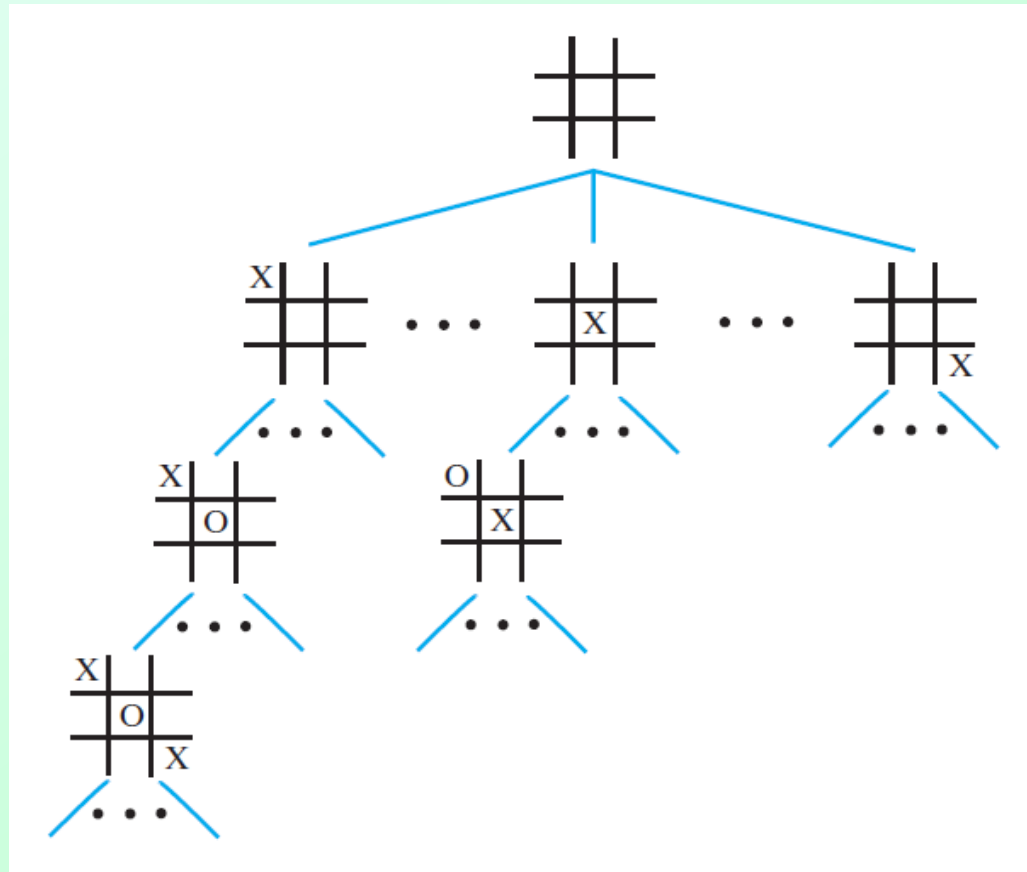


Figure 23-22 A portion of a game tree for tic-tac-toe



# End

## Chapter 23

