CHAPTER 6
PART 3

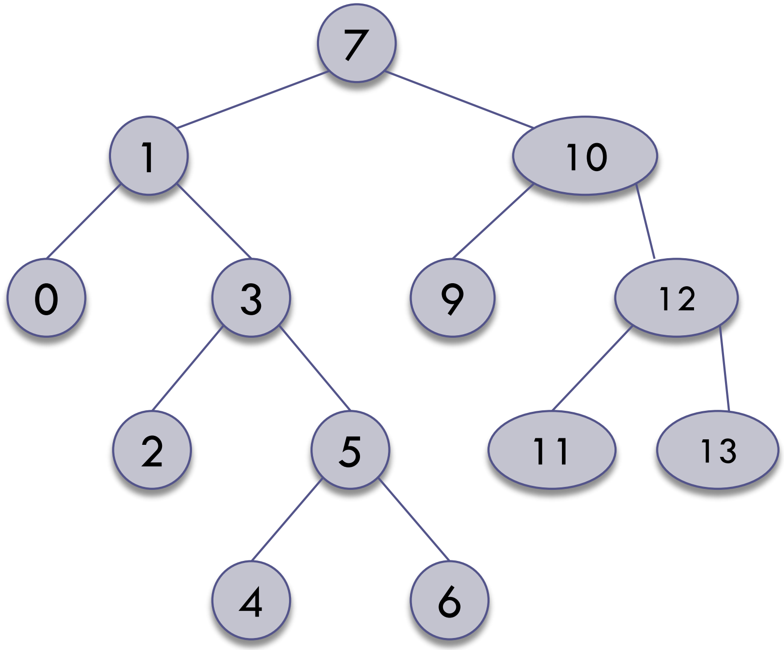
Full/Perfect/Complete BT, General Tree, Tree Traversal

Key Topics

- Full, Perfect, and Complete Binary Trees
- General Tree
- Tree Traversals
  - In-Order
  - Pre-Order
  - Post-Order

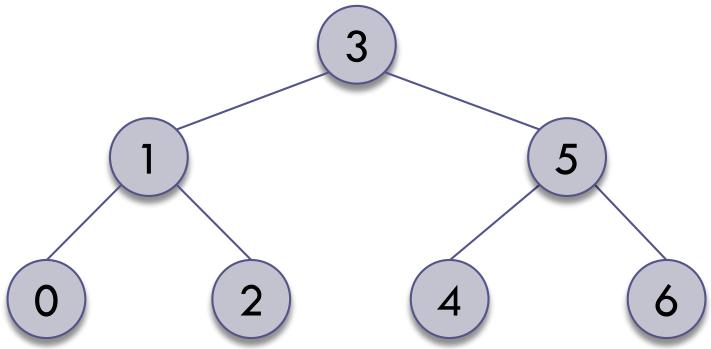
Full, Perfect, and Complete Binary Trees

- **Full** binary tree
  - binary tree
  - all nodes have either 2 children or 0 children (the leaf nodes)



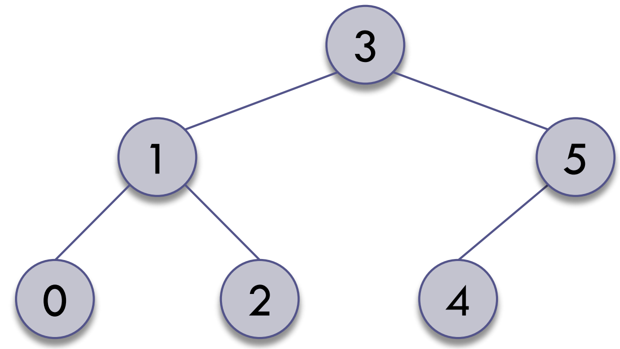
Full, Perfect, and Complete Binary Trees (cont.)

- **Perfect** *binary tree*
  - Full binary tree of height n
  - with exactly $2^n - 1$ nodes
- In this case,
  - $n = 3$ and $2^n - 1 = 7$



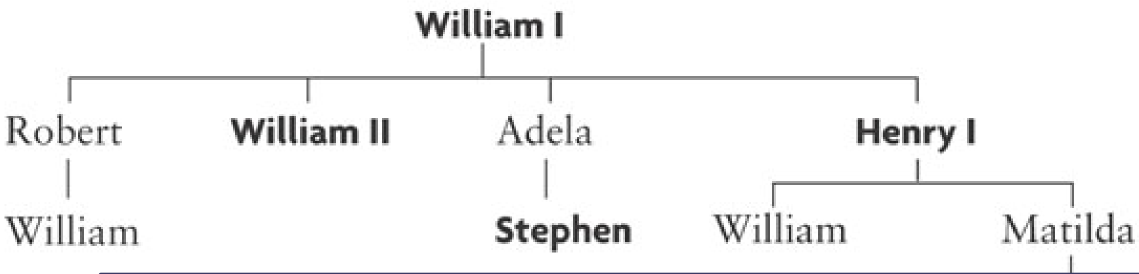
Full, Perfect, and Complete Binary Trees (cont.)

- **Complete** *binary tree*
  - Perfect binary tree through level (n - 1)
  - With some extra leaf nodes at level n (the tree height)
    - All located toward the LEFT



General Trees

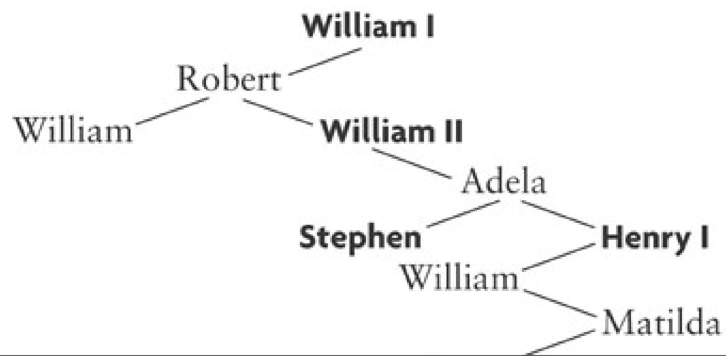
- Not discuss general trees in this chapter
- Nodes can have any number of subtrees



See more in the textbook

General Trees (cont.)

- Can represent a general tree using a binary tree
- The **left branch** of a node is the **oldest (leftmost) child**,
- Each **right branch** is connected to the **next younger sibling** (if any)



See more in the textbook

Tree Traversals

Section 6.2

Tree Traversals

- Process to
  - Walk through the tree in a prescribed order and
  - Visit the nodes as they are encountered
- Used to determine the nodes of a tree and their relationship
- Three common kinds of tree traversal
  - Inorder
  - Preorder
  - Postorder

Tree Traversals (cont.)

- **Pre**order: visit root node, traverse T_L, traverse T_R
- **In**order: traverse T_L, visit root node, traverse T_R
- **Post**order: traverse T_L, traverse T_R, 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.

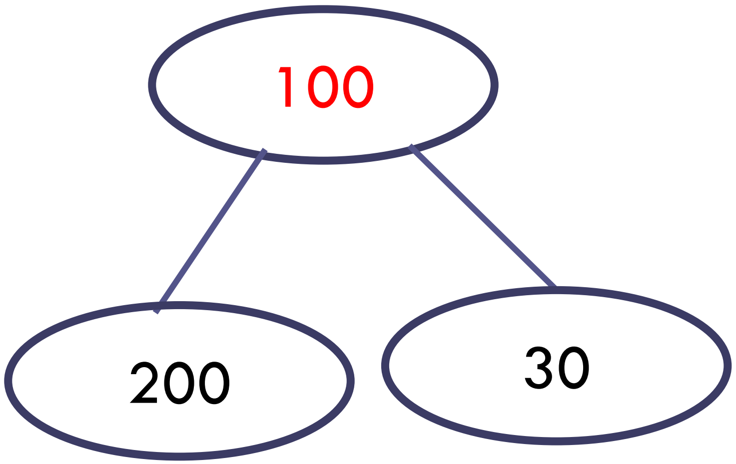
   `else`
3. Inorder traverse the left subtree.
4. Visit the root.
5. Inorder traverse the right subtree.

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.

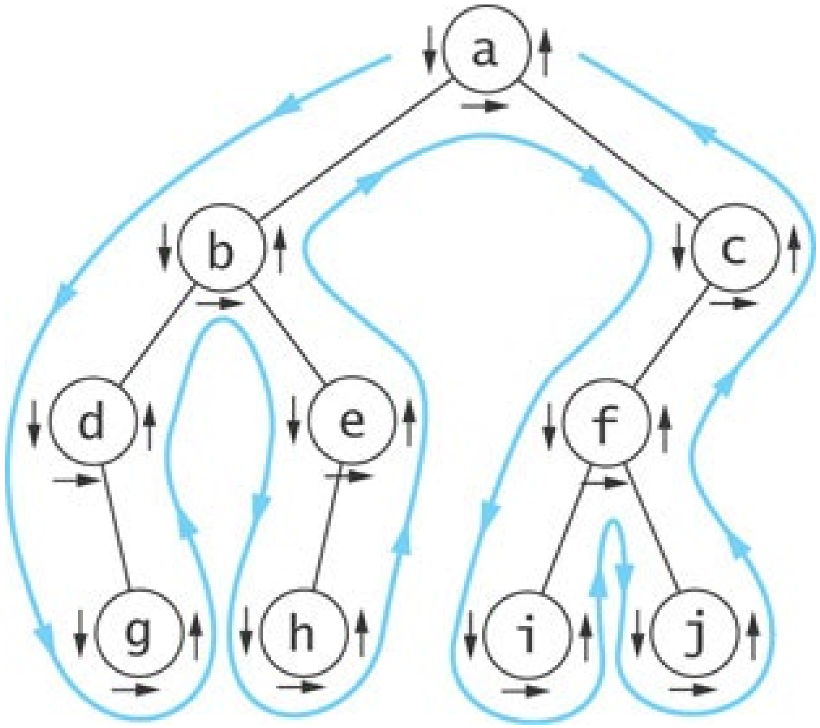
Tree Traversal Example



- **Pre**order sequence: 100 200 30
- **In**order sequence: 200 100 30
- **Post**order sequence: 200 30 100

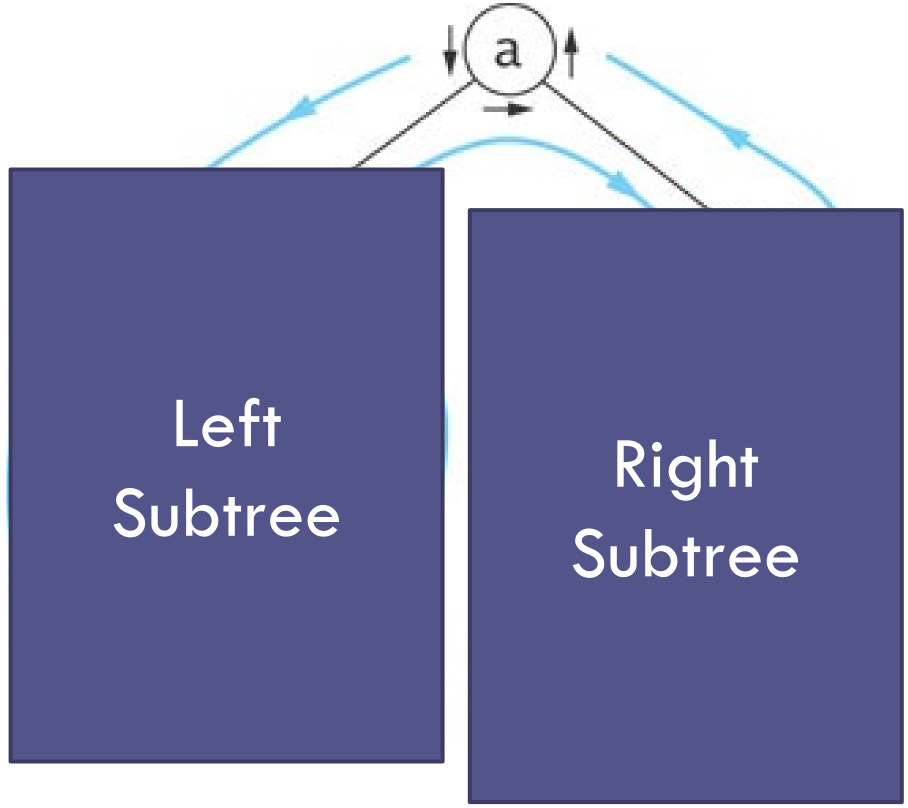
Visualizing Tree Traversals

- **Imagine a mouse walking along the edge** of the tree
- If the **mouse always keeps the tree to the left,** the trace route is the ***Euler tour***
  - **Path traced in blue** in the figure on the right



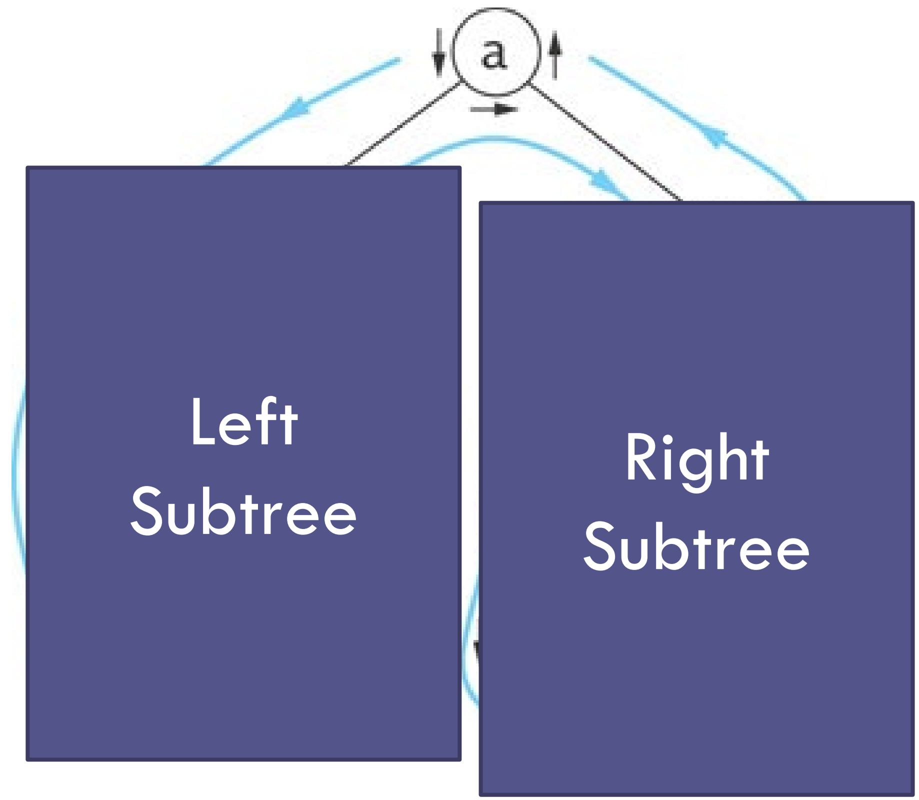
Visualizing Tree Traversals (cont.)

- **Preorder** traversal
  - **Euler tour** (**blue path**)
  - The mouse visits each node before traversing its subtrees
  - Marked by the **downward pointing arrows**



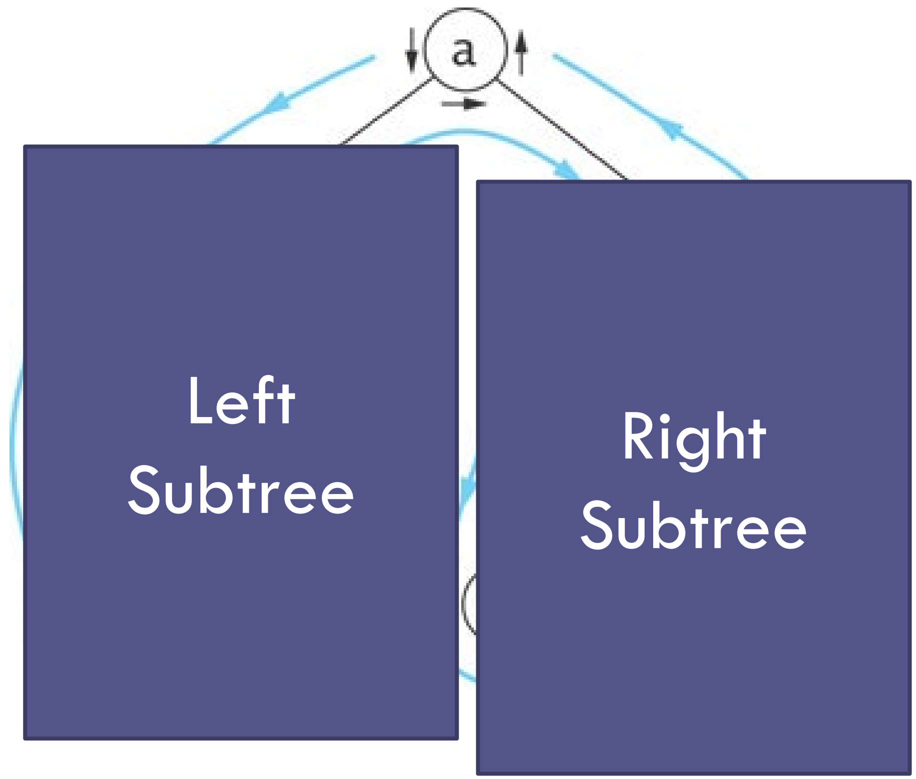
Visualizing Tree Traversals (cont.)

- **Inorder** traversal
  - **Record a node** as the mouse RETURNs from traversing its LEFT subtree
  - Marked by **horizontal black arrows** in the figure on right



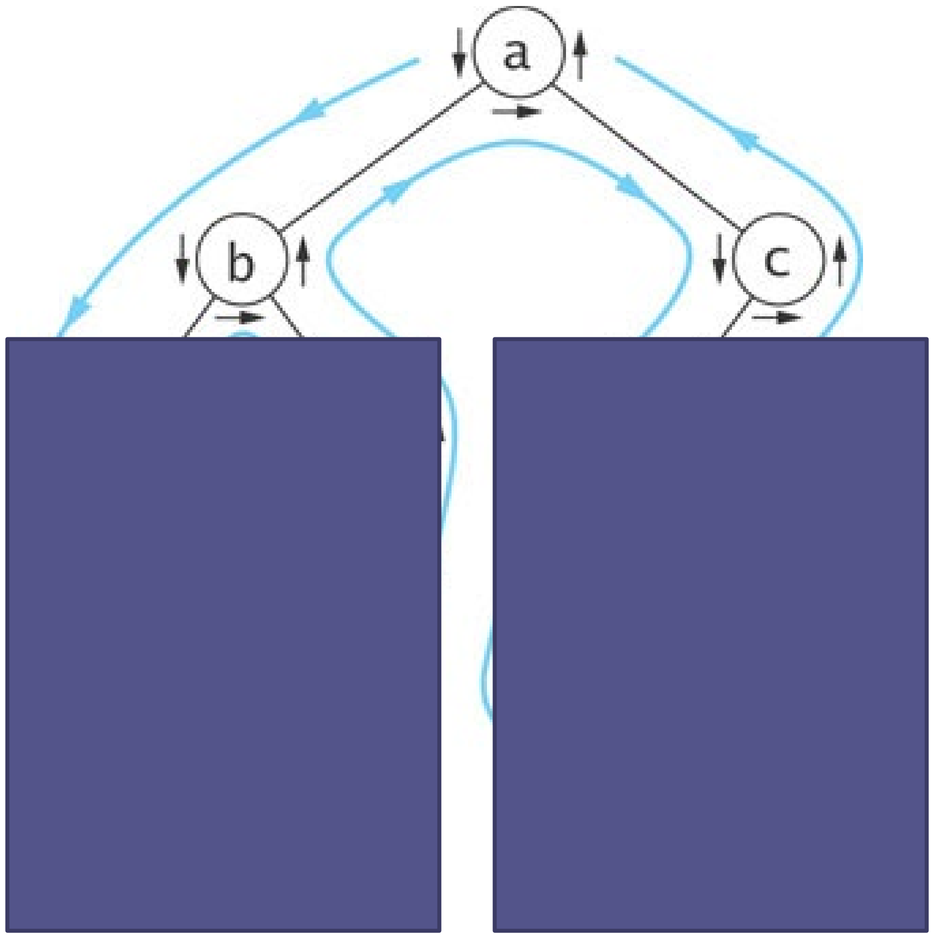
Visualizing Tree Traversals (cont.)

- **Postorder** traversal
  - **Record each node** as the mouse **LAST ENCOUNTERs** it
  - Marked by the **upward pointing arrow**



Visualizing Tree Traversals (cont.)

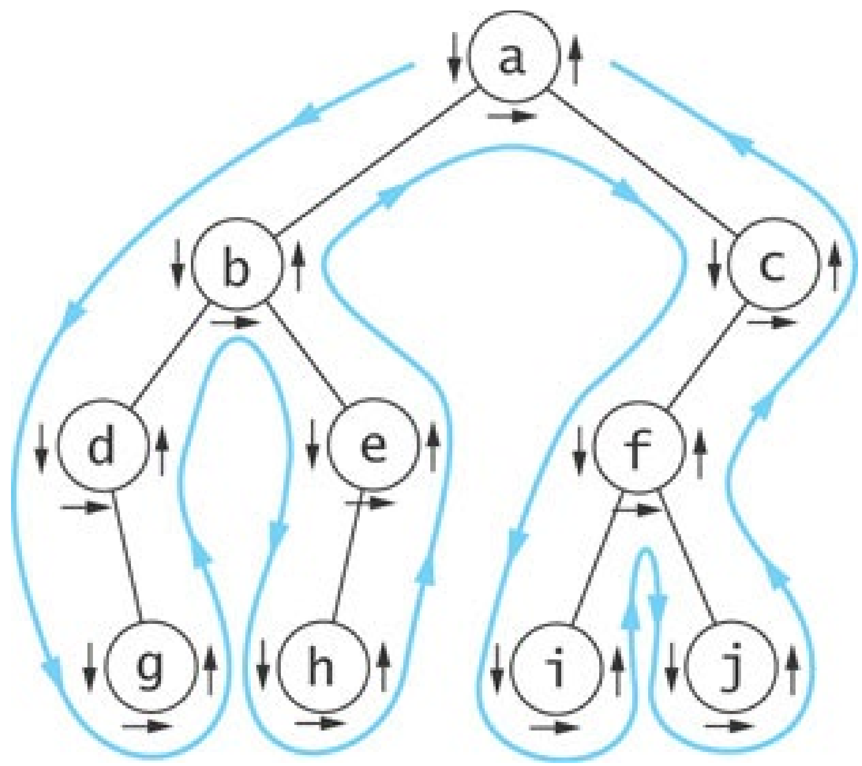
- **preorder** traversal
  - downward pointing arrows
  - Printout: abc
- **inorder** traversal
  - **horizontal black** arrows
  - Printout: bac
- **postorder** traversal
  - **upward** pointing arrows
  - Printout: bca



Visualizing Tree Traversals (cont.)

- **Preorder** traversal
  - **Euler tour** (**blue path**)
  - The mouse visits each node <u>before</u> traversing its subtrees
  - Marked by the **downward pointing arrows**
- The preorder sequence in this example

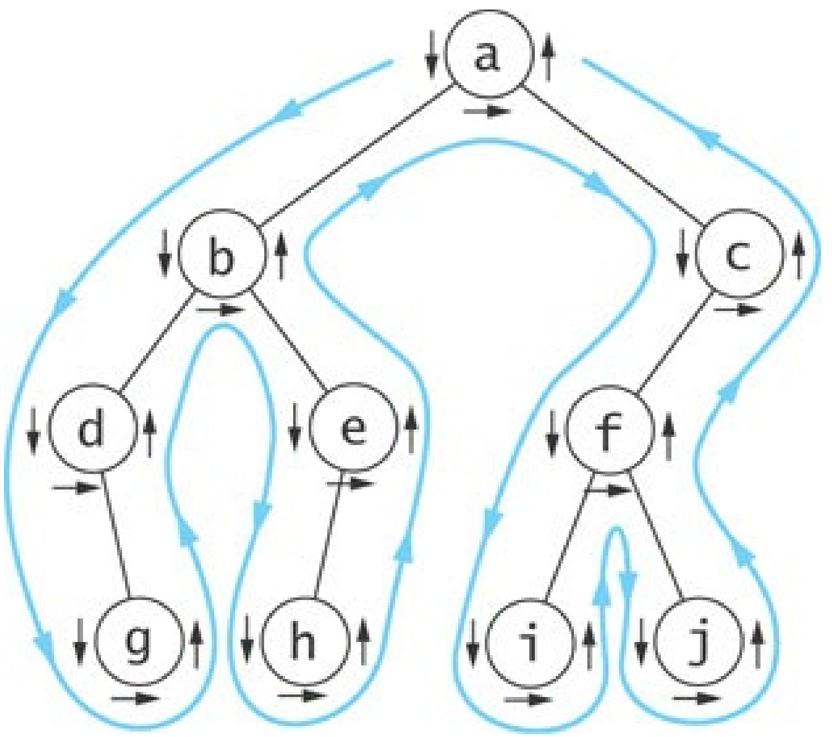
  a b d g e h c f i j



Visualizing Tree Traversals (cont.)

- **Inorder** traversal
  - **Record a node** as the mouse RETURNs from traversing its LEFT subtree
  - Marked by **horizontal black arrows** in the figure on right
- The inorder sequence:

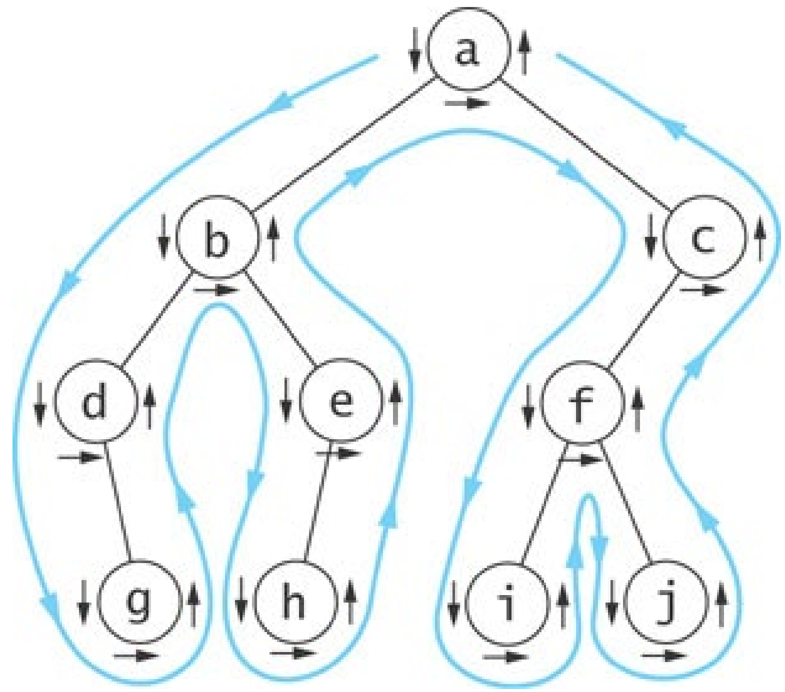
  d g b h e a i f j c



Visualizing Tree Traversals (cont.)

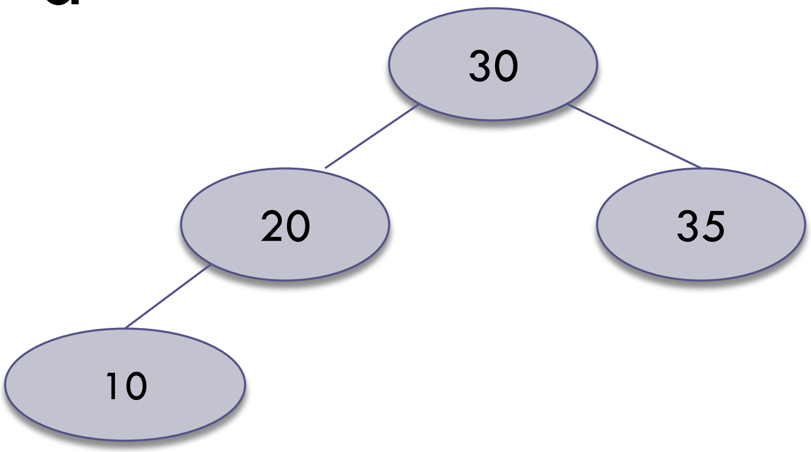
- **Postorder** traversal
  - **Record each node** as the mouse **LAST ENCOUNTERs** it
  - Marked by the **upward pointing arrow**
- The postorder sequence:

  g d h e b i j f c a



Traversals of Binary Search Trees

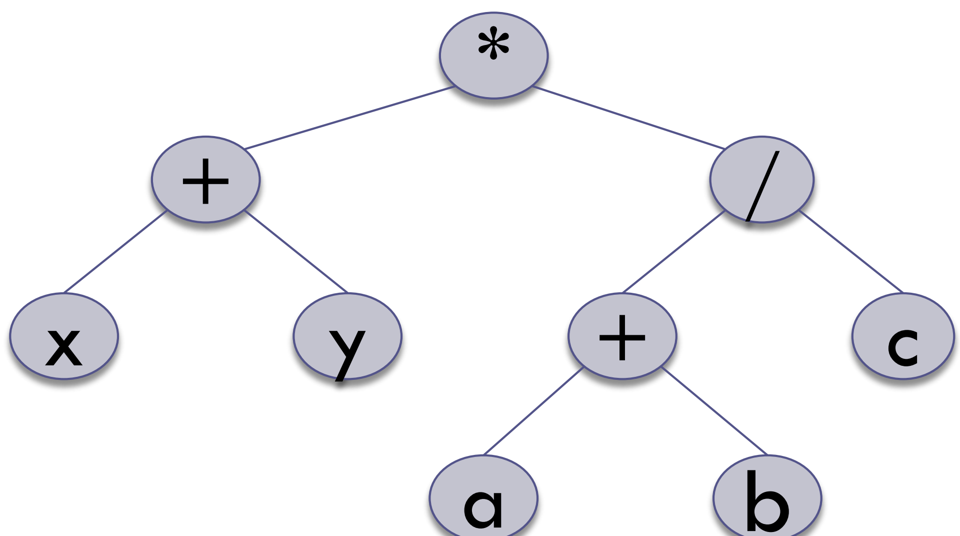
- An **inorder traversal** of a binary search tree
  - Nodes being visited in sequence by increasing data value



10, 20, 30, 35

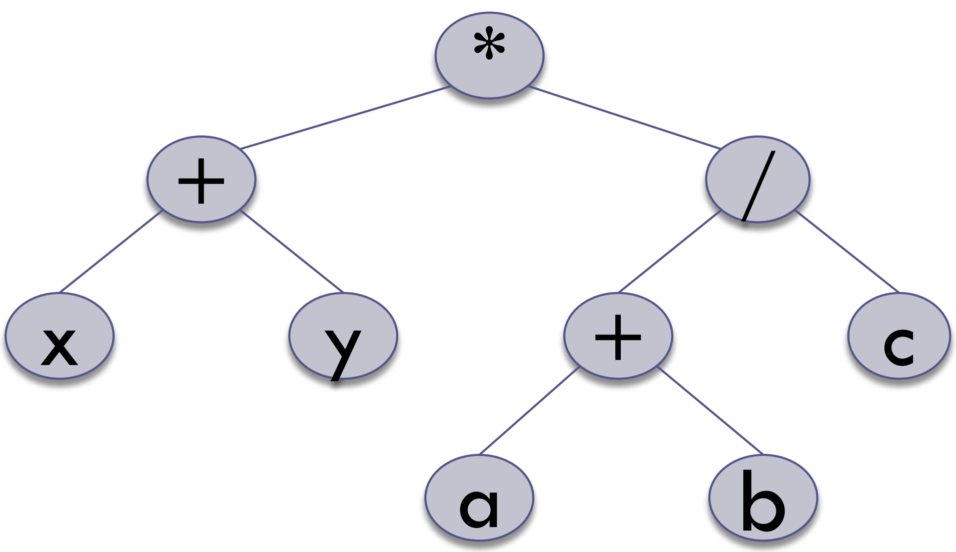
Traversals of Expression Trees

- **inorder traversal** sequence:
  x + y * a + b / c
- Add parentheses where they belong and get the **infix form:** `(x + y) * ((a + b) / c)`



Traversals of Expression Trees (cont.)

- **postorder traversal** sequence
  x y + a b + c / *
- ***postfix* or *reverse polish* form** of the expression
- **Operators follow operands**



Traversals of Binary Search Trees and Expression Trees (cont.)

- **preorder traversal** sequence
  * + x y / + a b c
- ***prefix* or *forward polish* form** of the expression
- **Operators precede operands**