Trees

Chapter 6

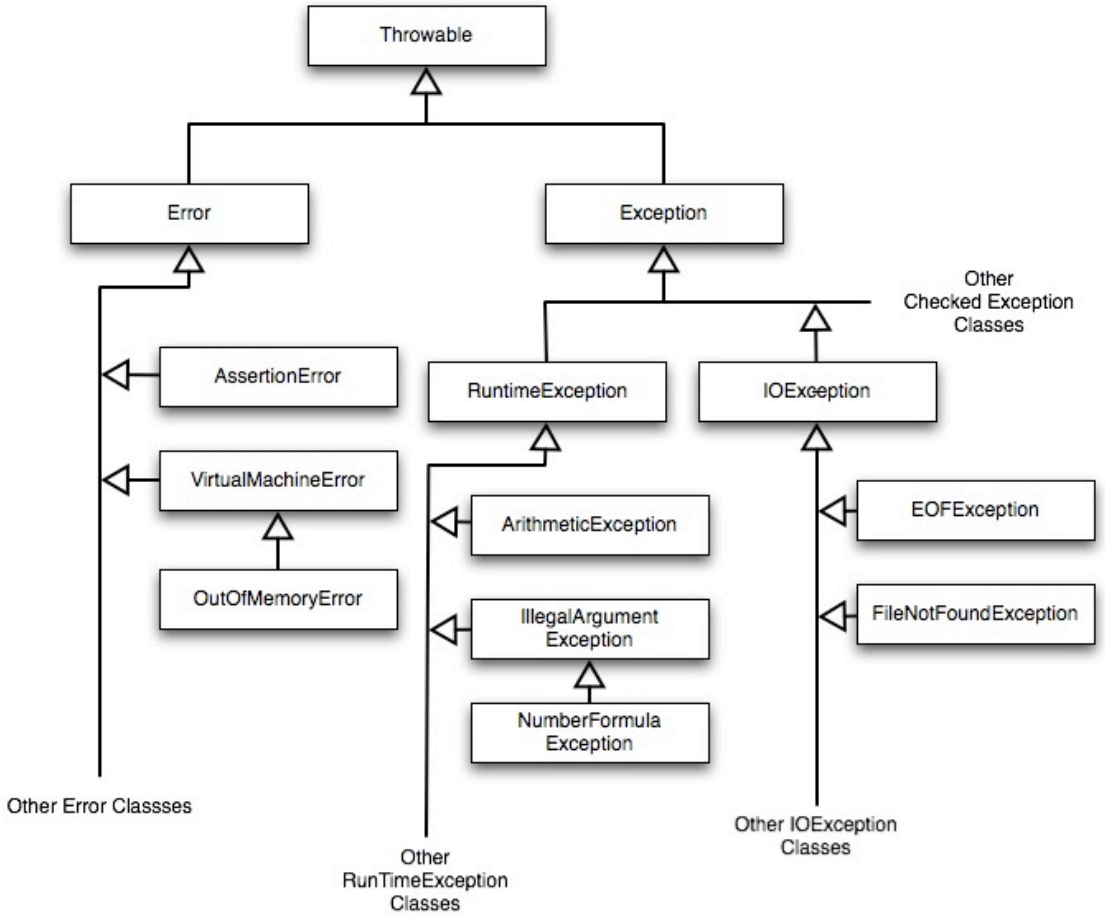
Chapter Objectives

- Use of trees to represent hierarchical organizations of information
- Use of recursion to process trees
  - Tree traversals
- Binary tree types and there implementation
  - Binary trees, binary search trees, heaps
- Use of binary search trees to store information
- Use of Huffman trees to encode characters efficiently

Tree Terminology

- A ***rooted tree*** consists of a collection of elements or nodes, with each node may be linked to its successors
  - The node at the top of a tree is called its ***root***
  - The links from a node to its successors are called ***branches***
  - The node without successors are called ***leaves***
- The successors of a node are called its ***children***
- There is at most one predecessor of a node, and that predecessor is called its ***parent***
- Nodes that have the same parent are ***siblings***
- A generalization of the parent-child relationship is the ***ancestor-descendent*** relationship

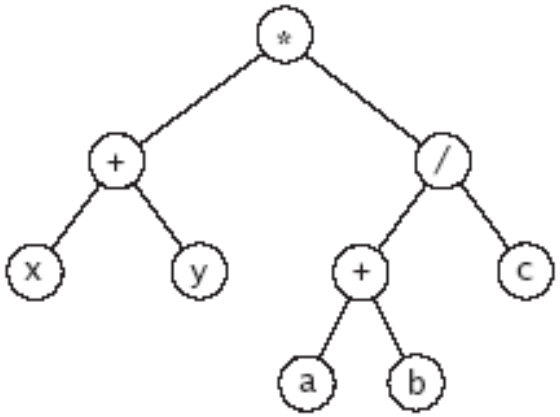
Tree Example



Example 1: Expression Tree

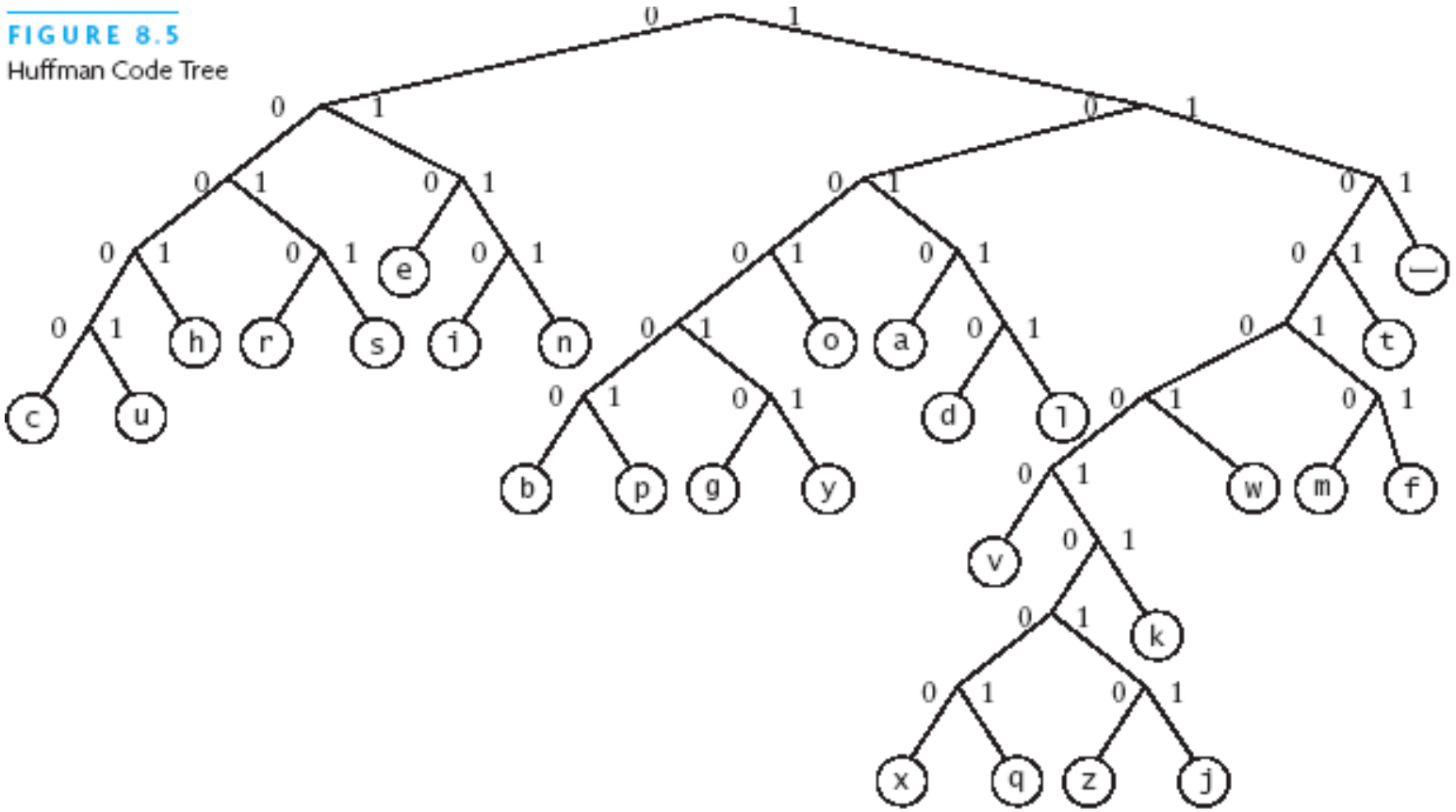
- Each node contains an operator or an operand

FIGURE 8.4
Expression Tree



Example 2: Huffman Tree

- Represents Huffman codes for characters that might appear in a text file
- Uses different numbers of bits to encode letters

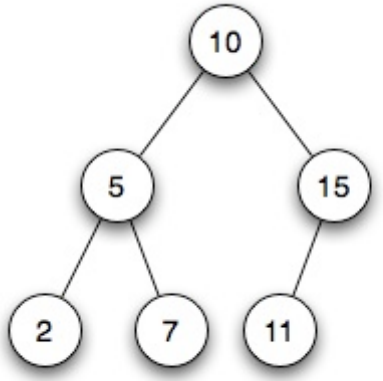


Binary Trees

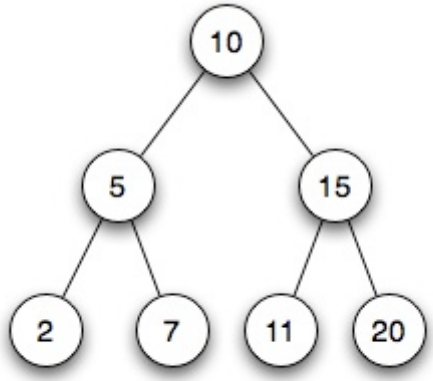
- A binary tree is a tree such that each node has at most two subtrees
- The binary tree is recursively defined as follows:
  - The empty tree is a binary tree.
  - A tree T with subtrees TL and TR is a binary tree if both TL and TR are binary trees.
  - Only trees satisfying constructed this manner are binary trees.

Fullness and Completeness

- Trees grow down from the top
  - Each new value is inserted in a new leaf node
- A ***full binary tree*** is one in which every downward path from the root to a leaf has an identical length.
- A ***complete binary tree*** is one constructed by removing from a full binary tree a number of leaves from right.



A complete binary tree



A full binary tree

General Trees

- Nodes of a general tree can have any number of subtrees
- A general tree can be represented using a binary tree

FIGURE 8.8
Binary Tree Equivalent of King William's Family Tree

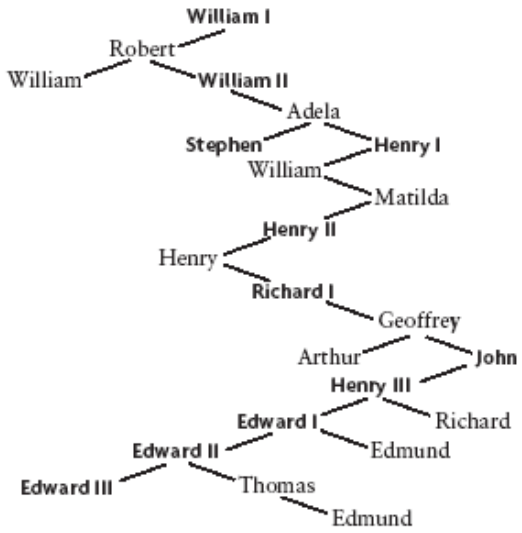
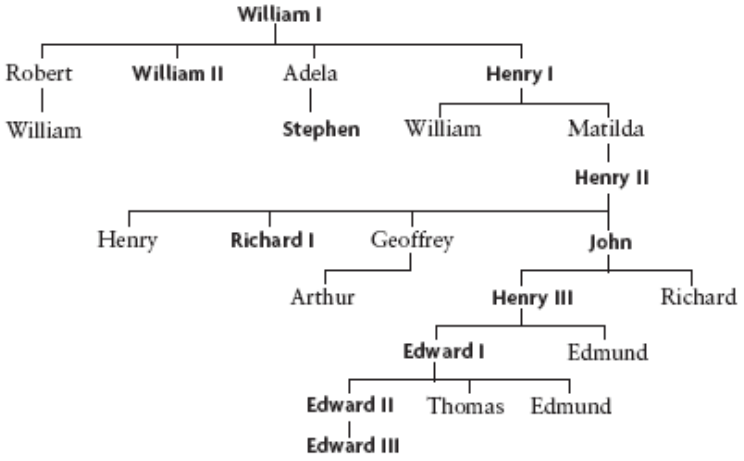


FIGURE 8.7
Family Tree for the Descendants of William I of England



Binary Tree Traversals

- A traversal of a binary tree is a walk through the tree to identify all the nodes in it in a prescribed order
  - Inorder
  - Preorder
  - Postorder

Binary Tree Traversals (continued)

- Preorder: Visit root node, traverse TL, traverse TR
- Inorder: Traverse TL, visit root node, traverse TR
- Postorder: Traverse TL, Traverse TR, visit root node

Algorithm for Preorder Traversal

1. if the tree is empty
2. Return.

else

3. Visit the root.
4. Preorder traverse the left subtree.
5. Preorder traverse the right subtree.

Algorithm for Inorder Traversal

1. if the tree is empty
2. Return.

else

3. Inorder traverse the left subtree.
4. Visit the root.
5. Inorder traverse the right subtree.

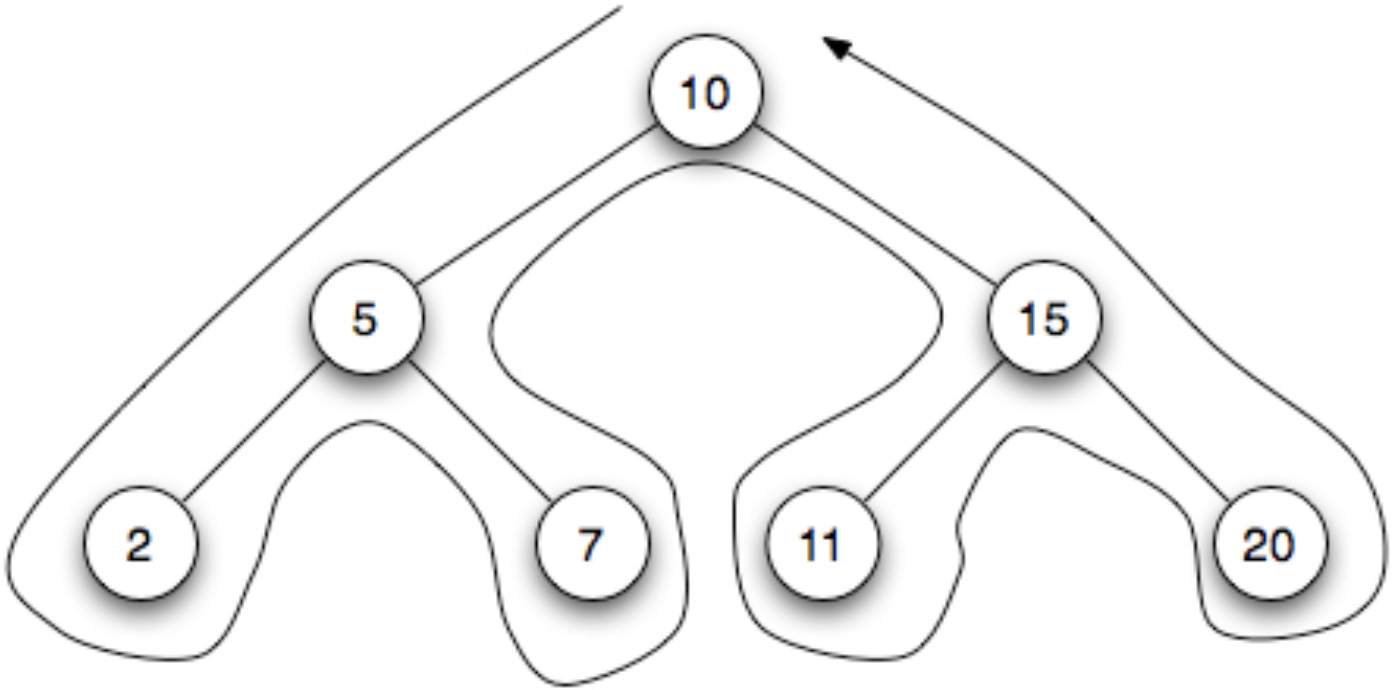
Algorithm for Postorder Traversal

1. if the tree is empty
2. Return.

else

3. Postorder traverse the left subtree.
4. Postorder traverse the right subtree.
5. Visit the root.

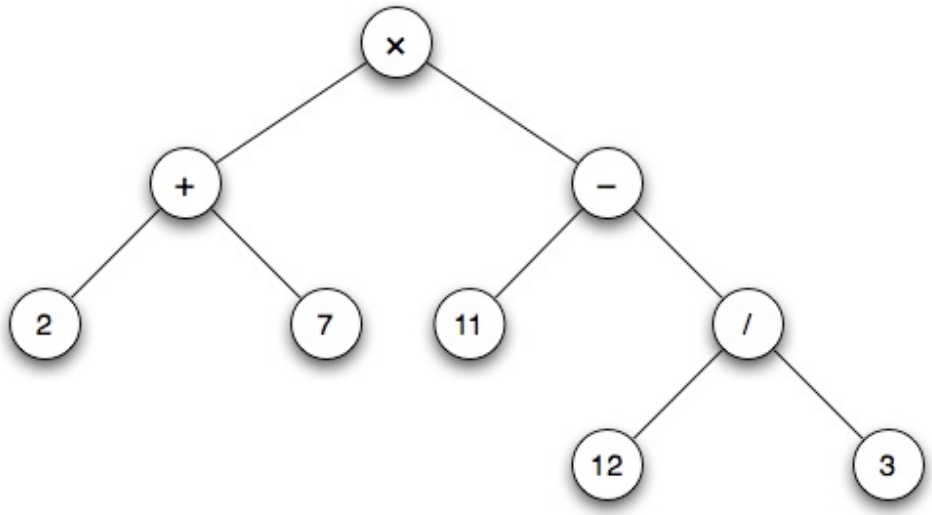
Visualizing Tree Traversals (continued)



Preorder: 10, 5, 2, 7, 15, 11, 20
Inorder: 2, 5, 7, 10, 11, 15, 20
Postorder: 2, 7, 5, 11, 20, 15, 10

Traversals of Expression Trees

- An inorder traversal of an expression tree inserts parenthesis where they belong (infix form)
- A postorder traversal of an expression tree results in postfix form



The Node<E> Class

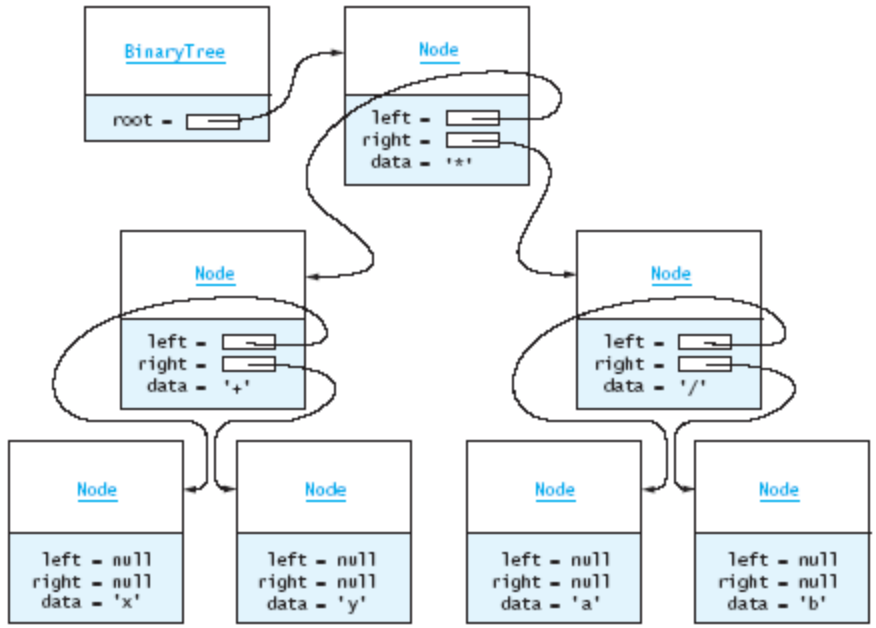
- Just as for a linked list, a node consists of a data part and links to successor nodes
- The data part is a reference to type E
- A binary tree node must have links to both its left and right subtrees

```
private class Node {
    E data;
    Node left;
    Node right;

    Node(E data) {
        this.data = data;
        left = null;
        right = null;
    }
    …
}
```

The BinaryTree<E> Class

FIGURE 8.11
Linked Representation of Expression Tree ((x + y) * (a / b))



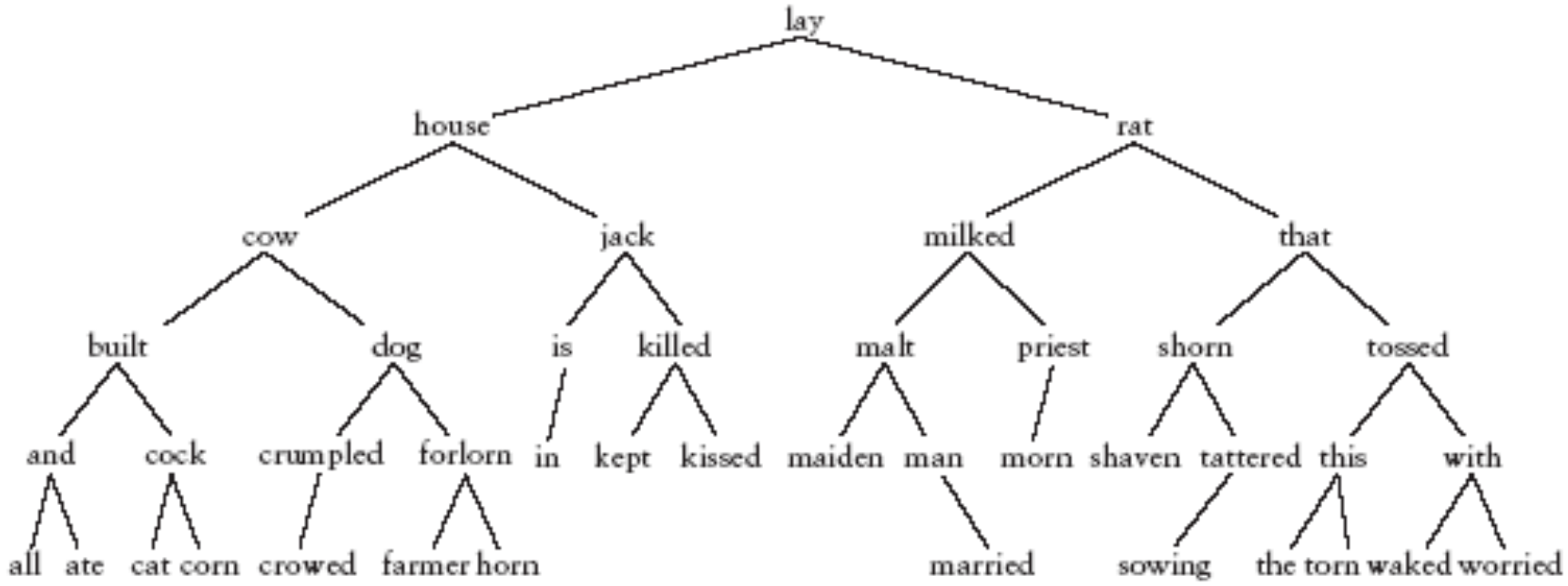
A Binary Search Tree

- A binary search tree is a binary tree with the following property:
  - Each data appearing in TL, if any, is smaller than the data in T.
  - Each data appearing in TR, if any, is greater than the data in T.

A Binary Search Tree

FIGURE 8.13

Binary Search Tree Containing All of the Words from "The House That Jack Built"



Searching for Data D in a Binary Search Tree T

if (T is null) return false;
else if (the data at T is equal to D) return true;
else if (the data at T is greater than D)
    return the result of the visit to TL;
else // the data at T is less than D
    return the result of the visit to TR;

Searching a Binary Tree

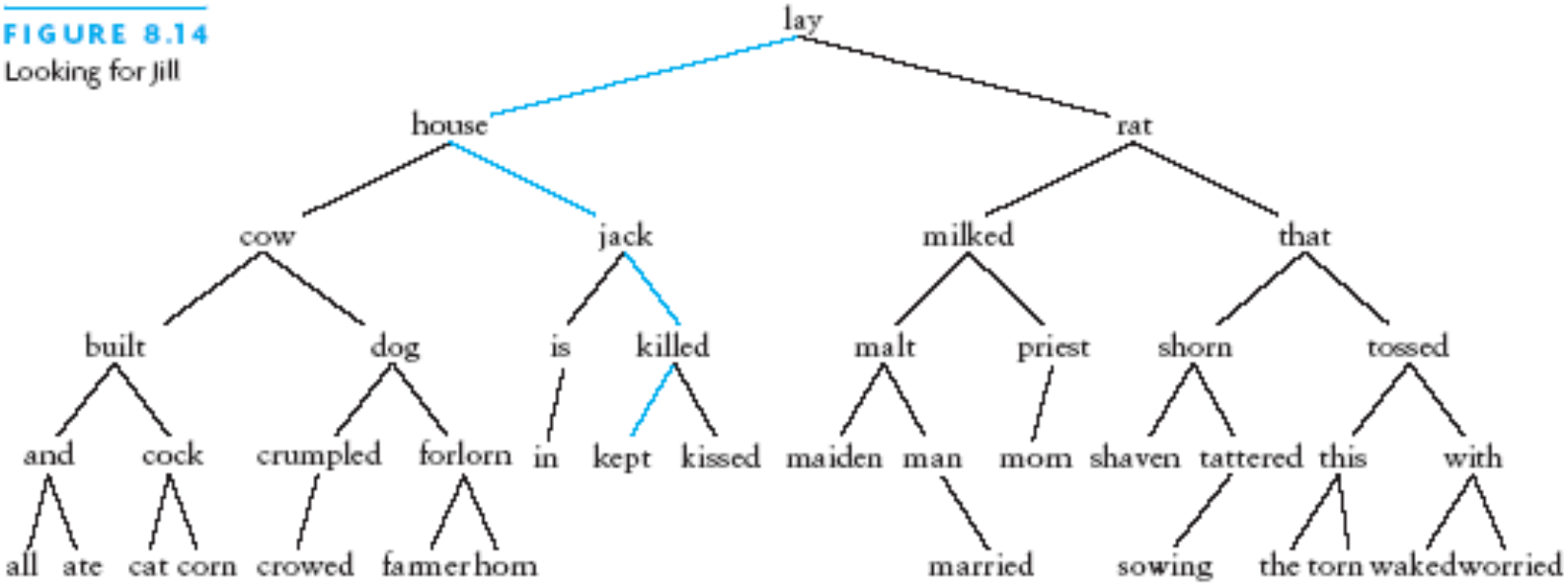


FIGURE 8.14
Looking for Jill

Insertion of a Data D into a Binary Search Tree T

- Execute the algorithm for binary search for D until the node to be visited next is null, replace that null with a node containing D as the data.

Insertion into a Binary Search Tree

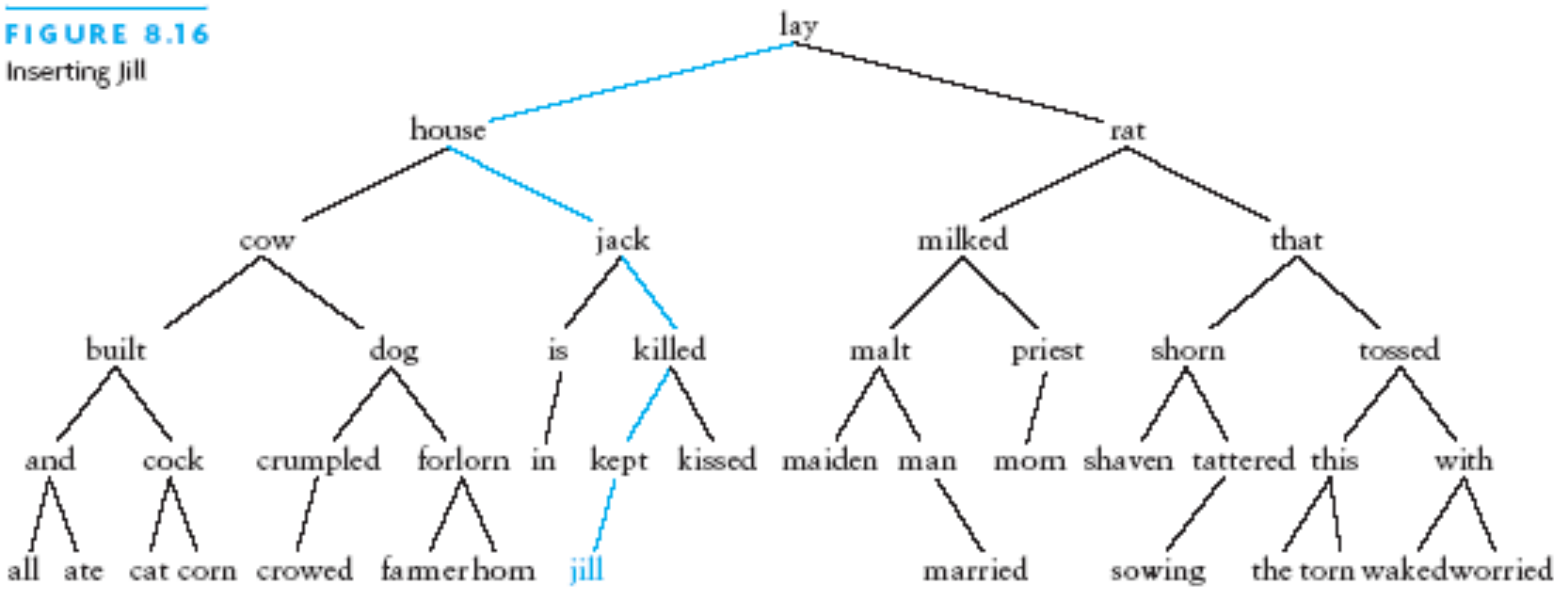


FIGURE 8.16
Inserting Jill

Deletion of D from a Binary Search Tree T

if D is found at node X, restructure the subtree rooted at D without the node containing D as follows:

if X has no children replace X with null

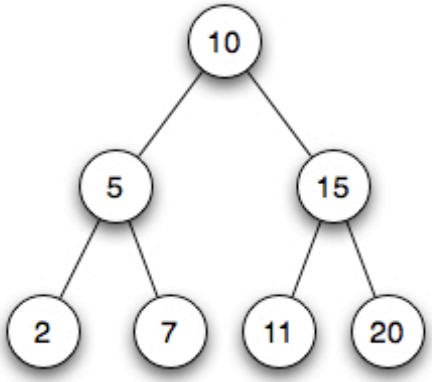
else if X has only one child, promote the child to X's position

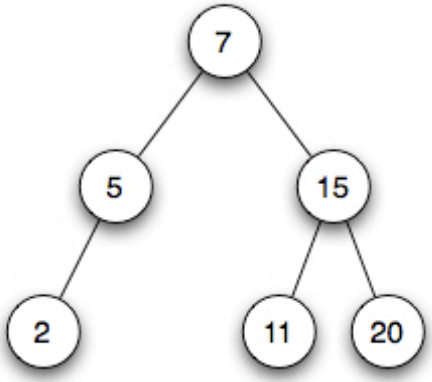
else if X's left child has no right child, promote the left child to X's position

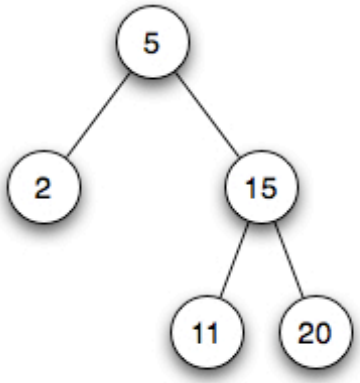
else

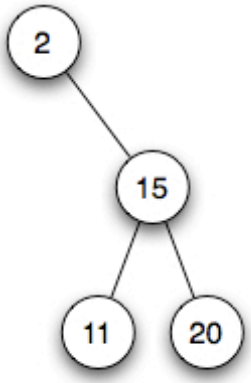
promote the rightmost node Y in the left subtree to X's position and promote the left child of Y to Y's position

Example: Remove 10, 7, 5, Then 2









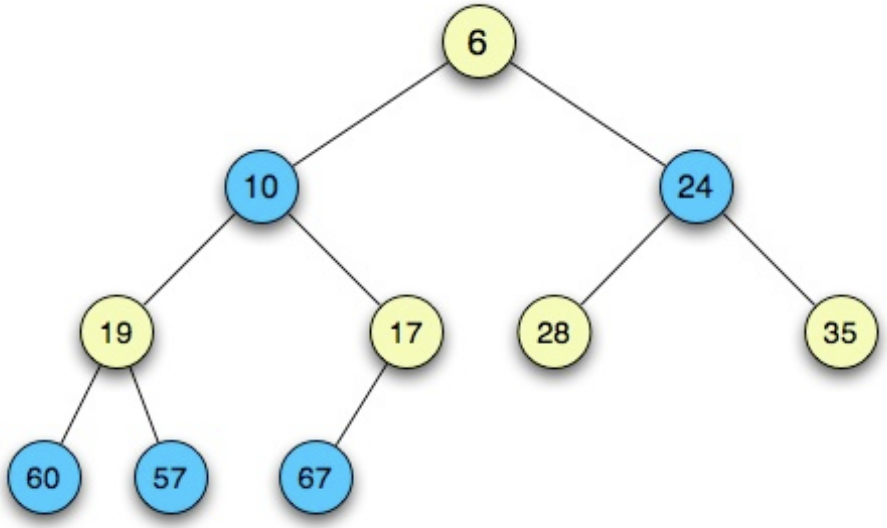
Interface SearchTree

Methods	Action
boolean add(E item)	Adds _item_ to the tree in an appropriate location. Returns whether successful.
boolean contains(E item)	Returns whether _item_ is in the tree.
E find(E item)	Returns a reference to _item_ if it is in the tree; otherwise, returns null.
E delete(E item)	Attempts to find and remove _item_. If successful, returns a reference to the deleted _item_. Otherwise, returns null.
boolean remove(E item)	Attempts to find and remove _item_. Returns whether the operation was successful.

Heaps and Priority Queues

Heaps and Priority Queues

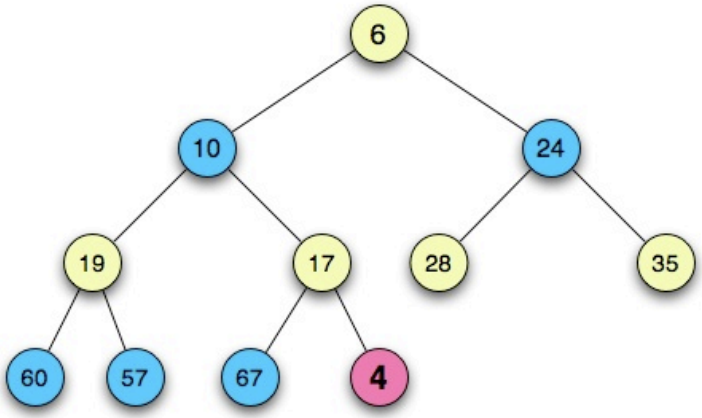
- Heap is a complete binary tree such that the value in each node is less than all values in its two subtrees

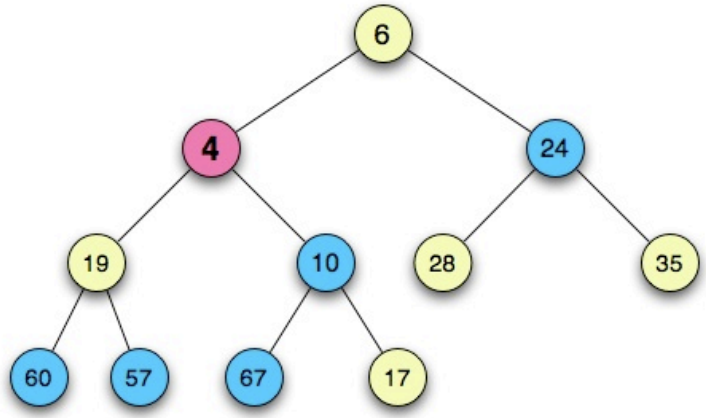


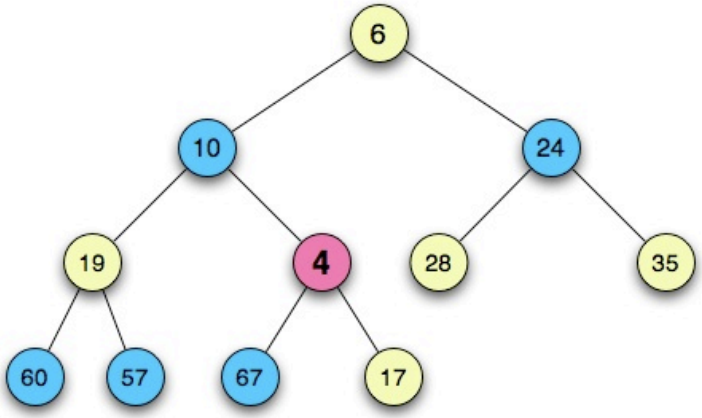
Inserting an Item into a Heap

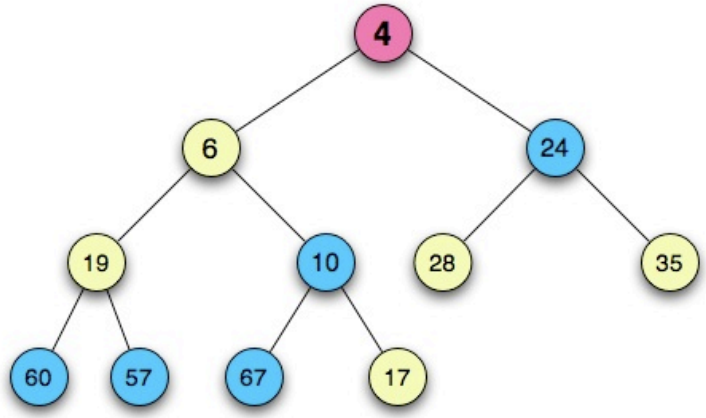
- Place the item at the next leaf position. It is the left child of the leftmost leaf is the tree is full.
- Work from that node towards the root by swapping a node and its parent that are violating the heap property.

Example: Inserting 4





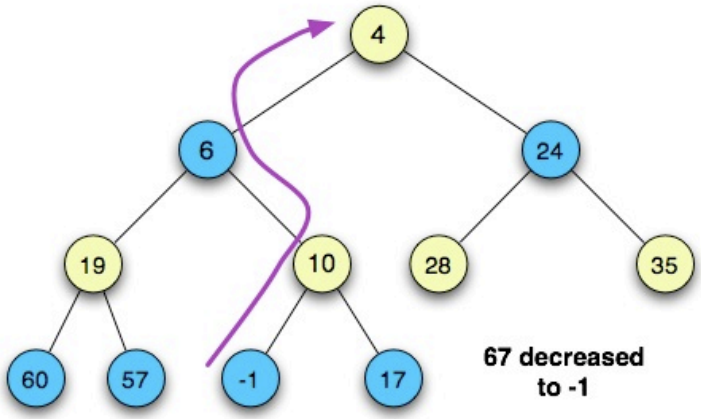




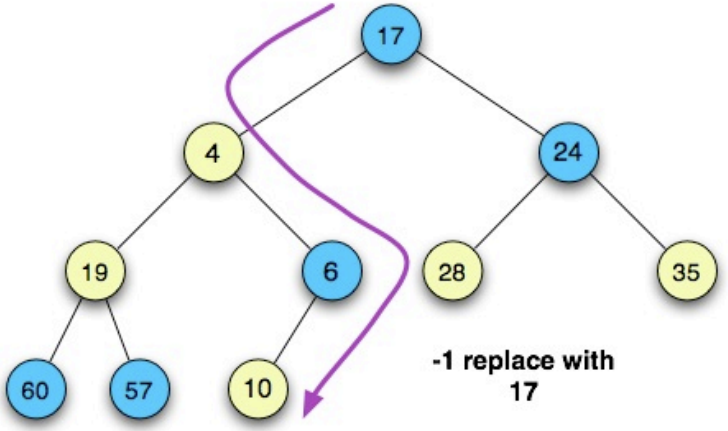
Removing an Item from a Heap

- Decrease the value of the item to a special value that is smaller than any other value appearing in the heap.
- Execute the resolution algorithm for insertion from that node.
- Promote the last leaf to the root.
- Starting from the root, recursive resolve violation as follows:
  - If the current node and the children do not violate the property stop.
  - Swap the current node with the child having the smaller value than the other sibling.
  - Move to the node with which the swap took place.

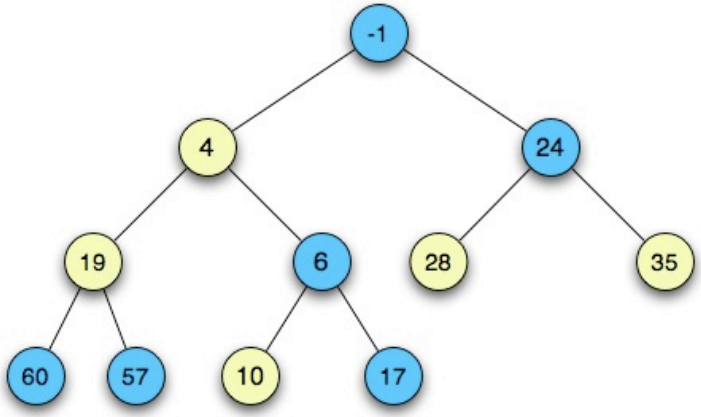
Example: Removing 67

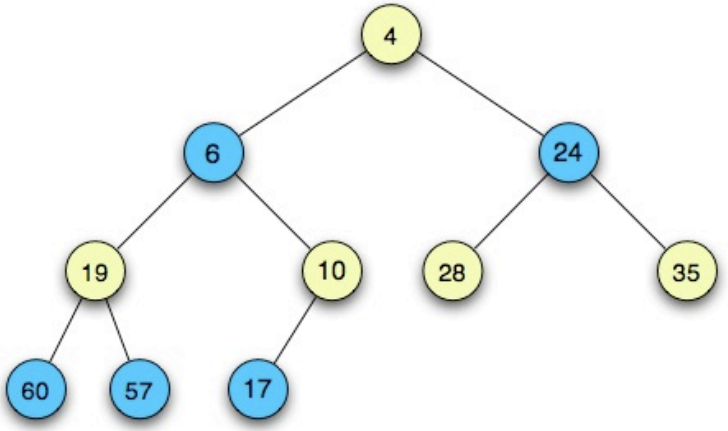


67 decreased to -1



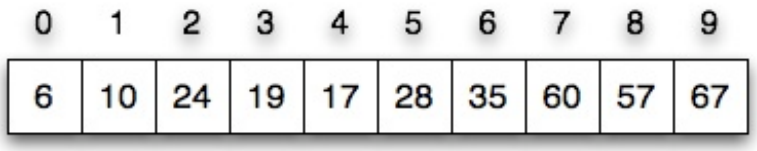
-1 replace with 17

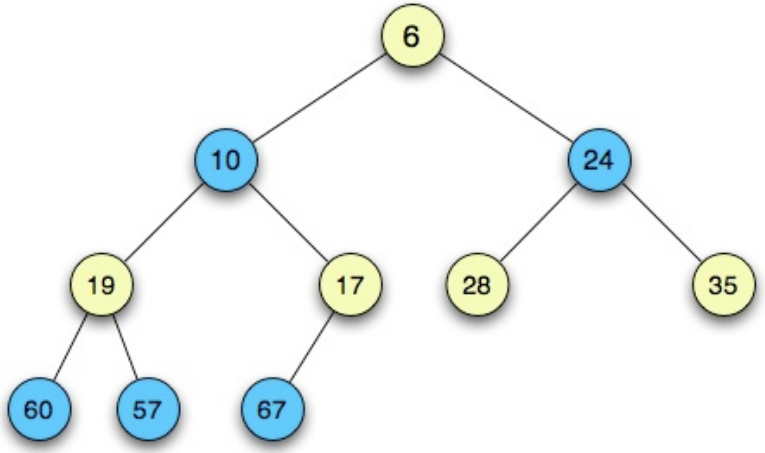




Implementing a Heap as an Array

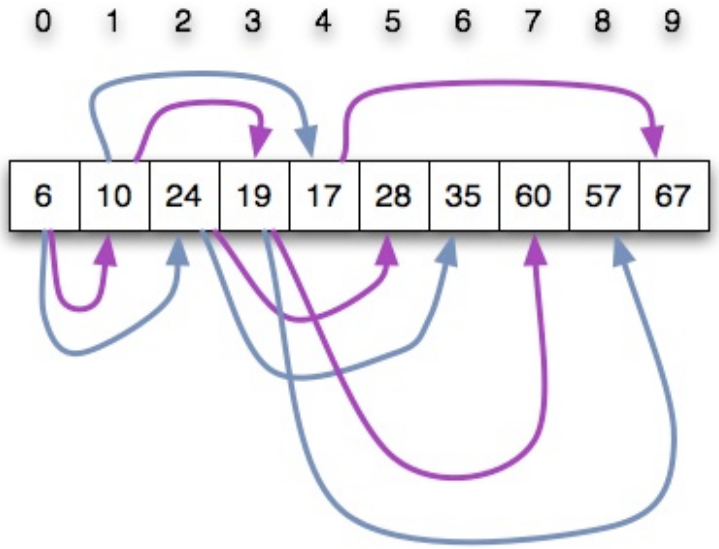
- Because a heap is a complete binary tree, it can be implemented efficiently using an array instead of a linked data structure
- Storing goes in the increasing order of the distance from the root and from left to right.

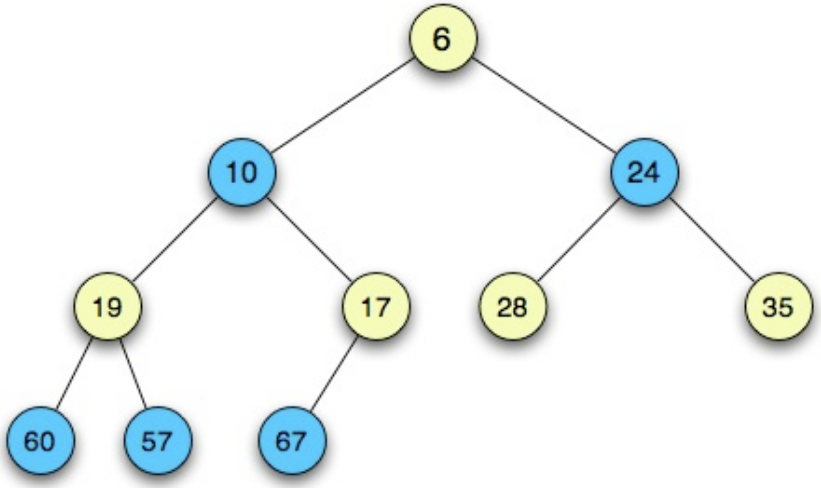




Relations Between Parent and Children

- If the parent has index P, then the left and right children have indices 2P+1 and 2P+2, respectively.
- If a child has index Q, then the parent has index (Q-1)/2, where / denotes the integer division.





Priority Queues

- A priority queue is a special kind of queue in which each object has an priority value
- The element with the smallest value is served the next and removed from the queue.
- The priority value of any node can be increased or decreased.
- This can be implemented using a heap.
  - Removal, insertion, and value modification require O ($\log_2 n$) time

The PriorityQueue Class

- Java provides a PriorityQueue<E> class that implements the Queue<E> interface given in Chapter 6.
- Peek, poll, and remove methods return the smallest item in the queue rather than the oldest item in the queue.

Methods	Behavior
E element()	Returns the object at the top of the queue without removing it. . If the queue is empty, returns NoSuchElement exception.
E peek()	Returns the object at the top of the queue without removing it. . If the queue is empty, returns null.
E remove()	Returns the object at the top of the queu and removes it. If the queue is empty, returns NoSuchElement exception.
E poll()	Returns the object at the top of the queue and removes it. If the queue is empty, returns null.
boolean offer(E obj)	Appends item obj at the end of queue. Returns **true** if successful false otherwise.

Interface Comparator<E>

- This interface requires only one method:
  - int compare(E o1, E o2), which returns the result of comparison
- For a method that requires, a Comparator<E> object, you need to supply an object of class that implements Comparator<E>
  - For example, a class MyC implements Comparator<E> (and thus has a method compare(E o1, E o2) that returns int), then
    - You can supply: new MyC()

Design of a KWPriorityQueue Class

Data Field	Attribute
ArrayList<E> theData	Array list to hold the data
Comparator<E> comparator	An object that implements the Comparator<E> interface, may be null
Method	**Action**
KWPriorityQueue ()	Constructor that provides an empty queue.
KWPriorityQueue(int cap, Comparator<E> comp)	Constructor that provides a queue with capacity cap and with comparator comp.
private int compare(E left, E right)	Returns -1 if left is smaller than right, 0 if they are equal, and 1 otherwise; if comparator is null, default to the compareTo method of E
private void swap(int i, int j)	Swaps object at index i and object at index j.

Huffman Trees

- A Huffman tree can be implemented using a binary tree and a PriorityQueue
- A straight binary encoding of an alphabet assigns a unique binary number to each symbol in the alphabet
  - Unicode for example
- The message “go eagles” requires 144 bits in Unicode but only 38 using Huffman coding

Huffman Trees (continued)

TABLE 8.8
Frequency of Letters in English Text

Symbol	Frequency	Symbol	Frequency	Symbol	Frequency
_	186	h	47	g	15
e	103	d	32	p	15
t	80	l	32	b	13
a	64	u	23	v	8
o	63	c	22	k	5
i	57	f	21	j	1
n	57	m	20	q	1
s	51	w	18	x	1
r	48	y	16	z	1

Construction of a Huffman Tree

- Input: characters c[0], …, c[m-1] and their frequency values v[0], …, v[m-1]
- Initialization: construct binary trees T[0], …, T[m-1], where each node T[i] has v[i] as the data
- Iteration: while there is more than one tree remaining do:
  - Find two trees, T1 and T2, whose data values are the smallest of all remaining trees
  - Construct a new tree T3, whose data value is the some of the data values of T1 and T2 and whose subtrees are T1 and T2.
  - Remove T1 and T2 from the collection and add T3.

Huffman Trees (continued)

FIGURE 8.35

Huffman Tree Based on Frequency of Letters in English Text