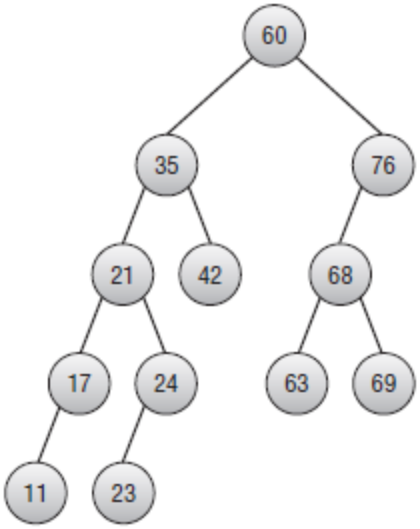
Trees, Part 1

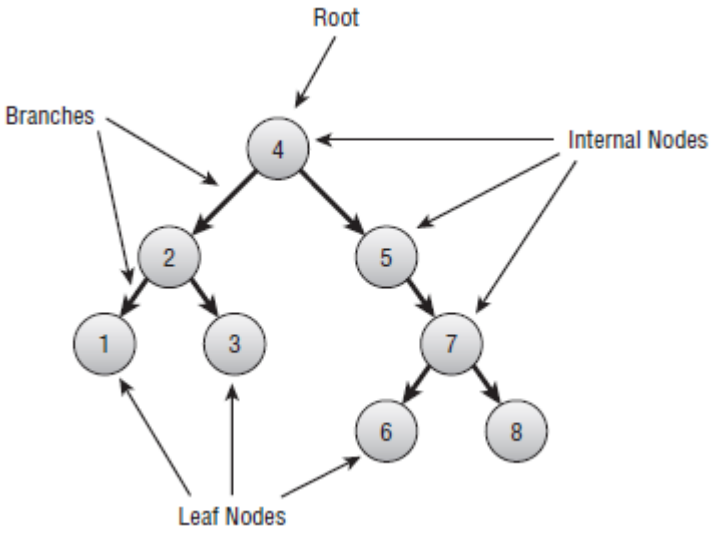


Agenda

- Tree Terminology
- Binary Trees Properties
- Tree Representations
- Tree Traversal
- Sorted Trees
- Lowest Common Ancestors
- Summary
- Exercises

Tree Terminology

- Node
  - Internal node
  - Leaf (external node)
- Branch (link, edge)
- Parent/child
- Descendant/ancestor
- Sibling
- Root

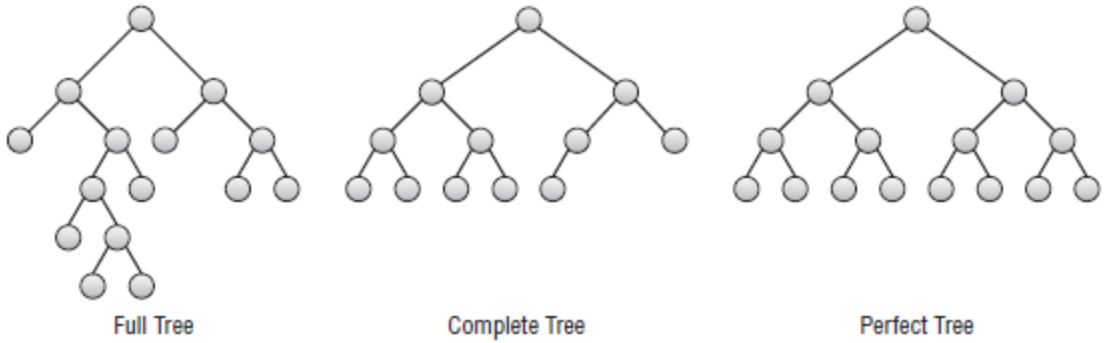


Tree Terminology (continued)

- Degree
  - Binary tree
- Level (depth)
- Height
- Subtree
- Ordered/unordered tree
- First (least) common ancestor

Tree Terminology (continued again)

- Full tree – Every node has zero or as manym children as possible
- Complete tree – Every level is full, except possibly the last level where every node is pushed to the left
- Perfect tree – Full and leaves are on the same level



Binary Trees Properties

- Let:
  - B = # branches
  - N = # nodes
  - L = # leaf nodes
  - I = # internal nodes
  - H = tree height
  - B = N – 1
  - M = # missing branches (a branch could be added)

Binary Trees Properties, Part 2

- In a perfect binary tree, $N = 2H+1 – 1$
- In a perfect binary tree, $H = \log2(N + 1) – 1$
- In a perfect binary tree, $L = 2H$
- $M = N + 1$
- If a binary tree has N_0 leaf nodes and N_2 nodes with degree 2, $N_0 = N_2 + 1$
- In other words, there is one more leaf node than nodes with degree 2.

Binary Trees Properties, Part 3

- In a perfect binary tree:

  $I = N - L$

  $= (2H+1 - 1) - 2H$

  $= (2H+1 - 2H) - 1$

  $= 2H \times (2 - 1) - 1$

  $= 2H - 1.$

  This means almost exactly half of the nodes are leaves and almost exactly half are internal nodes. More precisely, $I = L - 1$.

Tree Representations

- Use a Node class

Binary Tree Representation

```
Class BinaryNode
    String: Name
    BinaryNode: LeftChild
    BinaryNode: RightChild

    Constructor(String: name)
        Name = name
    End Constructor
End Class
```

Building a Binary Tree

```
BinaryNode: root = New BinaryNode("4")
BinaryNode: node1 = New BinaryNode("1")
BinaryNode: node2 = New BinaryNode("2")
BinaryNode: node3 = New BinaryNode("3")
BinaryNode: node5 = New BinaryNode("5")
BinaryNode: node6 = New BinaryNode("6")
BinaryNode: node7 = New BinaryNode("7")
BinaryNode: node8 = New BinaryNode("8")

root.LeftChild = node2
root.RightChild = node5
node2.LeftChild = node1
node2.RightChild = node3
node5.RightChild = node7
node7.LeftChild = node6
node7.RightChild = node8
```

N-ary Tree Representation

```
Class TreeNode
    String: Name
    List Of TreeNode: Children

    Constructor(String: name)
        Name = name
    End Constructor
End Class
```

N-ary Tree With Data

```
Class TreeNode
    String: Name
    List Of TreeNode: Children
    List Of Data: ChildData

    Constructor(String: name)
        Name = name
    End Constructor
End Class
```

Branch Data

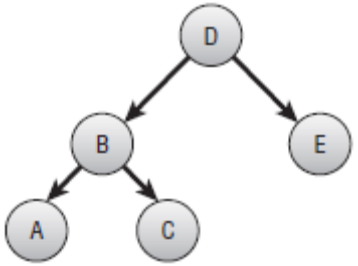
```
Class TreeNode
    String: Name
    List Of Branch: Branches

    Constructor(String: name)
        Name = name
    End Constructor
End Class

Class Branch
    Data: BranchData
    TreeNode: Child
End Class
```

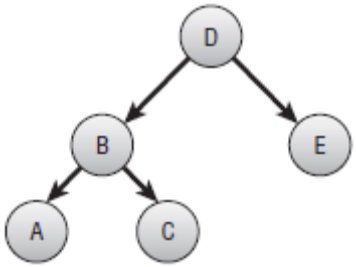
Tree Traversal

- Traversals visit a tree's nodes in different orders



Preorder

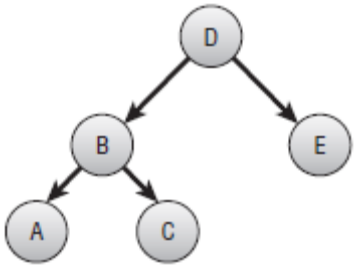
- Visit the node, then recursively visit the children



- D B A C E

Inorder (Symmetric)

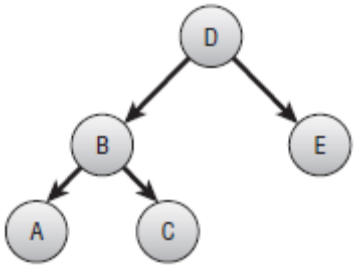
- Recursively visit the left child, then the node, then recursively visit the right child



- A B C D E

Postorder

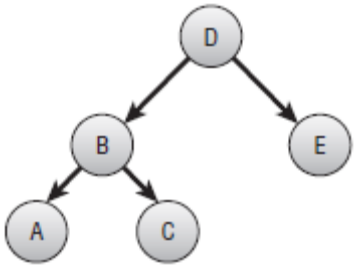
- Recursively visit the children, then visit the node



- A C B E D

Depth-First

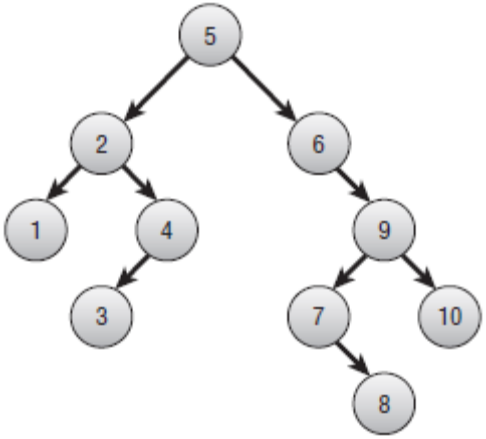
- Visit the nodes on each level of the tree before visiting any nodes on the next level



- D B E A C

Sorted Trees

- Nodes are arranged so an inorder traversal visits them in sorted order



Adding Nodes

- If the new value is smaller than the current node's value, move down the left branch
- If the new value is larger than the current node's value, move down the right branch
- When there is no branch, add a new node

Finding Nodes

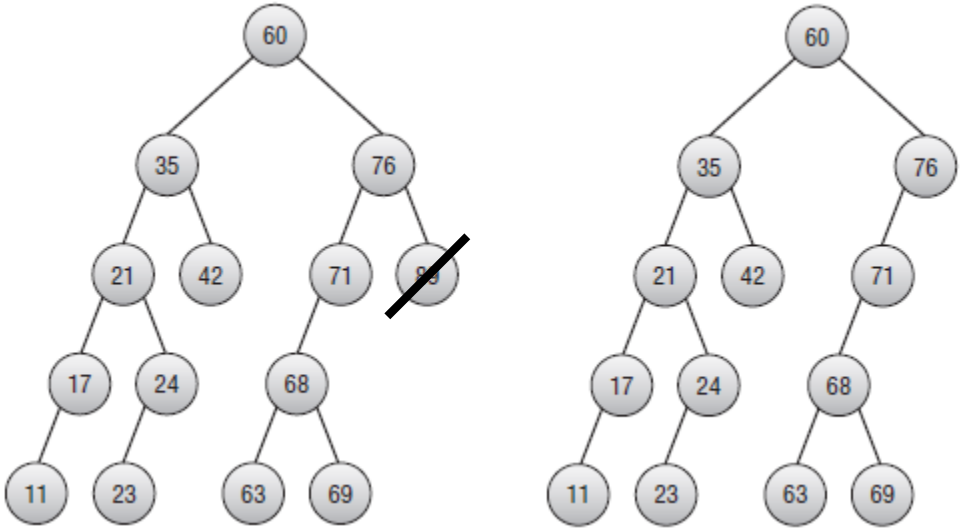
- If the target value is smaller than the current node, move down the left branch
- If the target value is larger than the current node, move down the right branch
- If you drop off the tree, the value isn't present

Deleting Nodes

- How you delete a node depends on its position in the tree

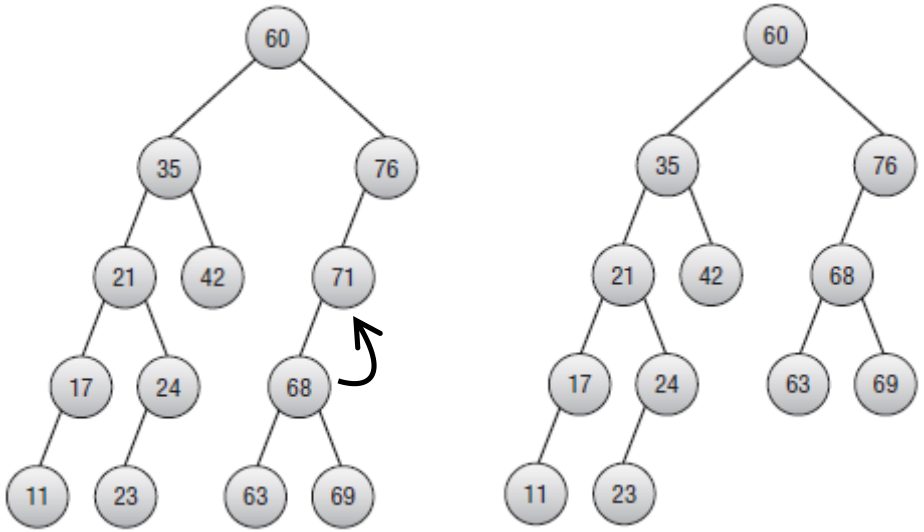
Deleting Leaf Nodes

- If the target is a leaf, delete it



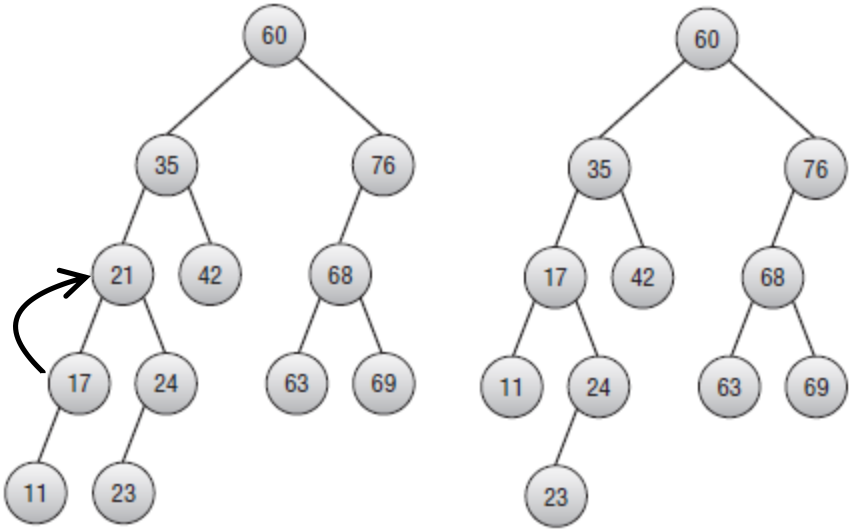
Deleting Nodes With One Child

- If the target has one child, replace it with its child



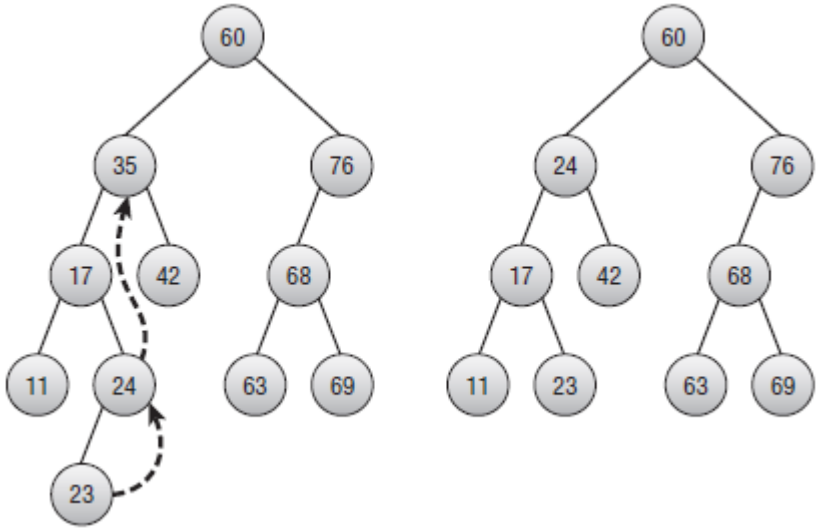
Deleting Nodes With Two Children

- If the target's left child has no right child, replace the target with its left child

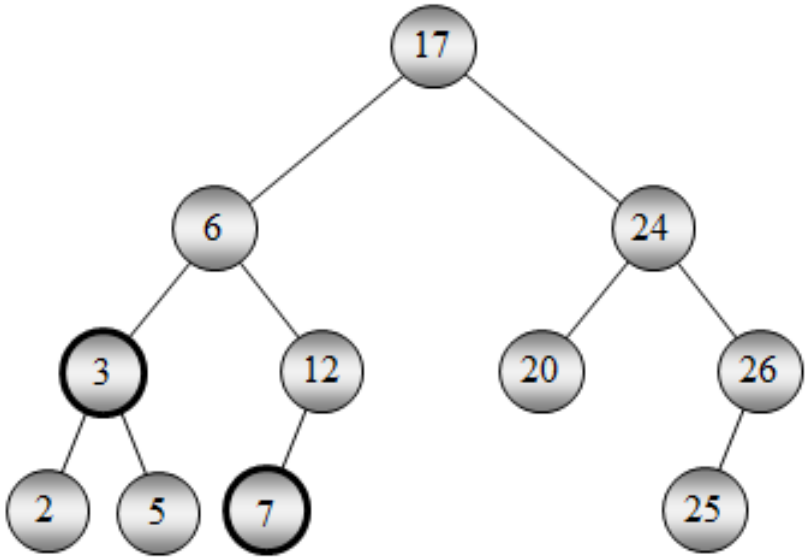


Deleting Nodes With Two Children (continued)

- If the target has two children and its left child has a right child:
  - Replace the target with its rightmost descendant to the left
  - If that node has a left child, replace it with its left child



Lowest Common Ancestors



Lowest Common Ancestors

- Sorted Trees
- Parent Pointers
- Parents and Depths
- General Trees
- Euler Tours
- All Pairs

Summary

- Tree Terminology
- Binary Trees Properties
- Tree Representations
- Tree Traversal
- Sorted Trees
  - Adding Nodes
  - Finding Nodes
  - Deleting Nodes
- Lowest Common Ancestors

Exercises

- Chapter 10 Exercises 1 – 4, 7 – 13.
- Bonus: Chapter 10 Exercise 5.
- Super Bonus: Chapter 10 Exercises 6, 14, 15.
- Read *Essential Algorithms, 2e* Chapter 10 pages 317 – 348. (The rest of Chapter 10.)