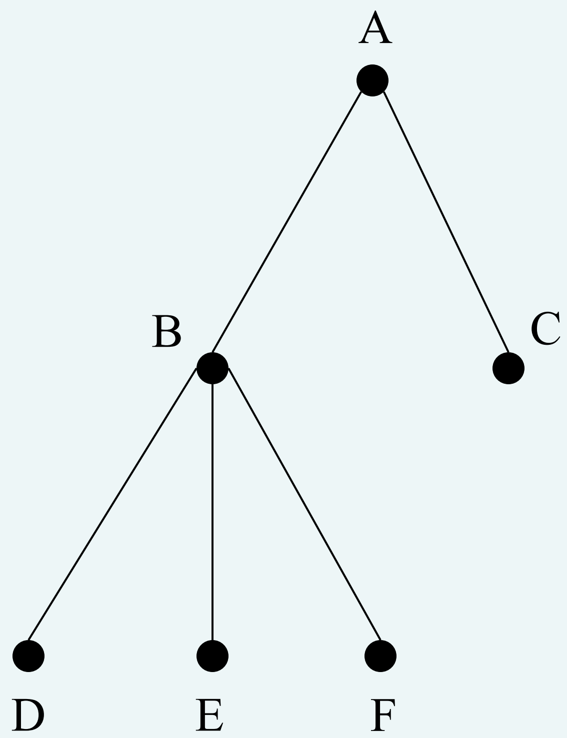
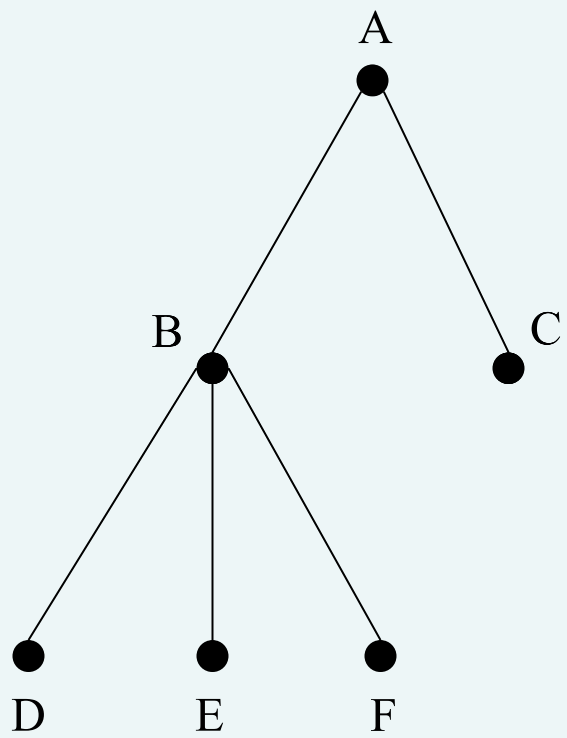
Chapter 11

Trees

Terminology



Terminology



Subtree

Terminology

- Definition of a general tree
  - A general tree T is a set of one or more nodes such that T is partitioned into disjoint subsets:
    - A single node r, the root
    - Sets that are general trees, called subtrees of r

Terminology

- Definition of a binary tree
  - A binary tree is a set T of nodes such that either
    - T is empty, or
    - T is partitioned into three disjoint subsets:
      - A single node r, the root
      - Two possibly empty sets that are binary trees, called left and right subtrees of r

Terminology

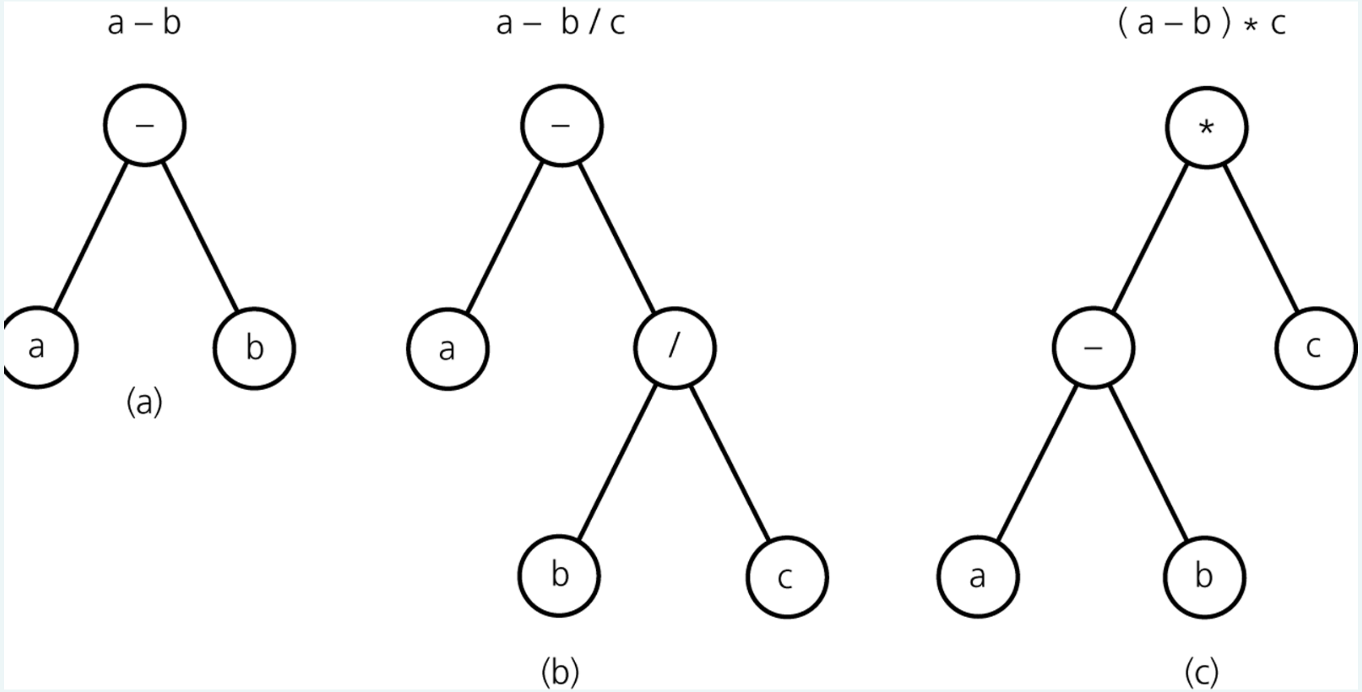


Figure 11-4

Binary trees that represent algebraic expressions

Terminology

- A binary search tree
  - A binary tree that has the following properties for each node n
    - n's value is greater than all values in its left subtree T_L
    - n's value is less than all values in its right subtree T_R
    - Both T_L and T_R are binary search trees

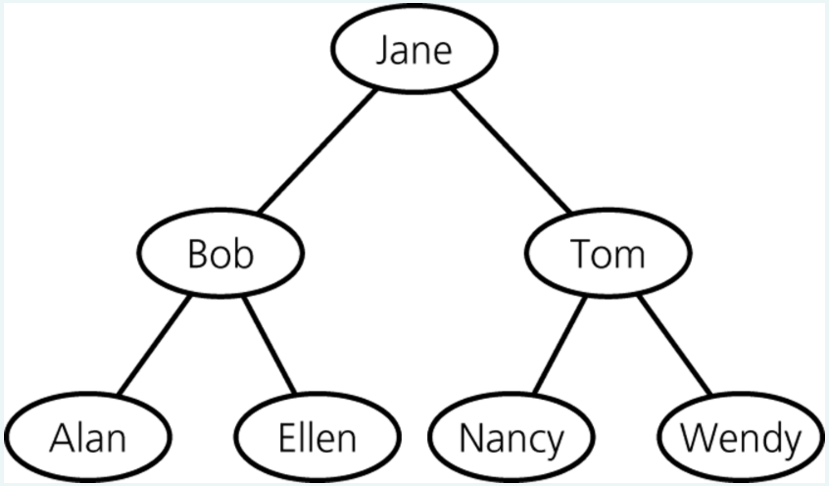


Figure 11-5

A binary search tree of names

Terminology

- The height of trees
  - Level of a node n in a tree T
    - If n is the root of T, it is at level 1
    - If n is not the root of T, its level is 1 greater than the level of its parent
  - Height of a tree T defined in terms of the levels of its nodes
    - If T is empty, its height is 0
    - If T is not empty, its height is equal to the maximum level of its nodes

Terminology

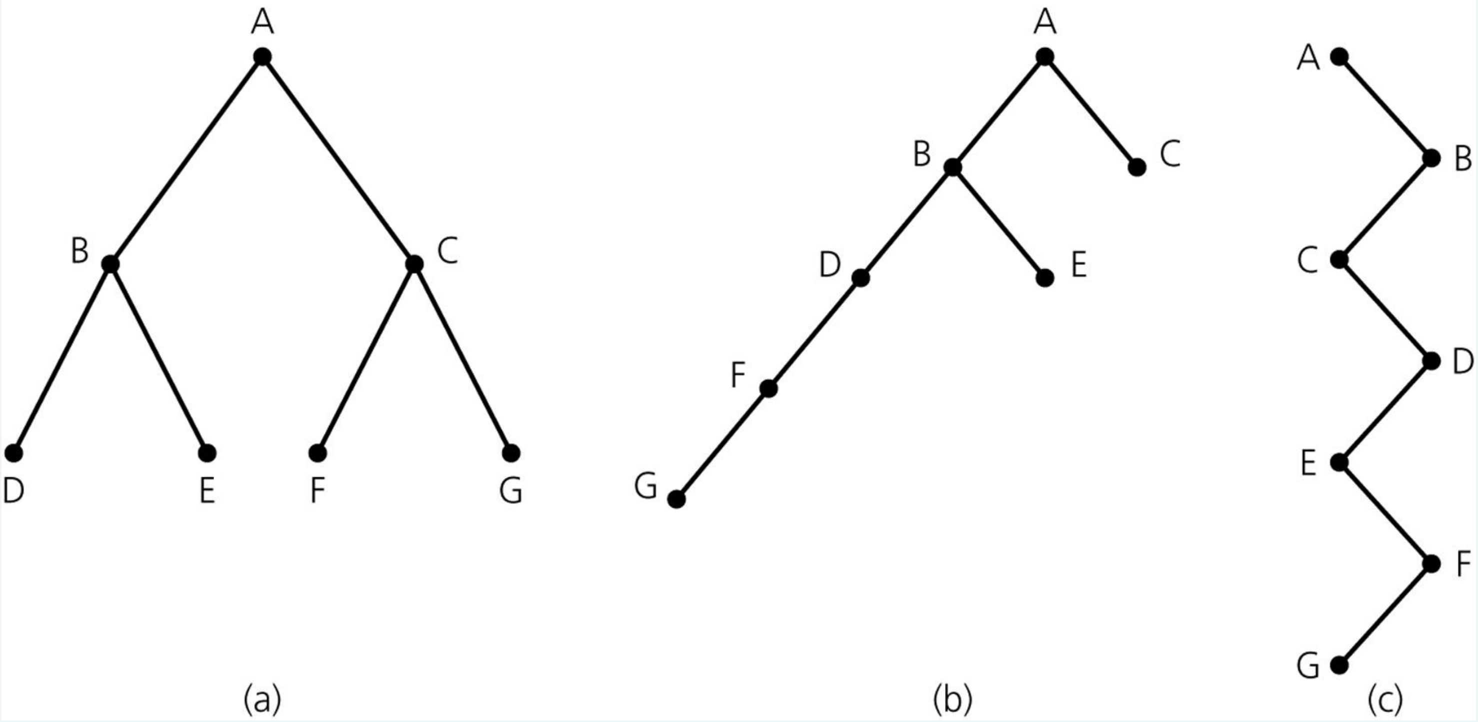


Figure 11-6

Binary trees with the same nodes but different heights

Terminology

- Full, complete, and balanced binary trees
  - Recursive definition of a full binary tree
    - If T is empty, T is a full binary tree of height 0
    - If T is not empty and has height $h > 0$, T is a full binary tree if its root's subtrees are both full binary trees of height $h - 1$

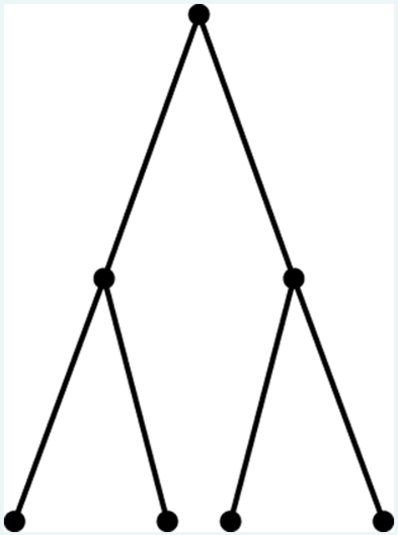


Figure 11-7

A full binary tree of height 3

Terminology

- Complete binary trees
  - A binary tree T of height h is complete if
    - All nodes at level h – 2 and above have two children each, and
    - When a node at level h – 1 has children, all nodes to its left at the same level have two children each, and
    - When a node at level h – 1 has one child, it is a left child

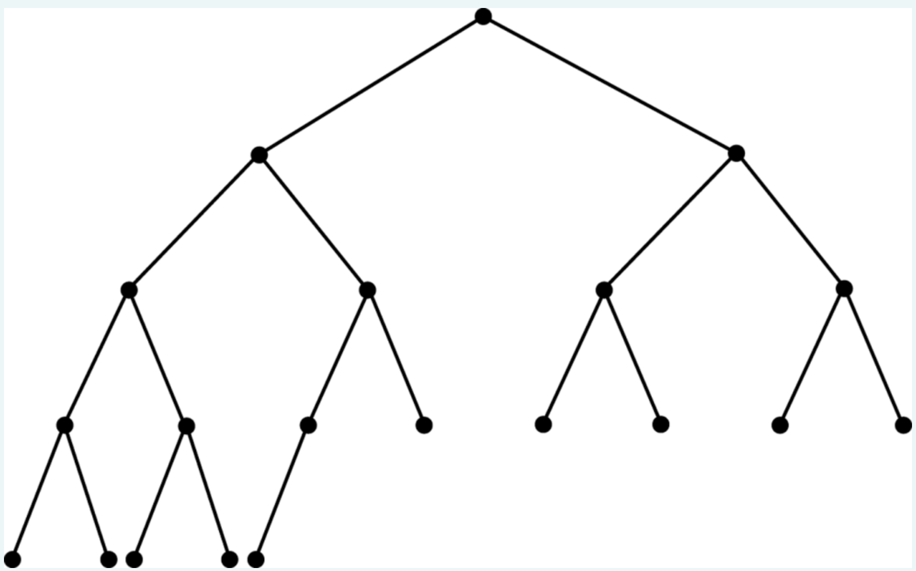


Figure 11-8

A complete binary tree

Terminology

- Balanced binary trees
  - A binary tree is balanced if the height of any node's right subtree differs from the height of the node's left subtree by no more than 1
- Full binary trees are complete
- Complete binary trees are balanced

Terminology

- Summary of tree terminology
  - General tree
    - A set of one or more nodes, partitioned into a root node and subsets that are general subtrees of the root
  - Parent of node n
    - The node directly above node n in the tree
  - Child of node n
    - A node directly below node n in the tree
  - Root
    - The only node in the tree with no parent

Terminology

- Summary of tree terminology (Continued)
  - Leaf
    - A node with no children
  - Siblings
    - Nodes with a common parent
  - Ancestor of node n
    - A node on the path from the root to n
  - Descendant of node n
    - A node on a path from n to a leaf
  - Subtree of node n
    - A tree that consists of a child (if any) of n and the child's descendants

Terminology

- Summary of tree terminology (Continued)
  - Height
    - The number of nodes on the longest path from the root to a leaf
  - Binary tree
    - A set of nodes that is either empty or partitioned into a root node and one or two subsets that are binary subtrees of the root
    - Each node has at most two children, the left child and the right child
  - Left (right) child of node n
    - A node directly below and to the left (right) of node n in a binary tree

Terminology

- Summary of tree terminology (Continued)
  - Left (right) subtree of node n
    - In a binary tree, the left (right) child (if any) of node n plus its descendants
  - Binary search tree
    - A binary tree where the value in any node n is greater than the value in every node in n's left subtree, but less than the value of every node in n's right subtree
  - Empty binary tree
    - A binary tree with no nodes

Terminology

- Summary of tree terminology (Continued)
  - Full binary tree
    - A binary tree of height h with no missing nodes
    - All leaves are at level h and all other nodes each have two children
  - Complete binary tree
    - A binary tree of height h that is full to level h – 1 and has level h filled in from left to right
  - Balanced binary tree
    - A binary tree in which the left and right subtrees of any node have heights that differ by at most 1

The ADT Binary Tree: Basic Operations of the ADT Binary Tree

- The operations available for a particular ADT binary tree depend on the type of binary tree being implemented
- Basic operations of the ADT binary tree
  - `createBinaryTree()`
  - `createBinaryTree(rootItem)`
  - `makeEmpty()`
  - `isEmpty()`
  - `getRootItem() throws TreeException`

General Operations of the ADT Binary Tree

- General operations of the ADT binary tree
  - `createBinaryTree (rootItem, leftTree, rightTree)`
  - `setRootItem(newItem)`
  - `attachLeft(newItem) throws TreeException`
  - `attachRight(newItem) throws TreeException`
  - `attachLeftSubtree(leftTree) throws TreeException`
  - `attachRightSubtree(rightTree) throws TreeException`
  - `detachLeftSubtree() throws TreeException`
  - `detachRightSubtree() throws TreeException`

Traversals of a Binary Tree

- A traversal algorithm for a binary tree visits each node in the tree
- Recursive traversal algorithms
  - Preorder traversal
  - Inorder traversal
  - Postorder traversal
- Traversal is O(n)

Traversal of a Binary Tree

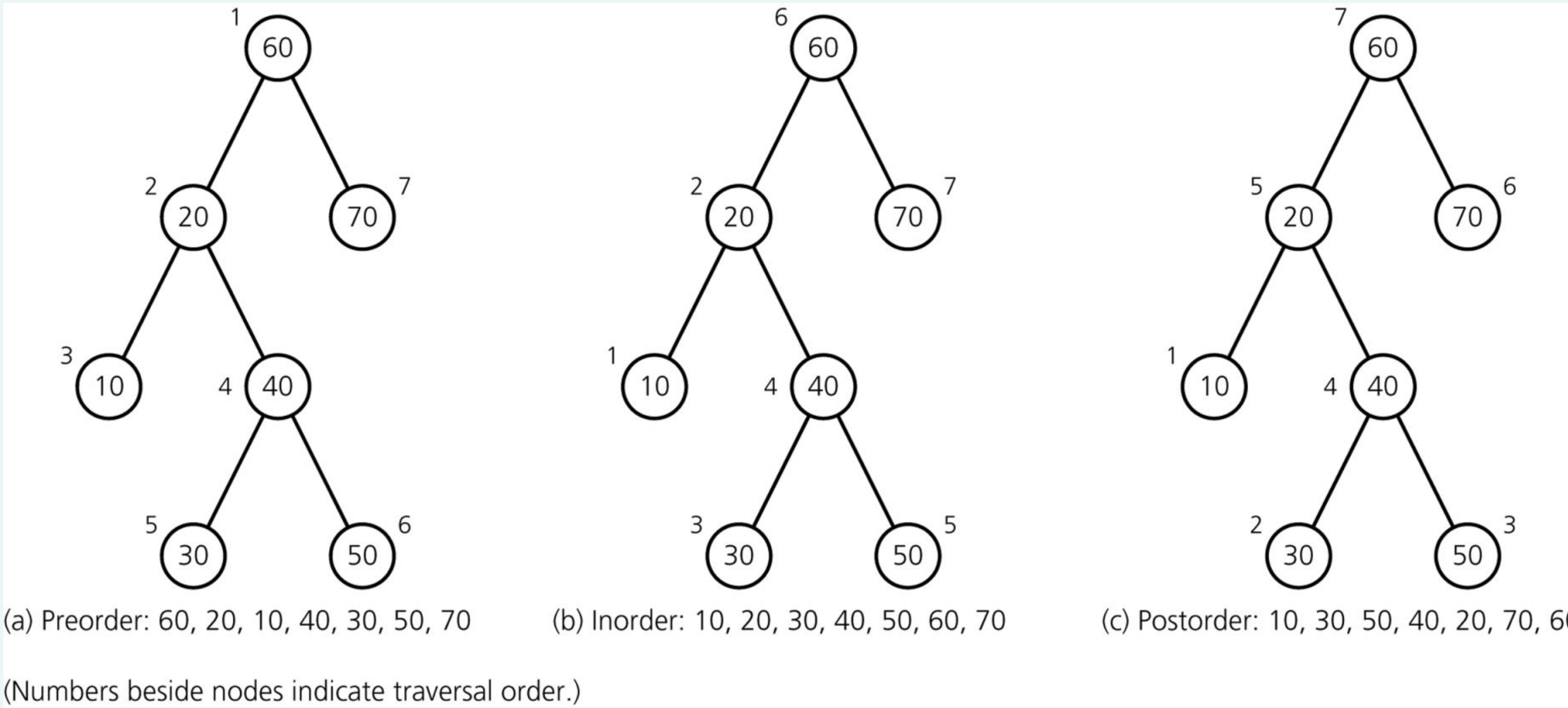


Figure 11-10

Traversals of a binary tree: a) preorder; b) inorder; c) postorder

Possible Representations of a Binary Tree

- An array-based representation
  - A Java class is used to define a node in the tree
  - A binary tree is represented by using an array of tree nodes
  - Each tree node contains a data portion and two indexes (one for each of the node's children)
  - Requires the creation of a free list which keeps track of available nodes

Possible Representations of a Binary Tree

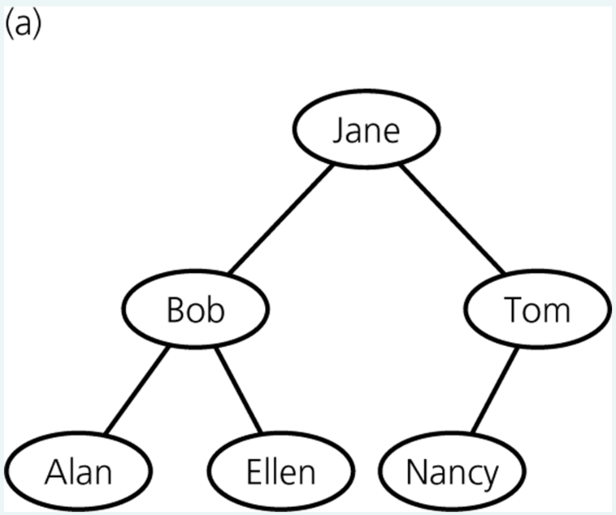


Figure 11-11a

a) A binary tree of names

(b)

tree

	item	leftChild	rightChild
0	Jane	1	2
1	Bob	3	4
2	Tom	5	–1
3	Alan	–1	–1
4	Ellen	–1	–1
5	Nancy	–1	–1
6	?	–1	7
7	?	–1	8
8	?	–1	9
⋮	⋮	⋮	⋮

root $\boxed{0}$

free $\boxed{6}$

Free list (rows 6 onward)

Figure 11-11b

b) its array-based implementations

Possible Representations of a Binary Tree

- An array-based representation of a complete tree
  - If the binary tree is complete and remains complete
    - A memory-efficient array-based implementation can be used

Possible Representations of a Binary Tree

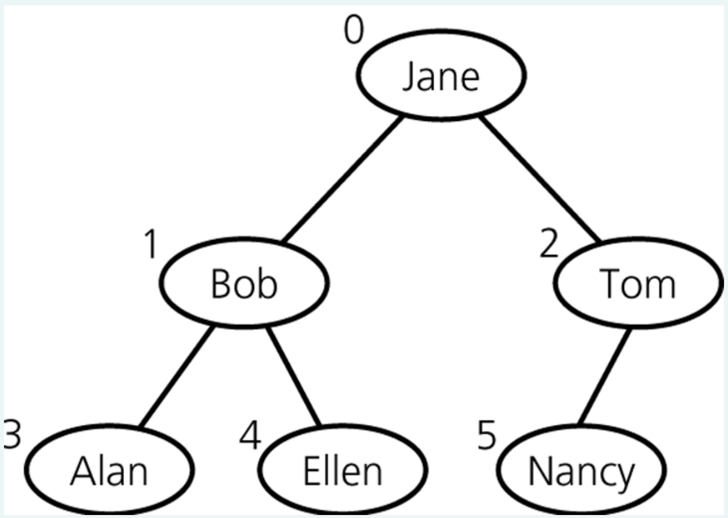


Figure 11-12

Level-by-level numbering of a complete binary tree

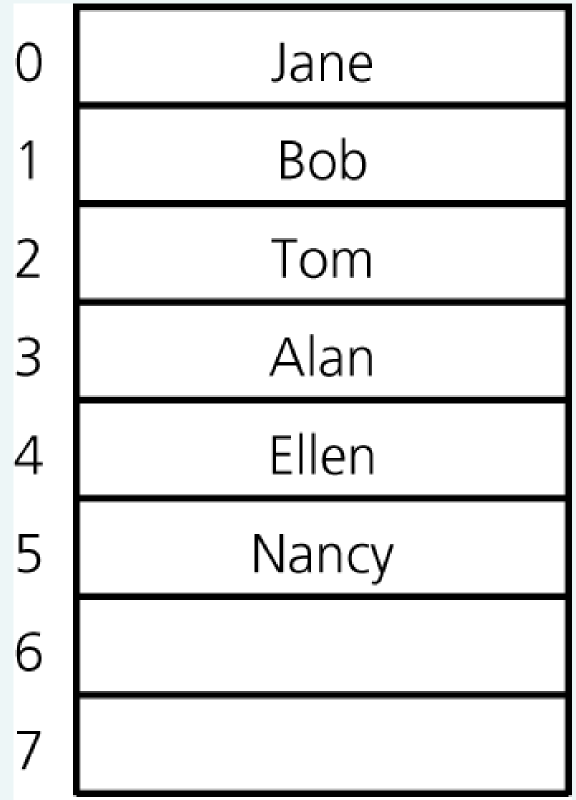


Figure 11-13

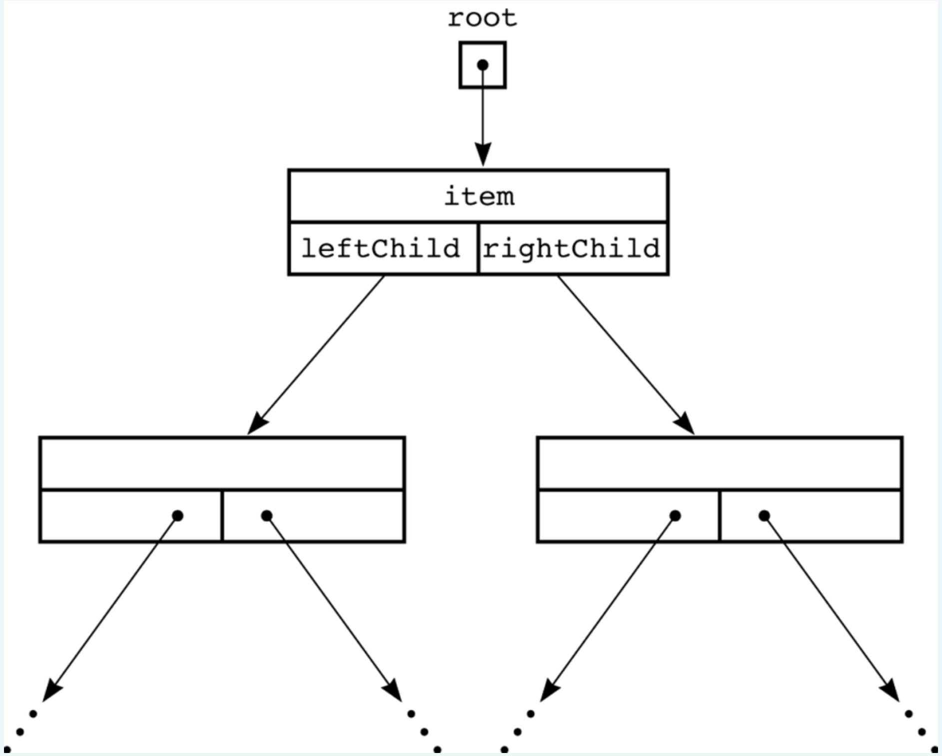
An array-based implementation of the complete binary tree in Figure 10-12

Possible Representations of a Binary Tree

- A reference-based representation
  - Java references can be used to link the nodes in the tree

Figure 11-14

A reference-based implementation of a binary tree



A Reference-Based Implementation of the ADT Binary Tree

- Classes that provide a reference-based implementation for the ADT binary tree
  - `TreeNode`
    - Represents a node in a binary tree
  - `TreeException`
    - An exception class
  - `BinaryTreeBasis`
    - An abstract class of basic tree operation
  - `BinaryTree`
    - Provides the general operations of a binary tree
    - Extends `BinaryTreeBasis`

Please open file *carrano_ppt11_B.ppt* to continue viewing chapter 11.

Chapter 11 (continued)

Trees

Tree Traversals Using an Iterator

- `TreeIterator`
  - Implements the Java `Iterator` interface
  - Provides methods to set the iterator to the type of traversal desired
  - Uses a queue to maintain the current traversal of the nodes in the tree
- Nonrecursive traversal (optional)
  - An iterative method and an explicit stack can be used to mimic actions at a return from a recursive call to `inorder`

The ADT Binary Search Tree

- A deficiency of the ADT binary tree which is corrected by the ADT binary search tree
  - Searching for a particular item
- Each node n in a binary search tree satisfies the following properties
  - n's value is greater than all values in its left subtree T_L
  - n's value is less than all values in its right subtree T_R
  - Both T_L and T_R are binary search trees

The ADT Binary Search Tree

- Record
  - A group of related items, called fields, that are not necessarily of the same data type
- Field
  - A data element within a record
- A data item in a binary search tree has a specially designated search key
  - A search key is the part of a record that identifies it within a collection of records
- `KeyedItem` class
  - Contains the search key as a data field and a method for accessing the search key
  - Must be extended by classes for items that are in a binary search tree

The ADT Binary Search Tree

- Operations of the ADT binary search tree
  - Insert a new item into a binary search tree
  - Delete the item with a given search key from a binary search tree
  - Retrieve the item with a given search key from a binary search tree
  - Traverse the items in a binary search tree in preorder, inorder, or postorder

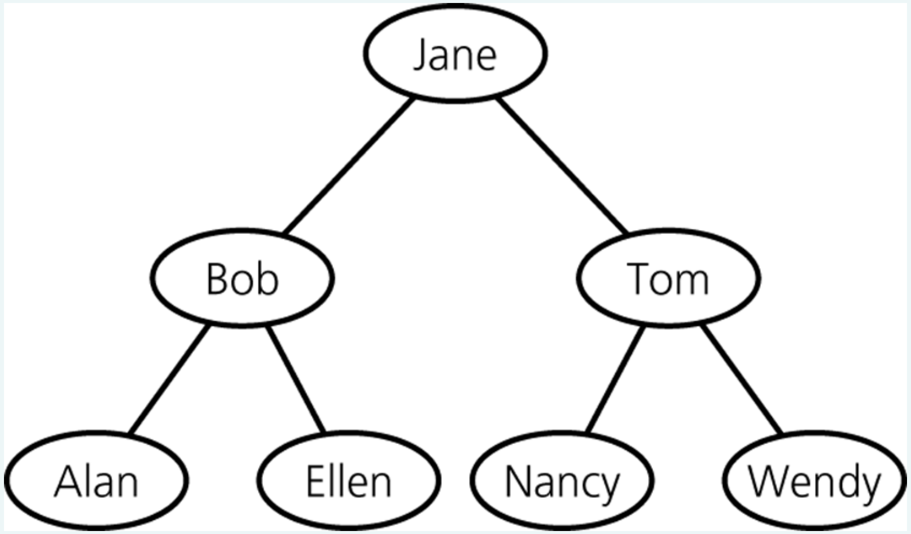


Figure 11-19

A binary search tree

Algorithms for the Operations of the ADT Binary Search Tree

- Since the binary search tree is recursive in nature, it is natural to formulate recursive algorithms for its operations
- A search algorithm
  - `search(bst, searchKey)`
    - Searches the binary search tree `bst` for the item whose search key is `searchKey`

Algorithms for the Operations of the ADT Binary Search Tree: Insertion

- `insertItem(treeNode, newItem)`
  - Inserts `newItem` into the binary search tree of which `treeNode` is the root

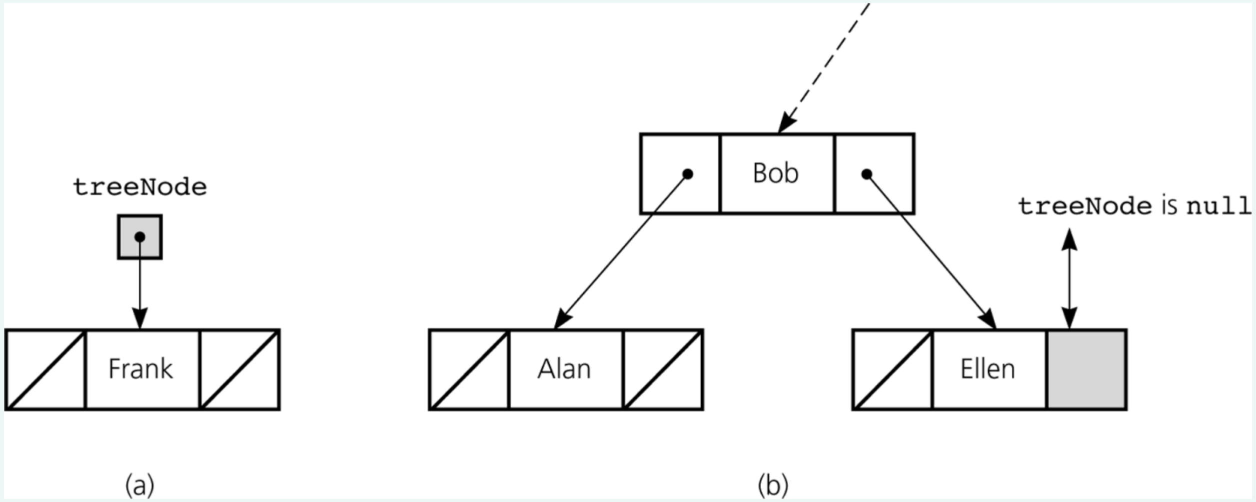


Figure 11-23a and 11-23b

a) Insertion into an empty tree; b) search terminates at a leaf

Algorithms for the Operations of the ADT Binary Search Tree: Insertion

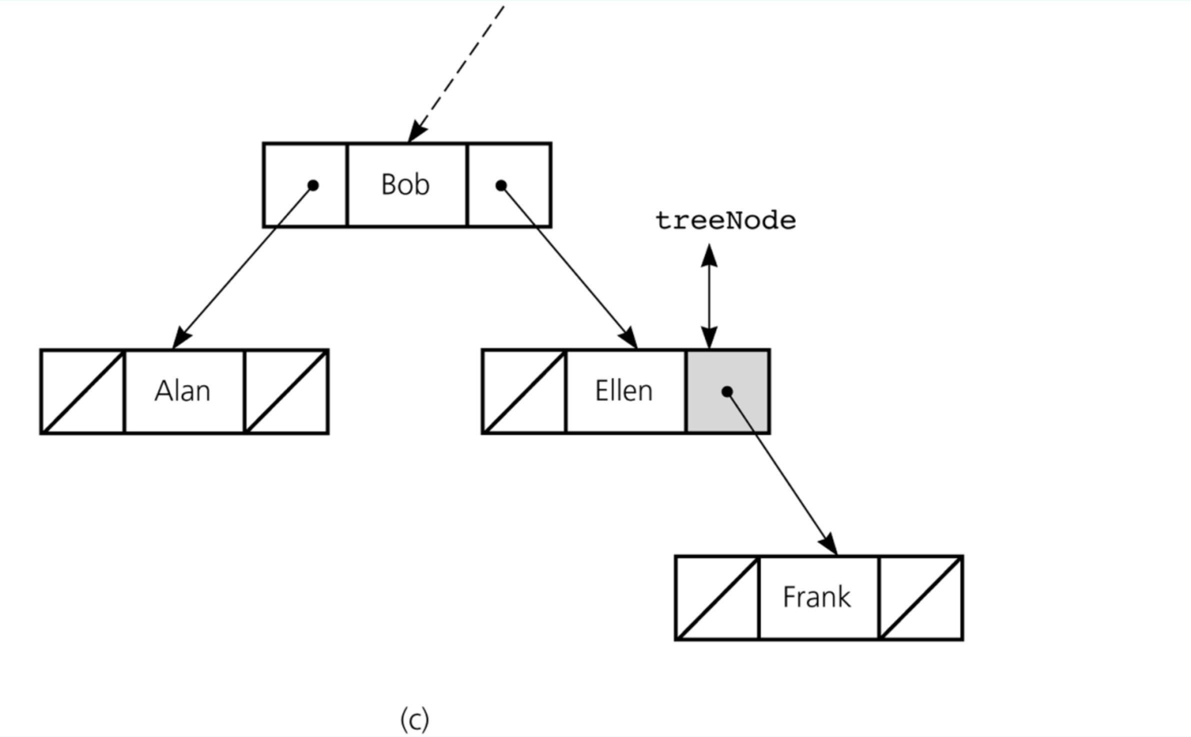


Figure 11-23c

c) insertion at a leaf

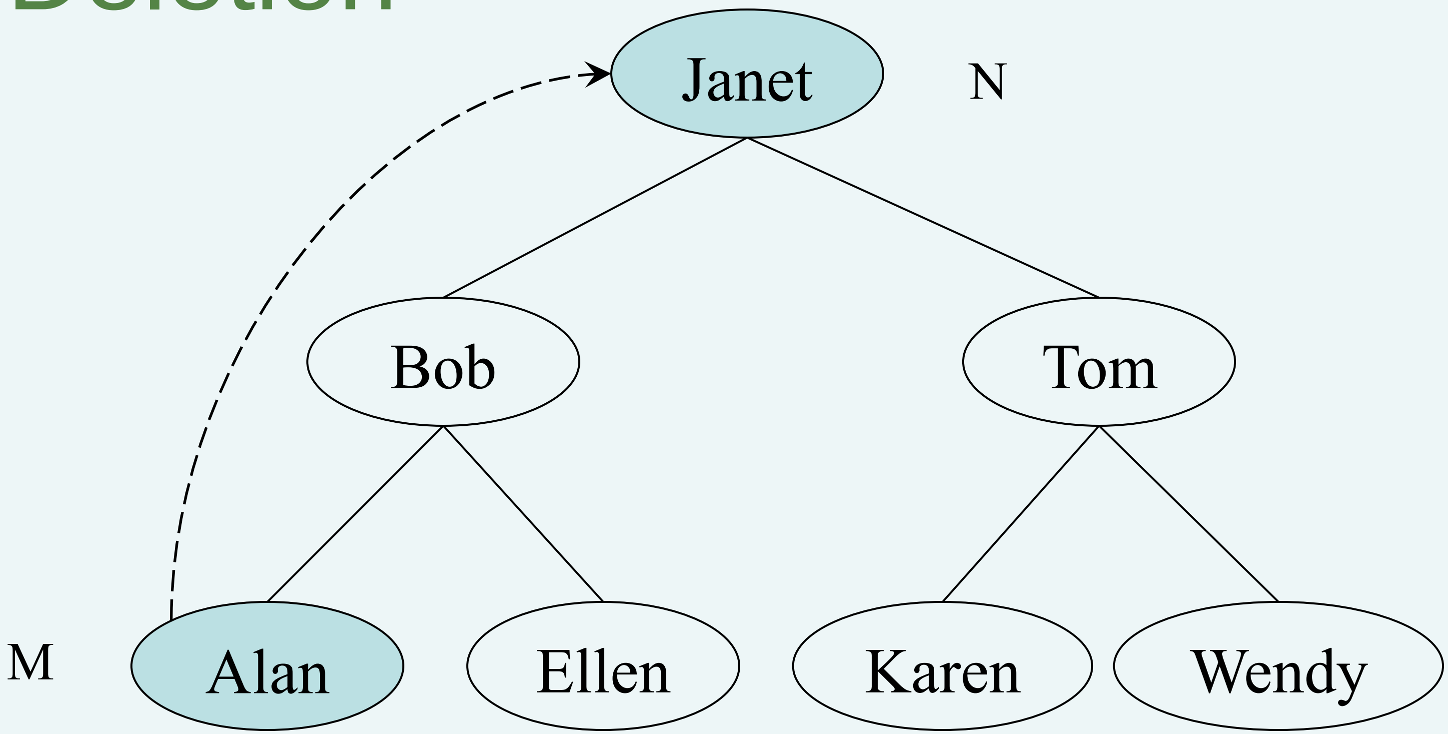
Algorithms for the Operations of the ADT Binary Search Tree: Deletion

- Steps for deletion
  - Use the search algorithm to locate the item with the specified key
  - If the item is found, remove the item from the tree
- Three possible cases for node N containing the item to be deleted
  - N is a leaf
  - N has only one child
  - N has two children

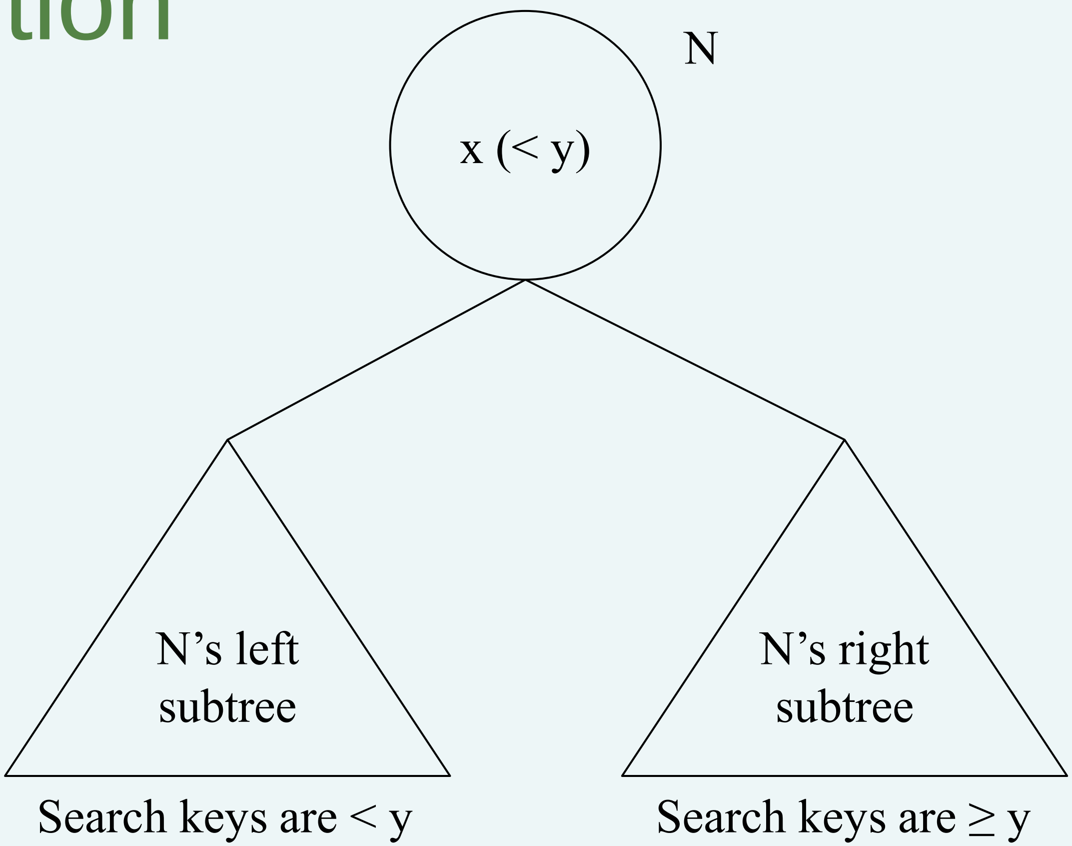
Algorithms for the Operations of the ADT Binary Search Tree: Deletion

- Strategies for deleting node N
  - If N is a leaf
    - Set the reference in N's parent to `null`
  - If N has only one child
    - Let N's parent adopt N's child
  - If N has two children
    - Locate another node M that is easier to remove from the tree than the node N
    - Copy the item that is in M to N
    - Remove the node M from the tree

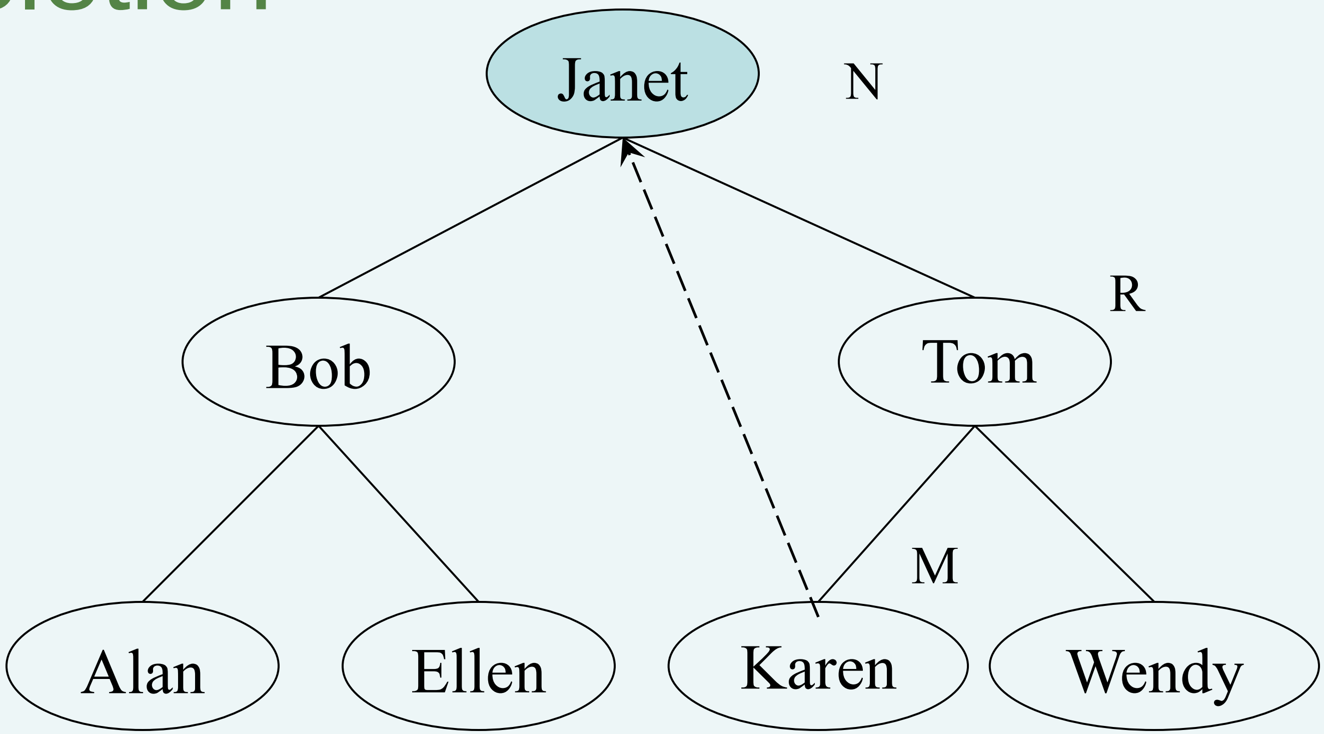
Algorithms for the Operations of the ADT Binary Search Tree: Deletion



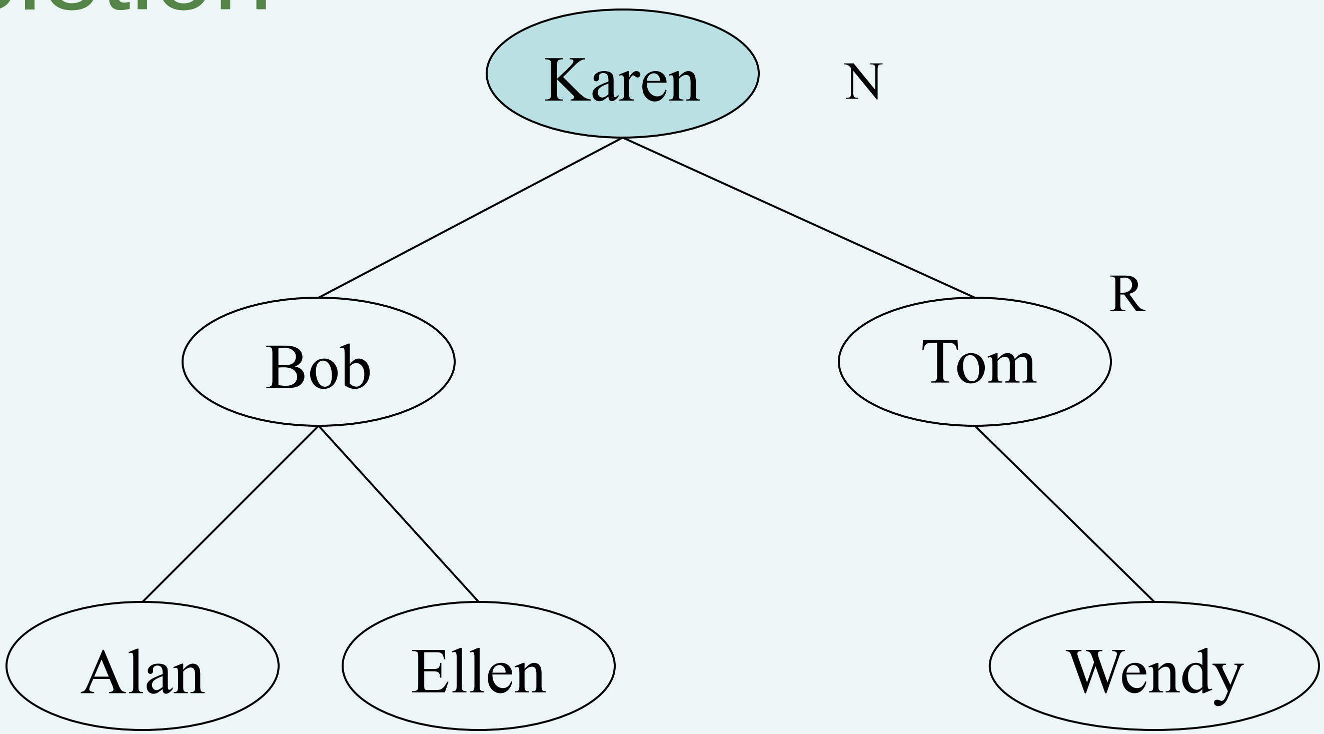
Algorithms for the Operations of the ADT Binary Search Tree: Deletion



Algorithms for the Operations of the ADT Binary Search Tree: Deletion



Algorithms for the Operations of the ADT Binary Search Tree: Deletion



Algorithms for the Operations of the ADT Binary Search Tree: Retrieval

- Retrieval operation can be implemented by refining the `search` algorithm
  - Return the item with the desired search key if it exists
  - Otherwise, return a `null` reference

Algorithms for the Operations of the ADT Binary Search Tree: Traversal

- Traversals for a binary search tree are the same as the traversals for a binary tree
- Theorem 11-1

  The inorder traversal of a binary search tree T will visit its nodes in sorted search-key order

A Reference-Based Implementation of the ADT Binary Search Tree

- `BinarySearchTree`
  - Extends `BinaryTreeBasis`
  - Inherits the following from `BinaryTreeBasis`
    - `isEmpty()`
    - `makeEmpty()`
    - `getRootItem()`
    - The use of the constructors
- `TreeIterator`
  - Can be used with `BinarySearchTree`

The Efficiency of Binary Search Tree Operations

- The maximum number of comparisons for a retrieval, insertion, or deletion is the height of the tree
- The maximum and minimum heights of a binary search tree
  - n is the maximum height of a binary tree with n nodes

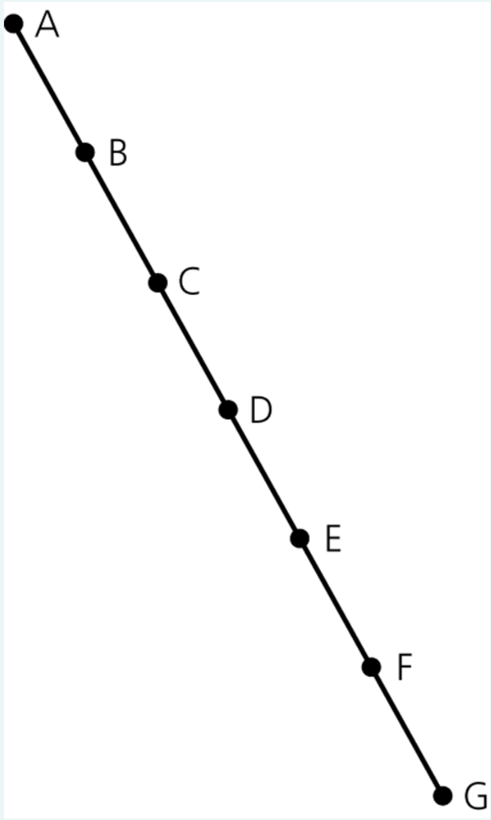


Figure 11-30

A maximum-height binary tree with seven nodes

The Efficiency of Binary Search Tree Operations

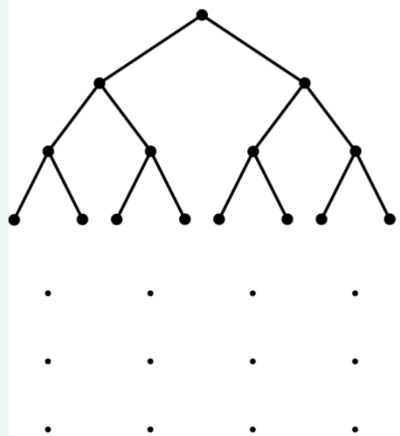
- Theorem 11-2

  A full binary tree of height $h \geq 0$ has $2^h - 1$ nodes
- Theorem 11-3

  The maximum number of nodes that a binary tree of height h can have is $2^h - 1$

Figure 11-32

Counting the nodes in a full binary tree of height *h*



Level	Number of nodes at this level	Number of nodes at this and previous levels
1	$1 = 2^0$	$1 = 2^1 - 1$
2	$2 = 2^1$	$3 = 2^2 - 1$
3	$4 = 2^2$	$7 = 2^3 - 1$
4	$8 = 2^3$	$15 = 2^4 - 1$
.	.	.
.	.	.
.	.	.
h	2^{h-1}	$2^h - 1$

The Efficiency of Binary Search Tree Operations

- Theorem 11-4

  The minimum height of a binary tree with n nodes is $\lceil \log_2(n+1) \rceil$

- The height of a particular binary search tree depends on the order in which insertion and deletion operations are performed

Operation	Average case	Worst case
Retrieval	O(log n)	O(n)
Insertion	O(log n)	O(n)
Deletion	O(log n)	O(n)
Traversal	O(n)	O(n)

Figure 11-34

The order of the retrieval, insertion, deletion, and traversal operations for the reference-based implementation of the ADT binary search tree

Treesort

- Treesort
  - Uses the ADT binary search tree to sort an array of records into search-key order
  - Efficiency
    - Average case: $O(n * \log n)$
    - Worst case: $O(n^2)$

Saving a Binary Search Tree in a File

- Two algorithms for saving and restoring a binary search tree
  - Saving a binary search tree and then restoring it to its original shape
    - Uses preorder traversal to save the tree to a file
  - Saving a binary tree and then restoring it to a balanced shape
    - Uses inorder traversal to save the tree to a file
    - Can be accomplished if
      - The data is sorted
      - The number of nodes in the tree is known

The JCF Binary Search Algorithm

- JCF has two binary search methods
  - Based on the natural ordering of elements:

static <T> **int**
binarySearch (List<? **extends** Comparable<? **super** T>> list, T key)

  - Based on a specified Comparator:

static <T> **int** binarySearch (List<? **extends** T> list, T key,
Comparator<? **super** T> c)

General Trees

- An n-ary tree
  - A generalization of a binary tree whose nodes each can have no more than n children

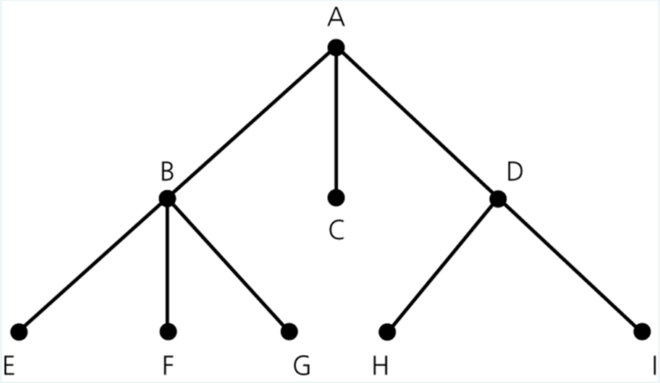


Figure 11-38

A general tree

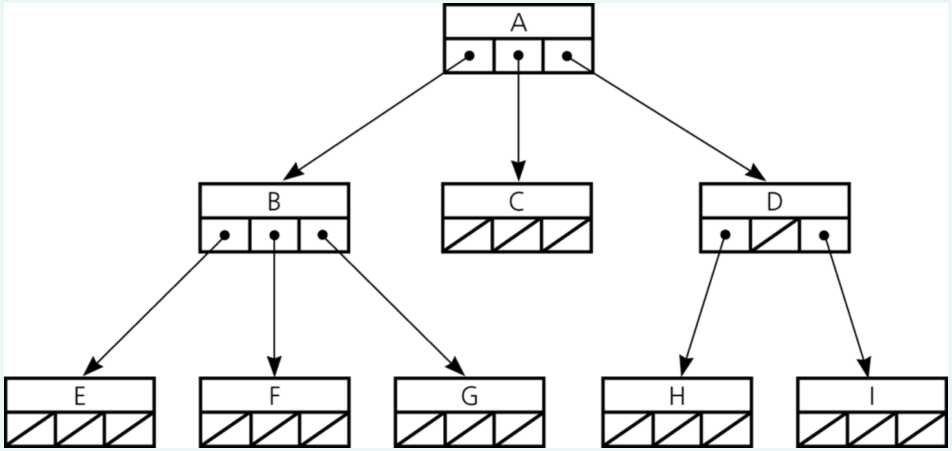


Figure 11-41

An implementation of the *n*-ary tree in Figure 11-38

Summary

- Binary trees provide a hierarchical organization of data
- Implementation of binary trees
  - The implementation of a binary tree is usually referenced-based
  - If the binary tree is complete, an efficient array-based implementation is possible
- Traversing a tree is a useful operation
- The binary search tree allows you to use a binary search-like algorithm to search for an item with a specified value

Summary

- Binary search trees come in many shapes
  - The height of a binary search tree with n nodes can range from a minimum of $\lceil \log_2(n + 1) \rceil$ to a maximum of n
  - The shape of a binary search tree determines the efficiency of its operations
- An inorder traversal of a binary search tree visits the tree's nodes in sorted search-key order
- The treesort algorithm efficiently sorts an array by using the binary search tree's insertion and traversal operations

Summary

- Saving a binary search tree to a file
  - To restore the tree as a binary search tree of minimum height
    - Perform inorder traversal while saving the tree to a file
  - To restore the tree to its original form
    - Perform preorder traversal while saving the tree to a file