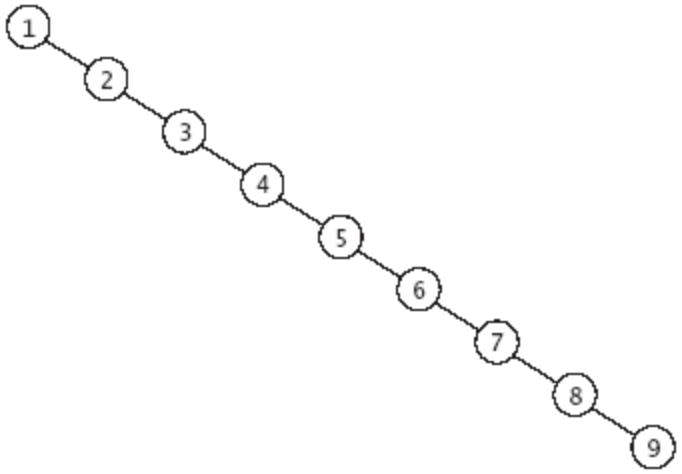
Self-Balancing Search Trees

Chapter 4

Why Balance is Important

- Searches into an unbalanced search tree could be O(n) at worst case

FIGURE 11.1
Very Unbalanced
Binary Search Tree



Rotation

- To achieve self-adjusting capability, we need an operation on a binary tree that will change the relative heights of left and right subtrees but preserve the binary search tree property
- Algorithm for rotation
  - Remember value of root.left (temp = root.left)
  - Set root.left to value of temp.right
  - Set temp.right to root
  - Set root to temp

Rotation (continued)

FIGURE 11.3
Unbalanced Tree Before Rotation

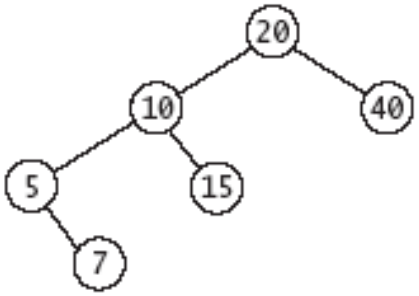


FIGURE 11.4
Right Rotation

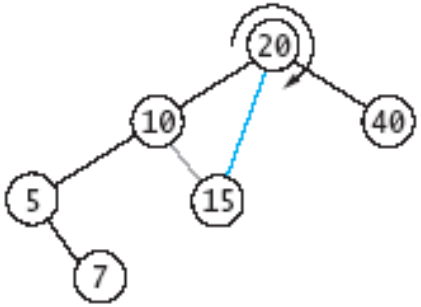
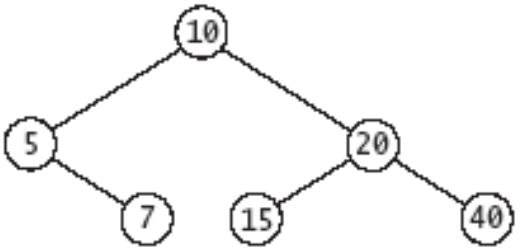
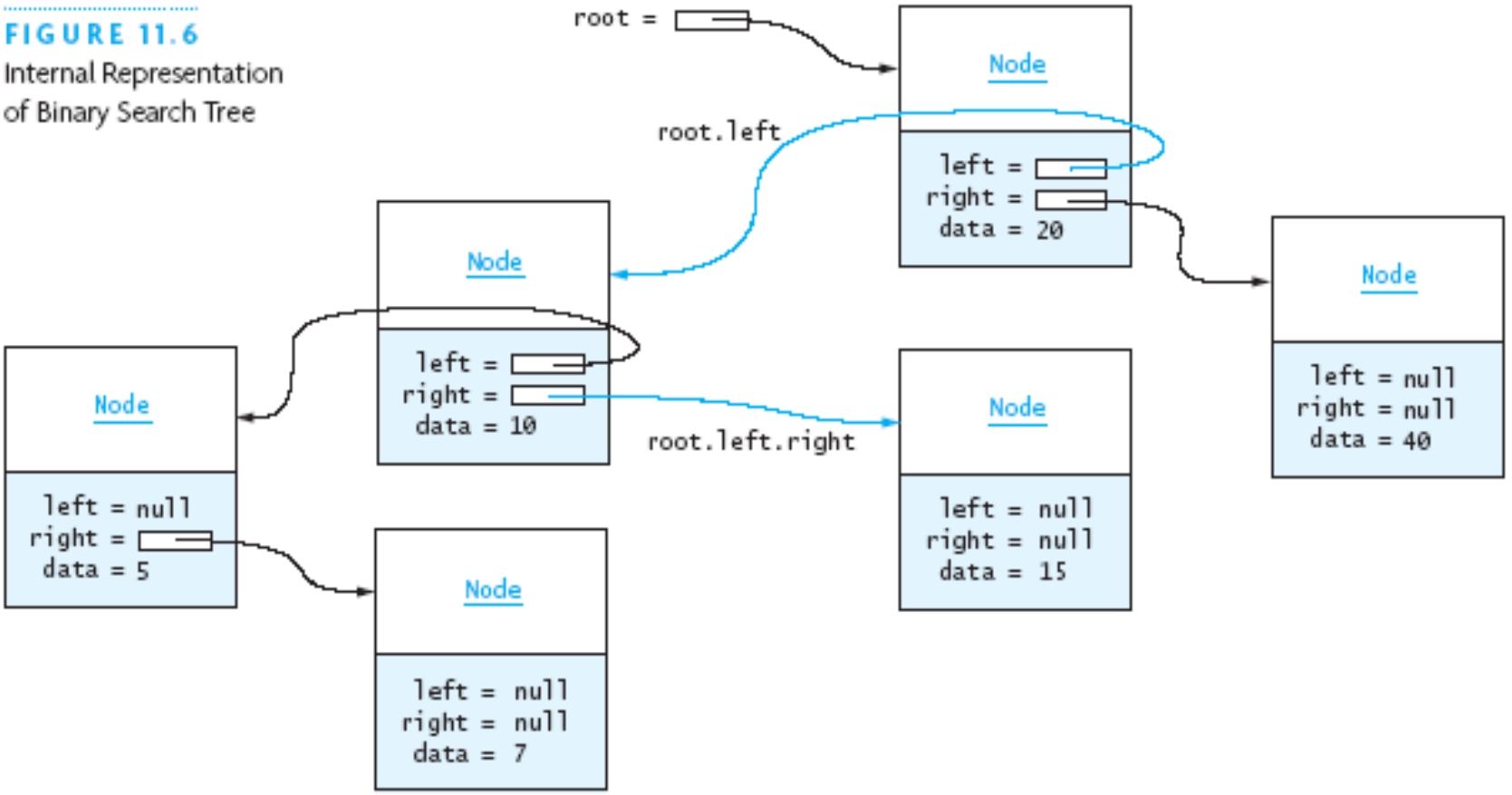


FIGURE 11.5
More Balanced Tree After Rotation

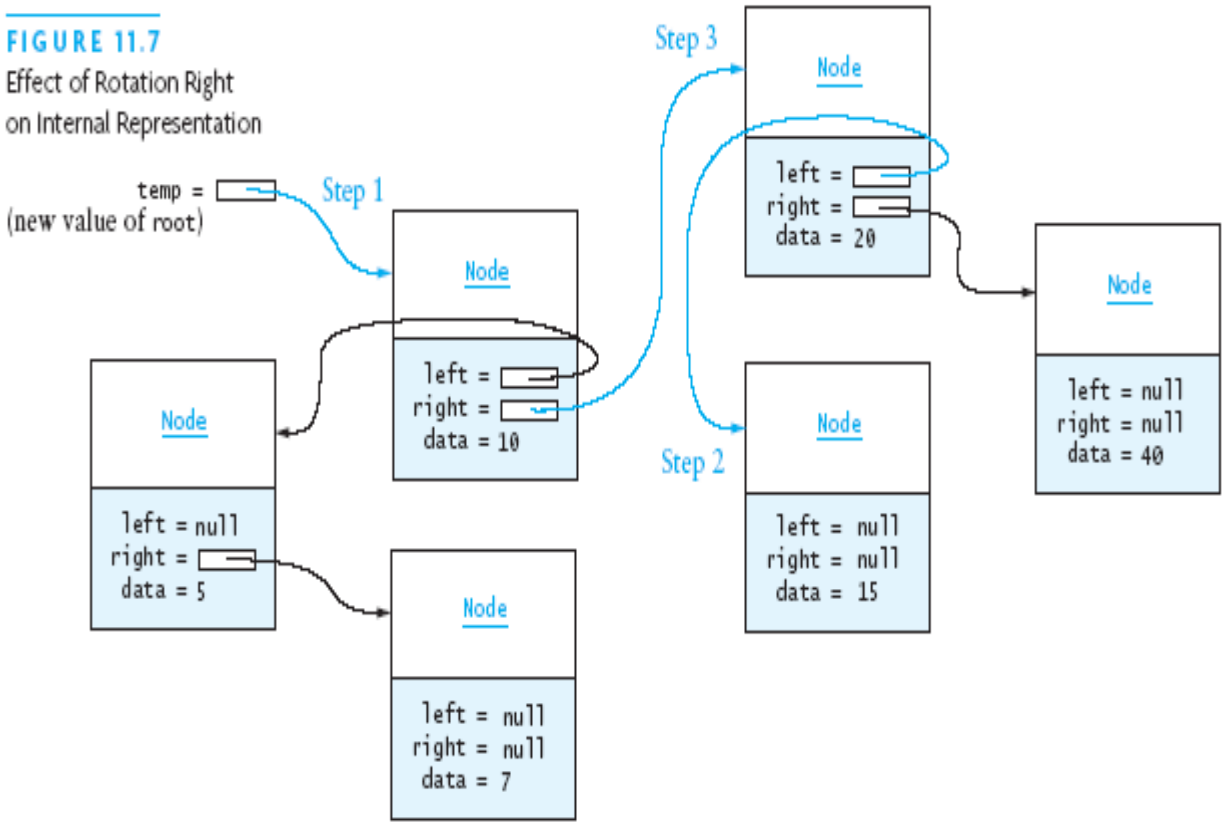


Rotation (continued)

FIGURE 11.6
Internal Representation of Binary Search Tree



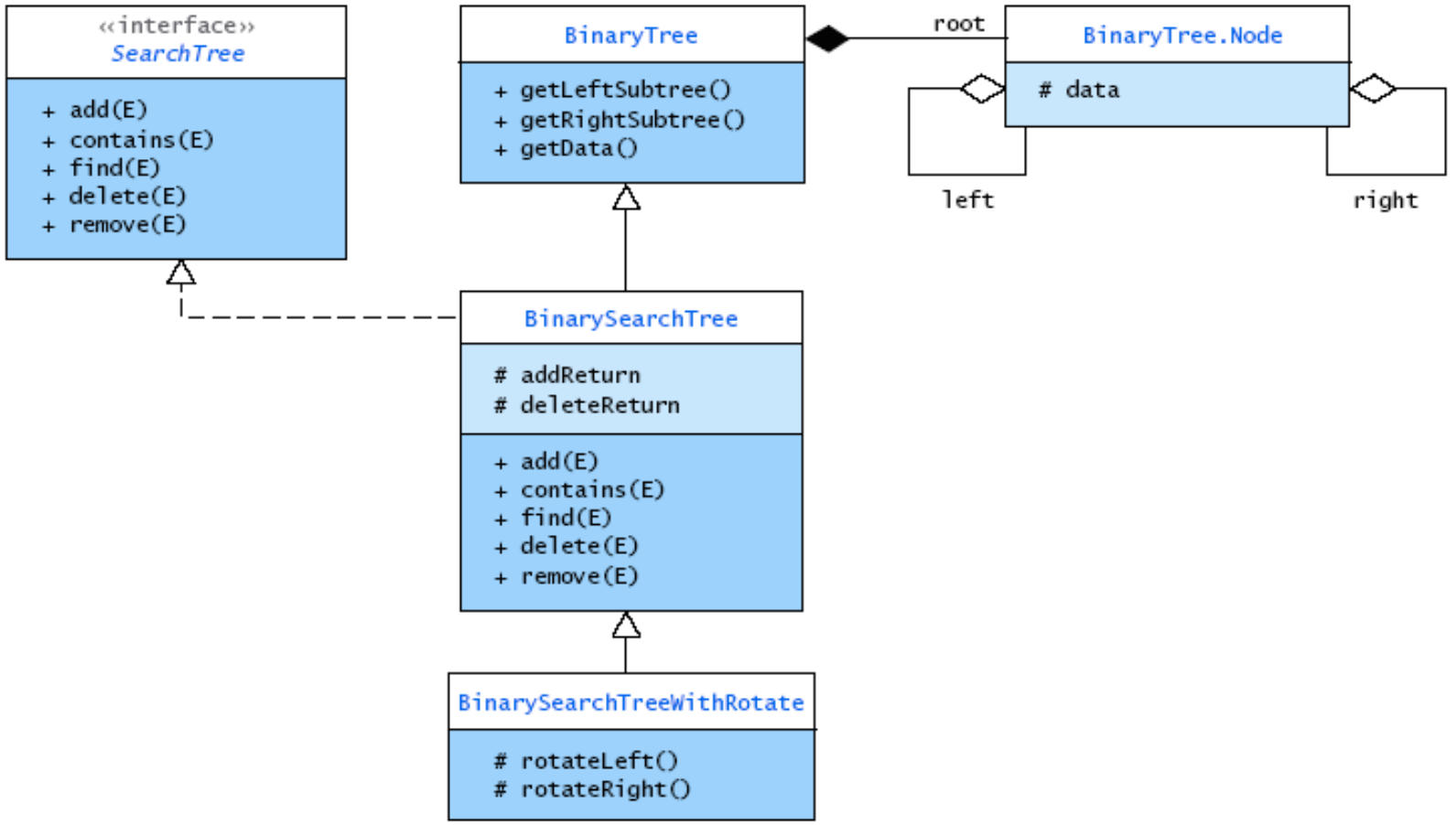
Rotation (continued)



Implementing Rotation

FIGURE 11.8

UML Diagram of BinarySearchTreeWithRotate



AVL Tree

- As items are added to or removed from the tree, the balance or each subtree from the insertion or removal point up to the root is updated
- Rotation is used to bring a tree back into balance
- The height of a tree is the number of nodes in the longest path from the root to a leaf node

Balancing a Left-Left Tree

- The heights of the left and right subtrees are unimportant; only the relative difference matters when balancing
- A left-left tree is a tree in which the root and the left subtree of the root are both left-heavy
- Right rotations are required

FIGURE 11.9
Left-Heavy Tree

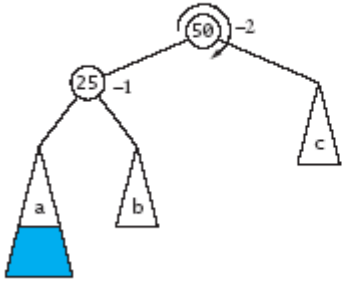
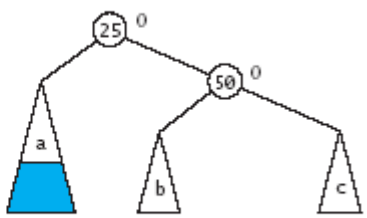


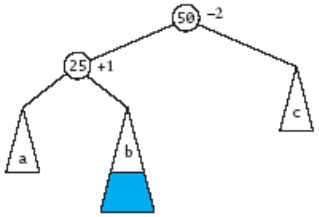
FIGURE 11.10
Left-Heavy Tree After Rotation Right



Balancing a Left-Right Tree

- Root is left-heavy but the left subtree of the root is right-heavy
- A simple right rotation cannot fix this
- Need both left and right rotations

FIGURE 11.11
Left-Right Tree



Balance 50 = $(k - (k + 2))$

Balance 25 = $((k + 1) - k)$

FIGURE 11.12
Insertion into b_L

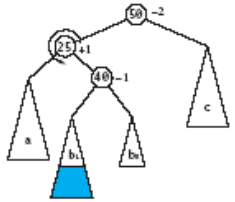


FIGURE 11.13
Left Subtree After Rotate Left

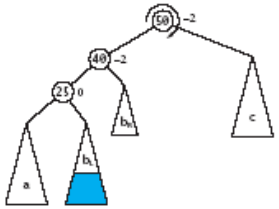


FIGURE 11.14
Tree After Rotate Right

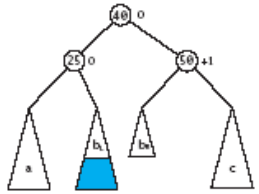


FIGURE 11.15
Insertion into b_R

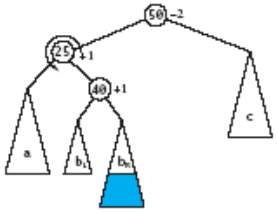


FIGURE 11.16
Left Subtree After Rotate Left

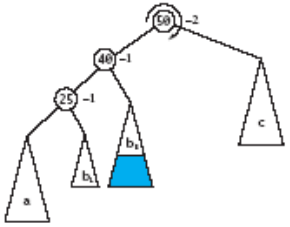
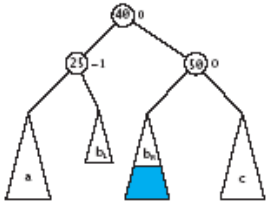


FIGURE 11.17
Tree After Rotate Right



Four Kinds of Critically Unbalanced Trees

- Left-Left (parent balance is -2, left child balance is -1)
  - Rotate right around parent
- Left-Right (parent balance -2, left child balance +1)
  - Rotate left around child
  - Rotate right around parent
- Right-Right (parent balance +2, right child balance +1)
  - Rotate left around parent
- Right-Left (parent balance +2, right child balance -1)
  - Rotate right around child
  - Rotate left around parent

Implementing an AVL Tree

FIGURE 11.18
UML Class Diagram of AVLTree