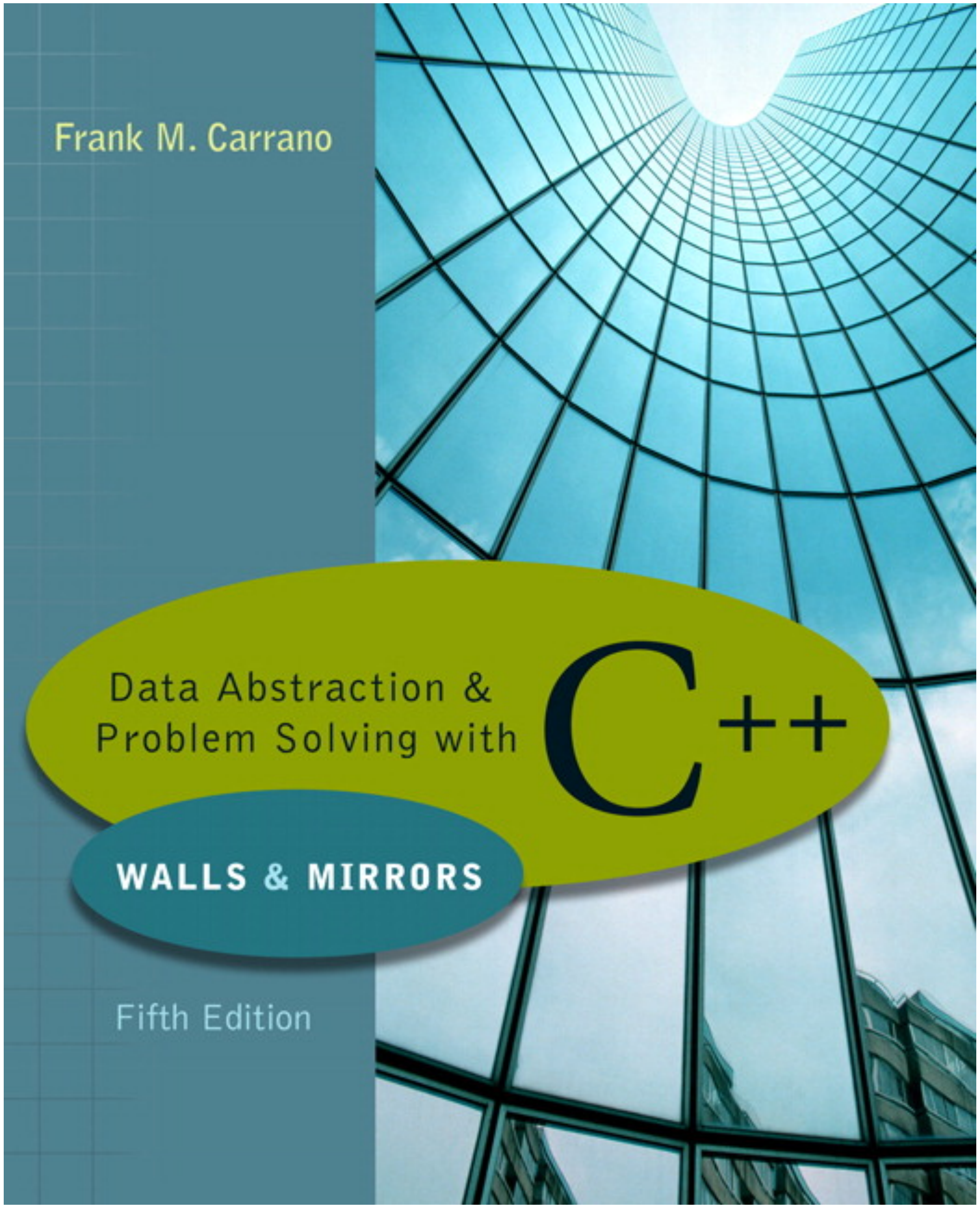
CHAPTER 10
Trees



What have we learned so far?

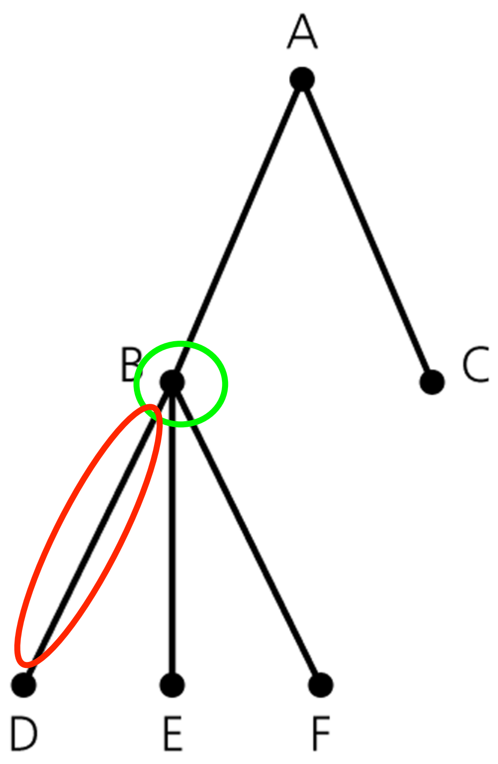
- ADT list, stack, queue
- General data management operations
  - Insert
  - Delete
  - Query: is empty? Length? retrieve.
- Operations
  - Position-oriented
  - Value-oriented

What is coming

- □ Trees
- □ Terminologies
- □ Binary search tree
- □ Operations

Terminology

- Tree: A connected, undirected graph without cycles
  - The process of arranging the vertices into a topological order
  - graph: a set of vertices and a set of edges vertices
- Vertex: a node in a graph
- Edge: line (connection) between nodes



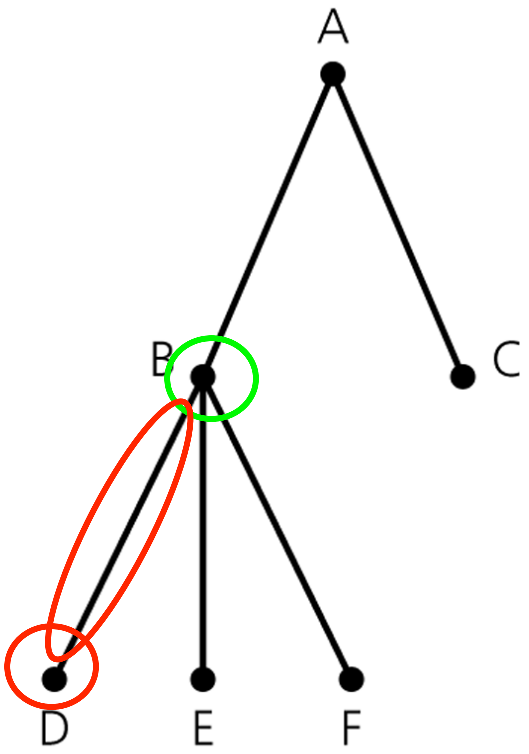
Terminology

- Tree: all trees are hierarchical in nature
  - Hierarchical: parent/child relationship between nodes in the tree
- Parent / child

  If an edge is between two nodes

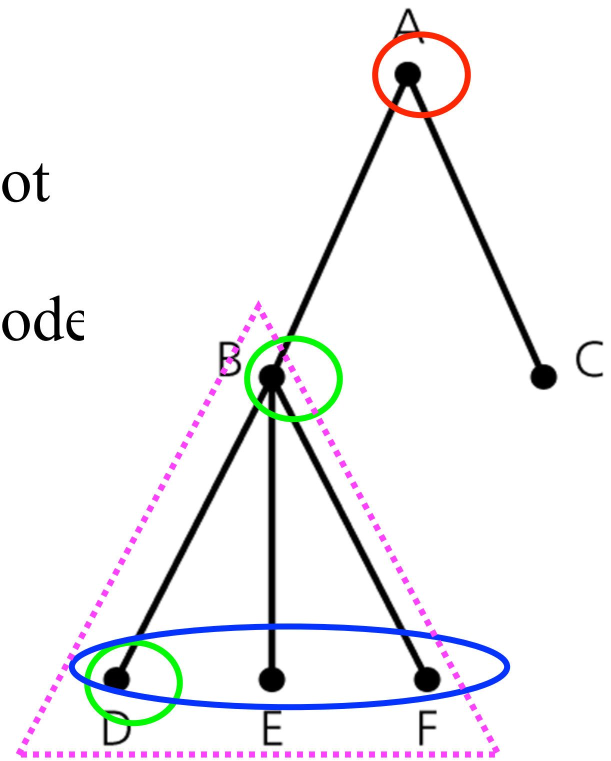
  B and D, B is above D in the tree

  B is the parent of D, D is a child of B

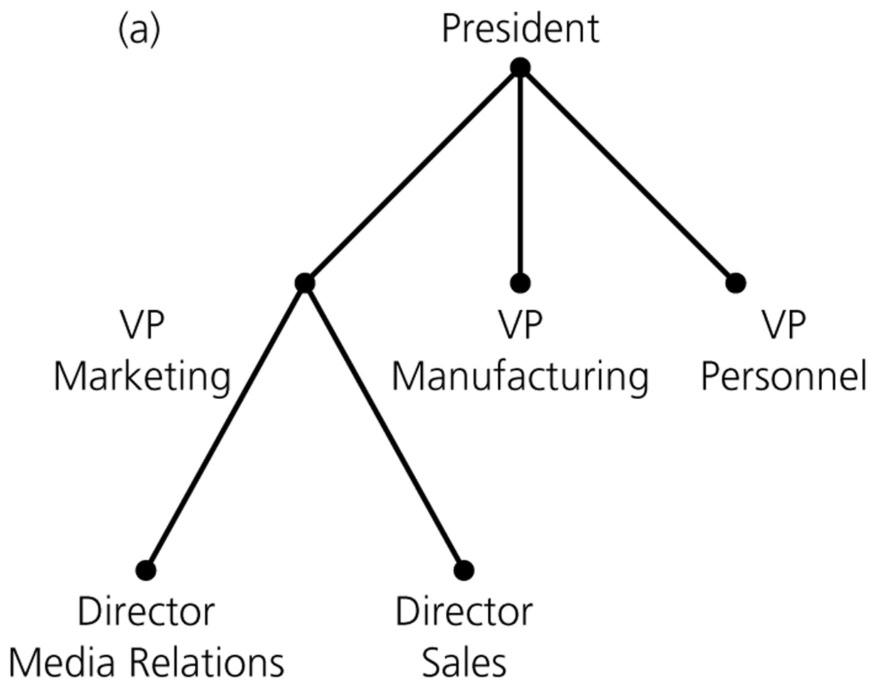


Terminology continued …

- Sibling: children of the same parent (DEF)
- Root: node with no parent (one node in a tree)
- ancestor / descendant
  - ancestor: a node on the path from the root of a tree to the node
  - descendant: a node on a path from the node to a leaf of a tree
- Subtree : any node in a tree, together with all of the node's descendants

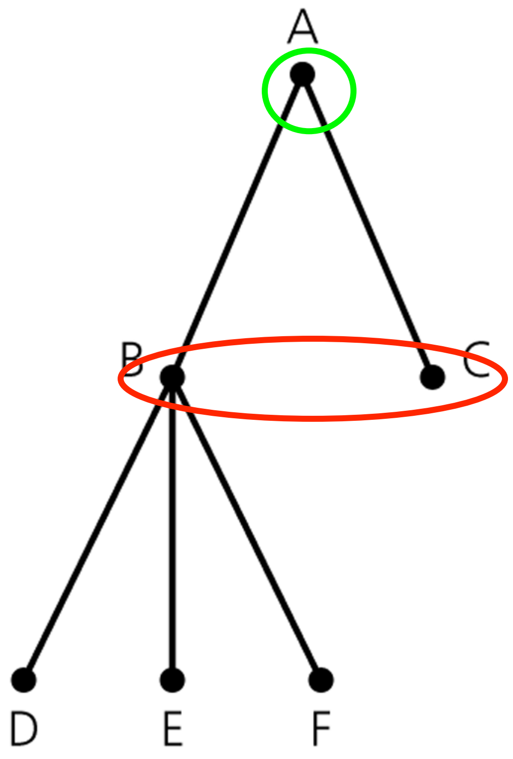


Represent information that is hierarchical in nature
(a)An organization chart;
(b)a family tree



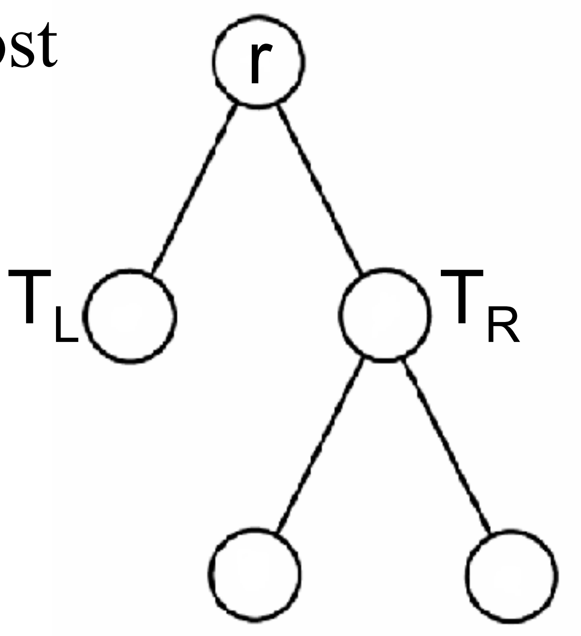
A general tree (T)

- 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



A binary tree

- A set of zero or more nodes, partitioned into a root node and two possibly empty sets that are binary trees.
- Each node in a binary tree has at most two children, the left child and right child
- T_L: left subtree
- T_R: right subtree



An application of binary tree:

Binary trees that represent algebraic expressions

$a - b$

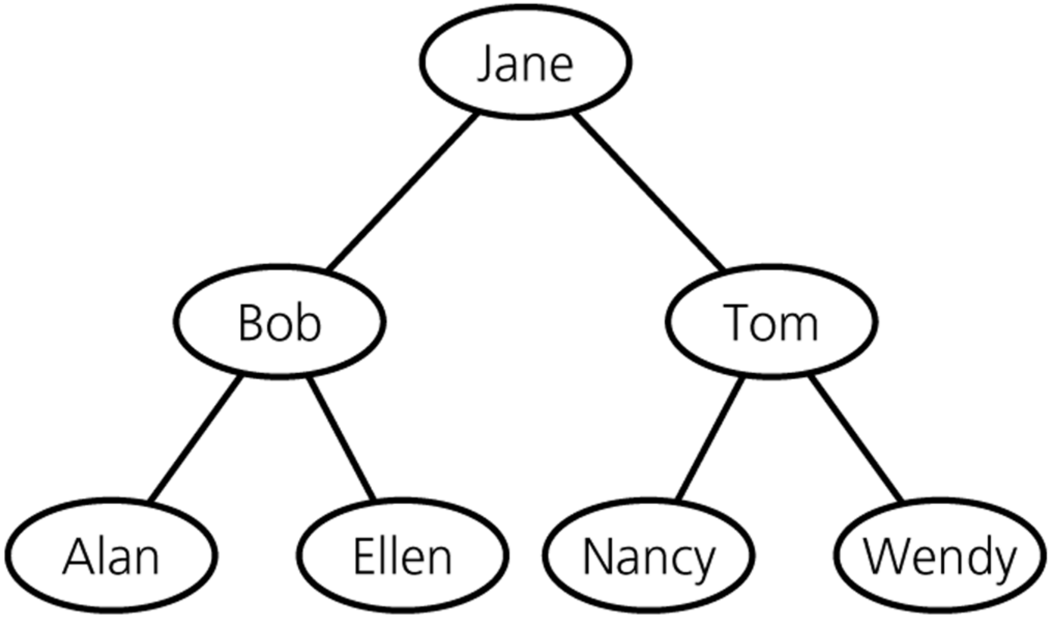
$a - b / c$

$(a - b) * c$

Leaves of these trees contain the expressions operands.
Other tree nodes contain the operators.

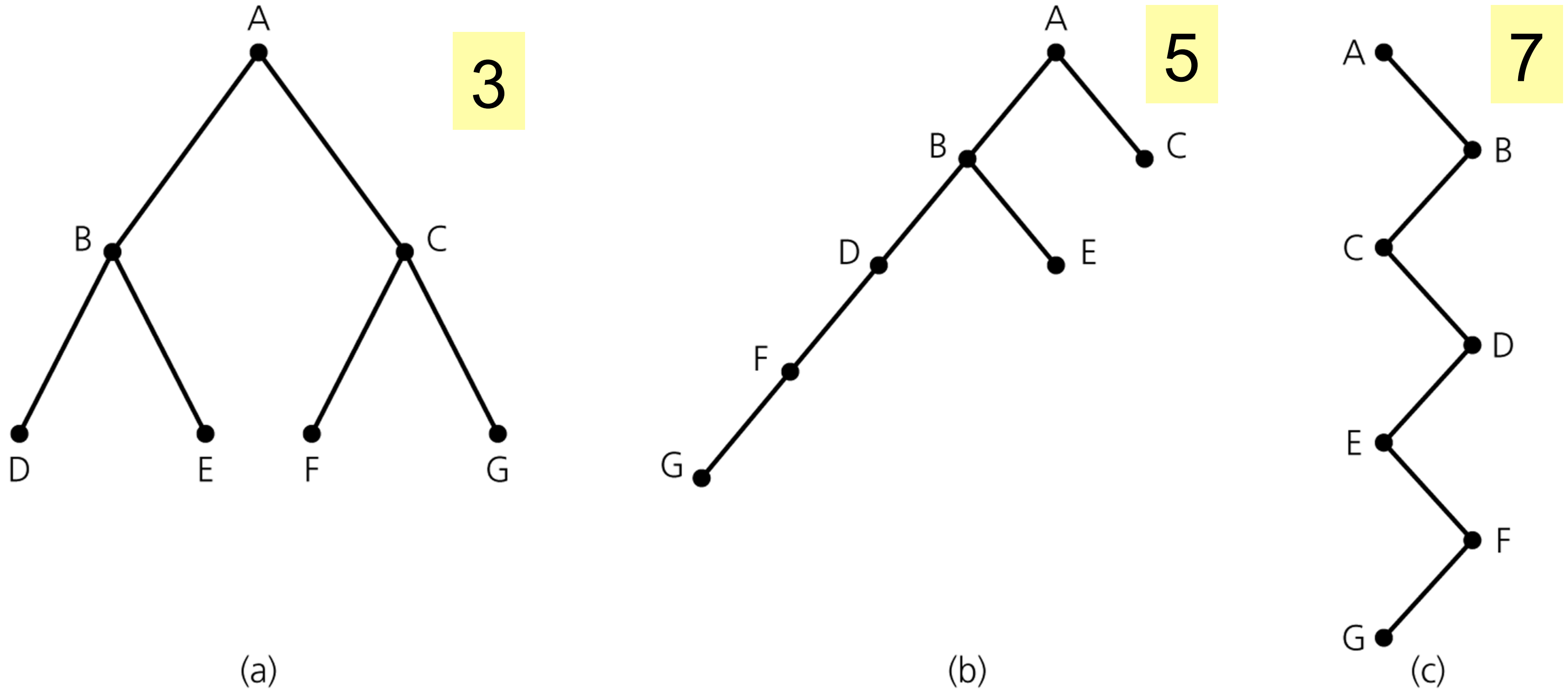
An application of binary tree: A binary search tree (BST)

- A binary tree where the search key in any node n is
  - greater than the search key in its left subtree (BST),
  - but less than the search key in any node in its right subtree (BST)



The height of trees

- The number of nodes on the longest path from the root to a leaf.



3

5

7

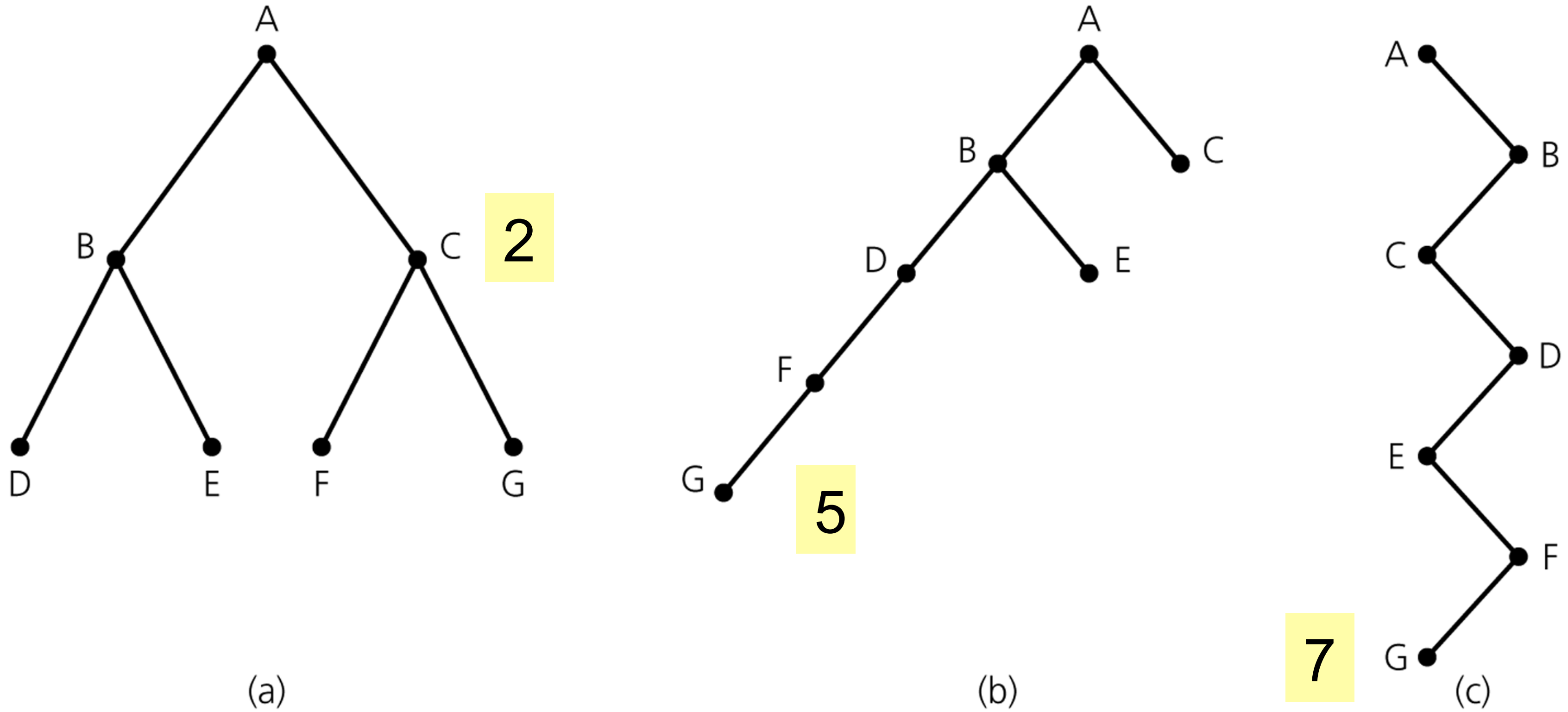
(a)

(b)

(c)

The level of a node n

- 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



(a)

(b)

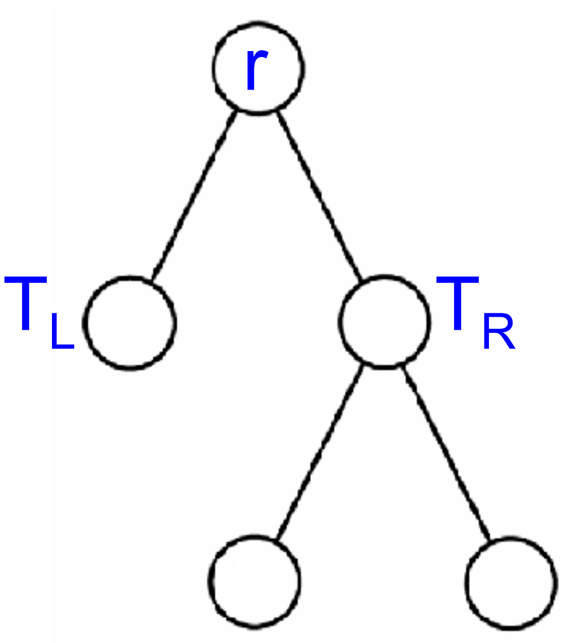
(c)

Height in a binary tree

- If T is empty, its height is 0
- If T is a nonempty binary tree,

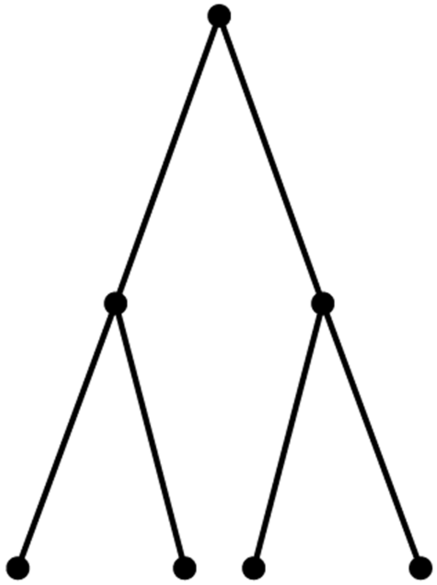
Height(T) = 1 + max(height(T_L), height(T_R))

- T_L: left subtree
- T_R: right subtree



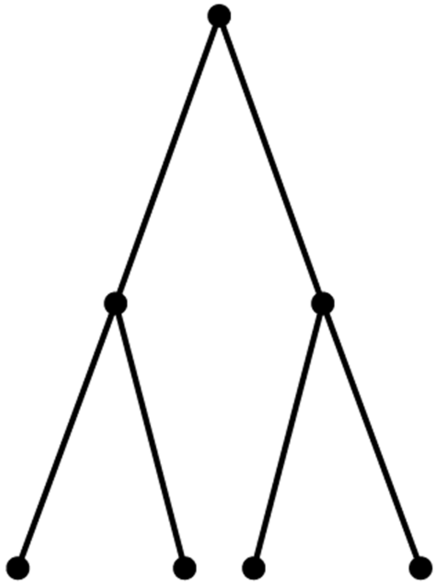
Full binary tree

- Full binary tree: all nodes that are at a level less than the height, h, has two children each
  - If T is empty, T is a full binary tree of height 0
  - If T is not empty and has height h, T is a full binary tree if its root's subtrees are both full binary tree of height h-1



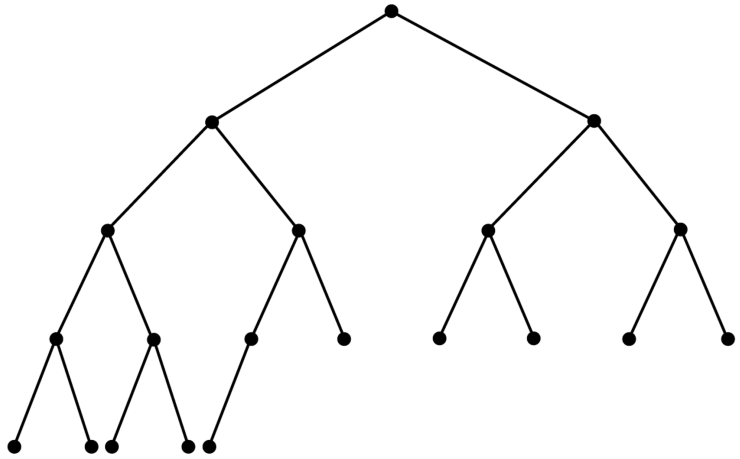
Full binary tree

- A full binary tree of height h has
  - $2^h - 1$ nodes
  - 2^{h-1} terminal nodes (leaves)
- A full binary tree of n nodes
  - $\lceil \log_2 n \rceil$ height



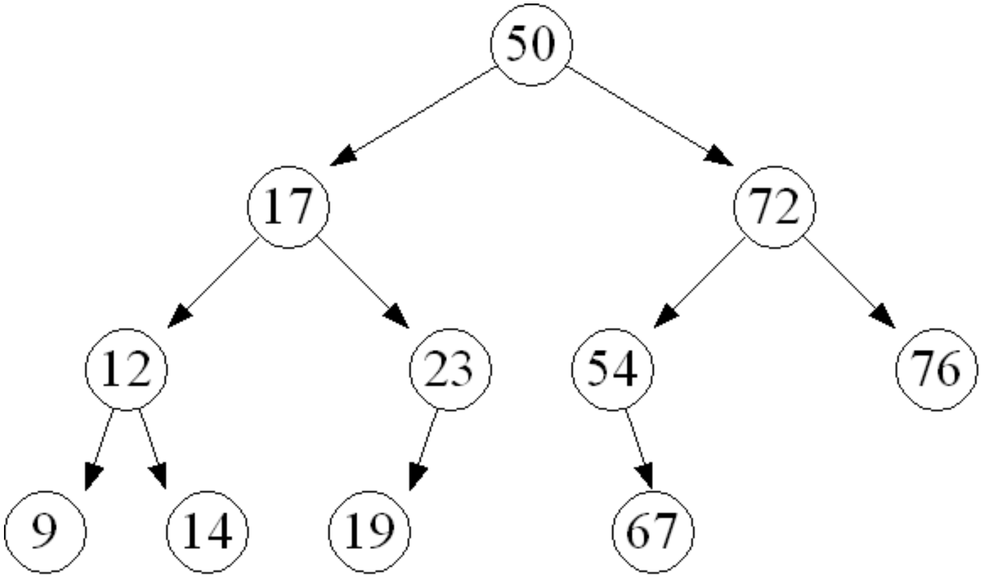
Complete binary tree

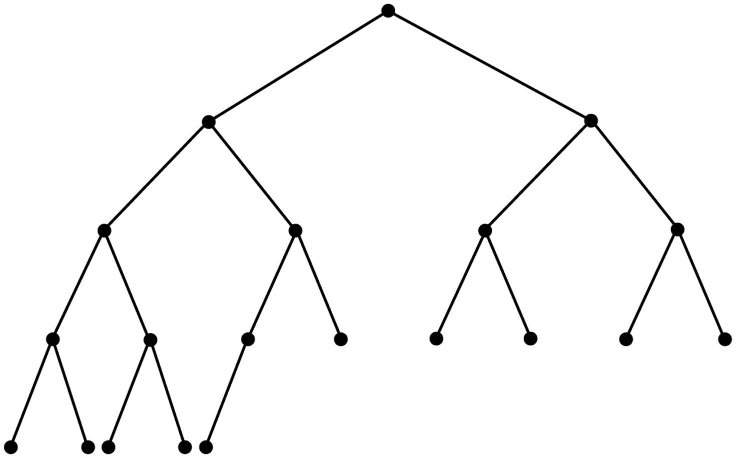
- Complete binary tree: full down to level h-1, with level h filled in from left to right
  - All nodes at level h-2 and above has two children each,
  - When a node at level h-1 has children, all nodes to its left at the same level have two children each
  - When a node at level h-1 has one child, it is left child



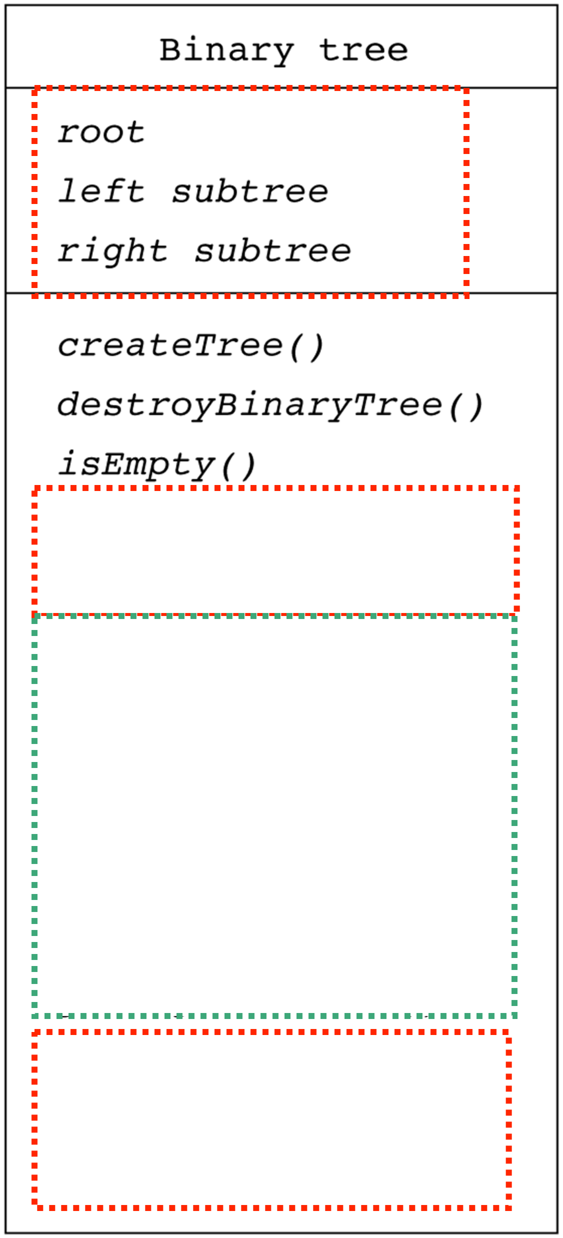
AVL tree: balanced binary search tree

- A binary tree is height 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
- The AVL tree is named after its two inventors, G.M. Adelson-Velsky and E.M. Landis





UML diagram for the class *BinaryTree*



Binary tree

root

left subtree

right subtree

createTree()

destroyBinaryTree()

isEmpty()

ADT binary tree operations

- Create an empty binary tree
- Create a one-node binary tree given an item
- Create a binary tree given an item for its root and two binary tree for the root’s subtrees

- Destroy a binary tree

- Determine whether a binary tree is empty

ADT binary tree operations

- Determine or change the data in the binary tree’s root

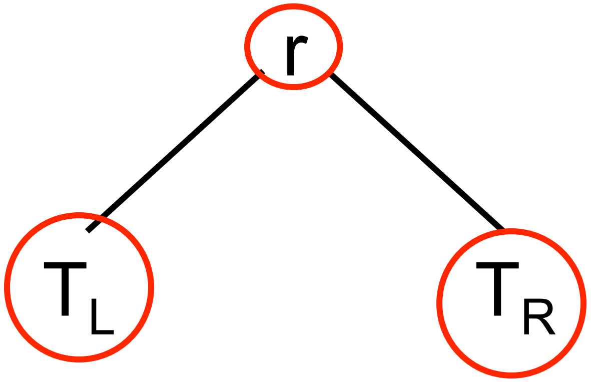
- Attach a left or right child to the binary tree’s root
- Attach a left or right subtree to the binary tree’s root
- Detach the left or right subtree of the binary tree’s root

- Return a copy of the left or right subtree of binary tree’s root

- Traverse the nodes in a binary tree in preorder, inorder, or postorder

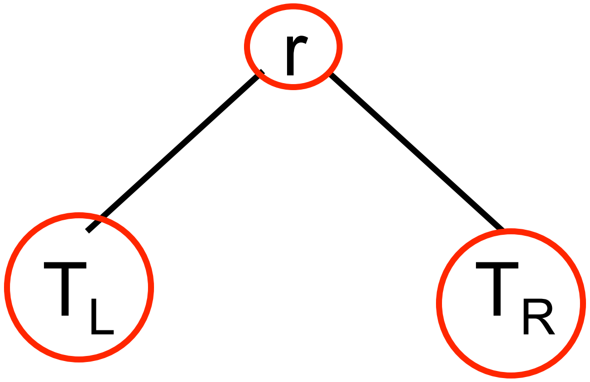
Binary tree traversal

- Preorder : r→ T_L→ T_R



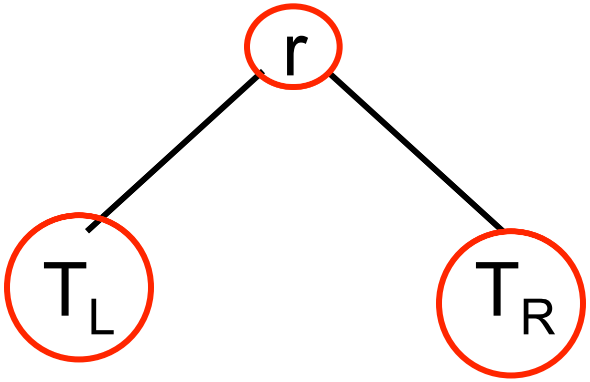
Binary tree traversal

- Preorder : r→ T_L→ T_R
- inorder : T_L → r→ T_R



Binary tree traversal

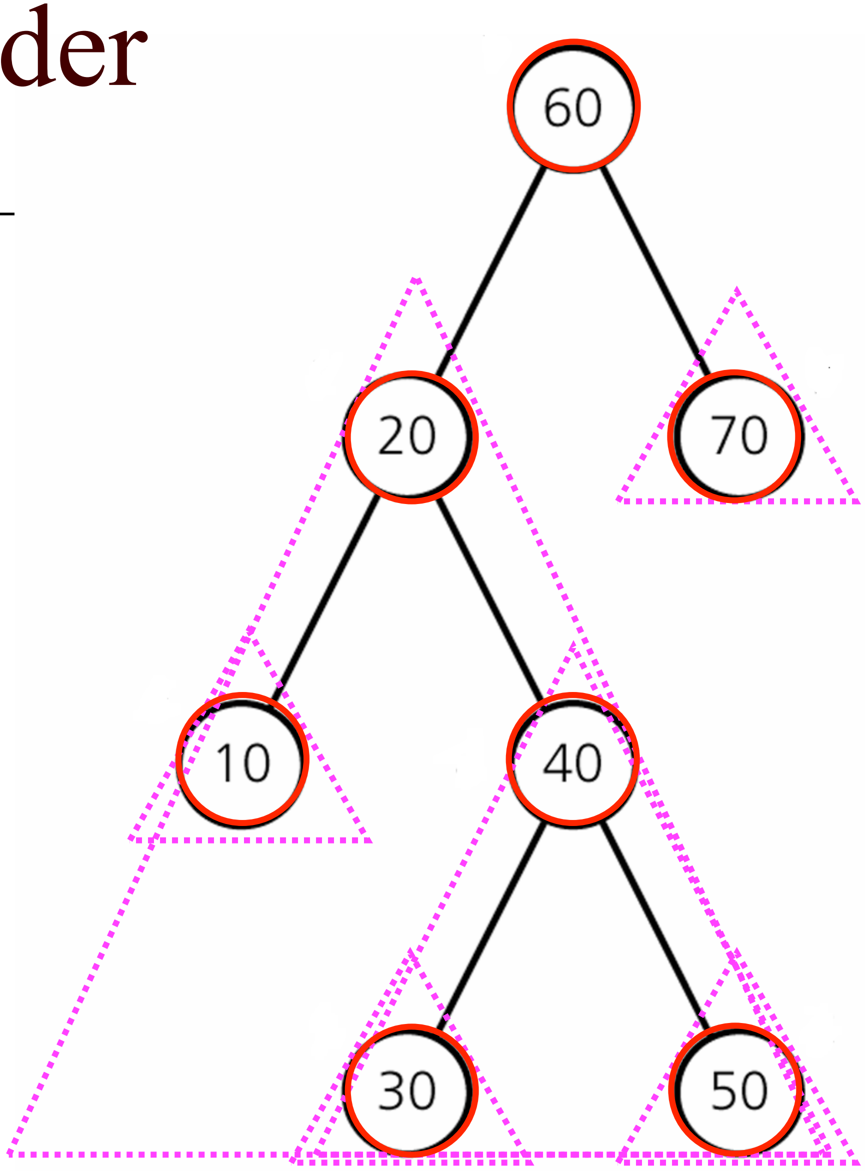
- Preorder : r→ T_L→ T_R
- inorder : T_L → r→ T_R
- Postorder : T_L→ T_R → r



Traversal: preorder

1. Root
2. Left child
3. Right child

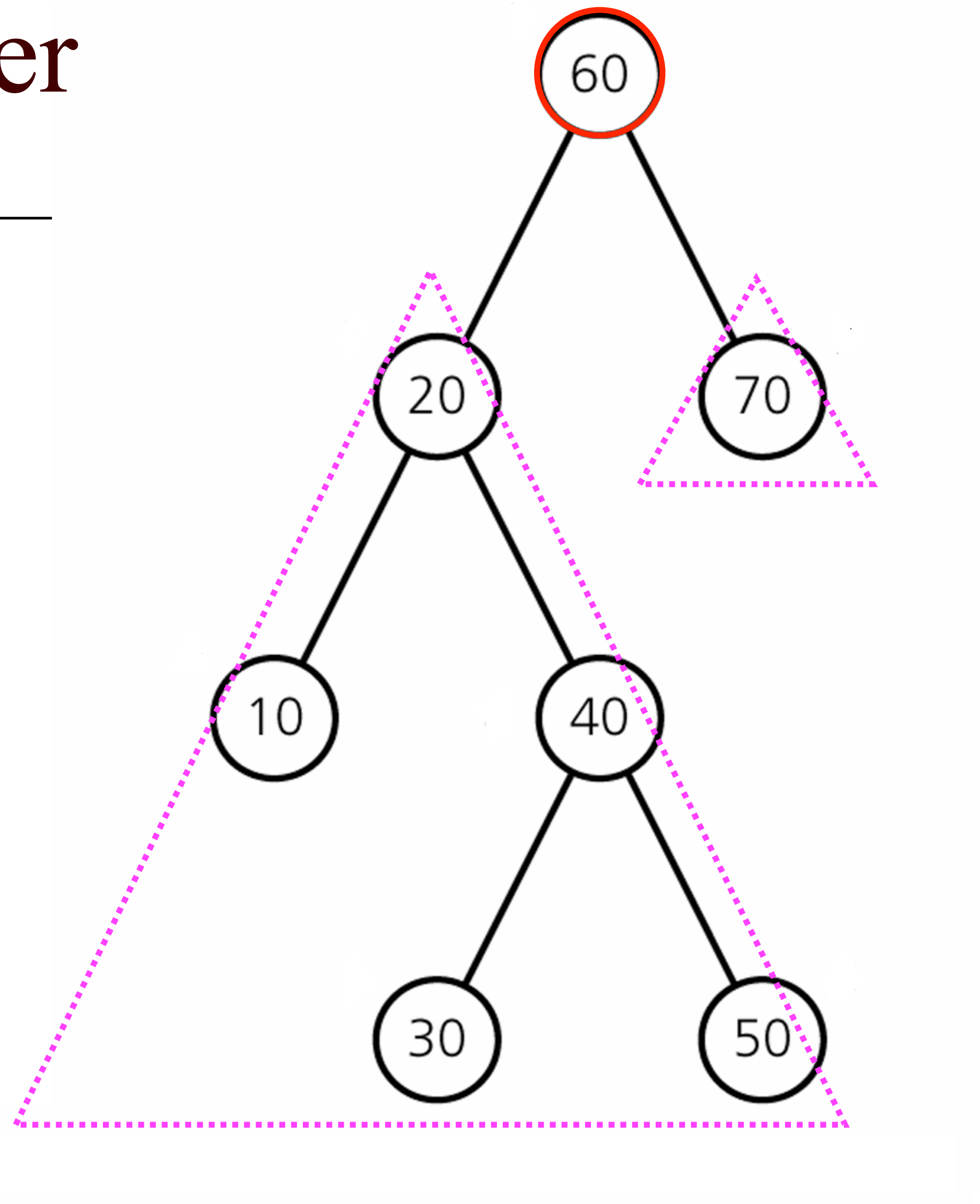
□ Recursively!!



Traversal: inorder

1. Left child
2. Root
3. Right child

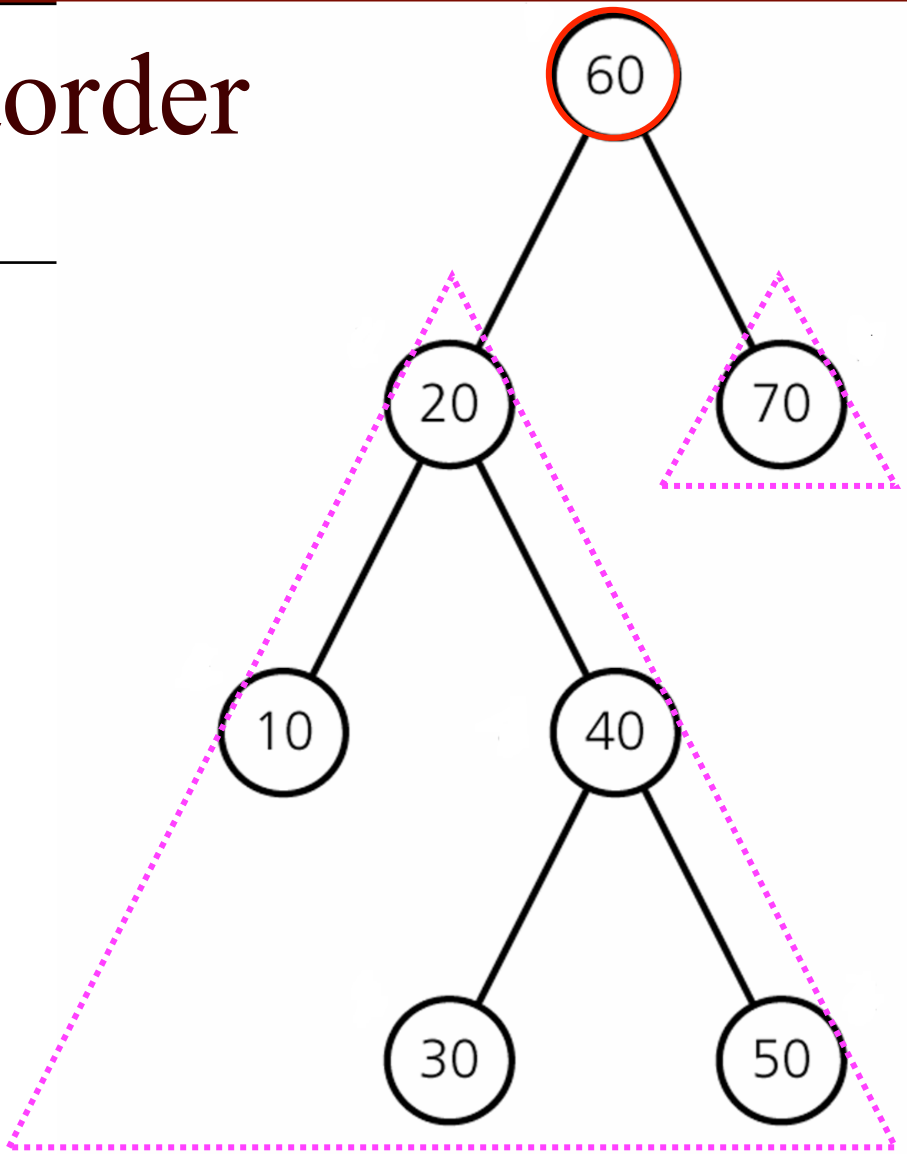
- Recursively!!



Traversal: postorder

1. Left child
2. Right child
3. Root

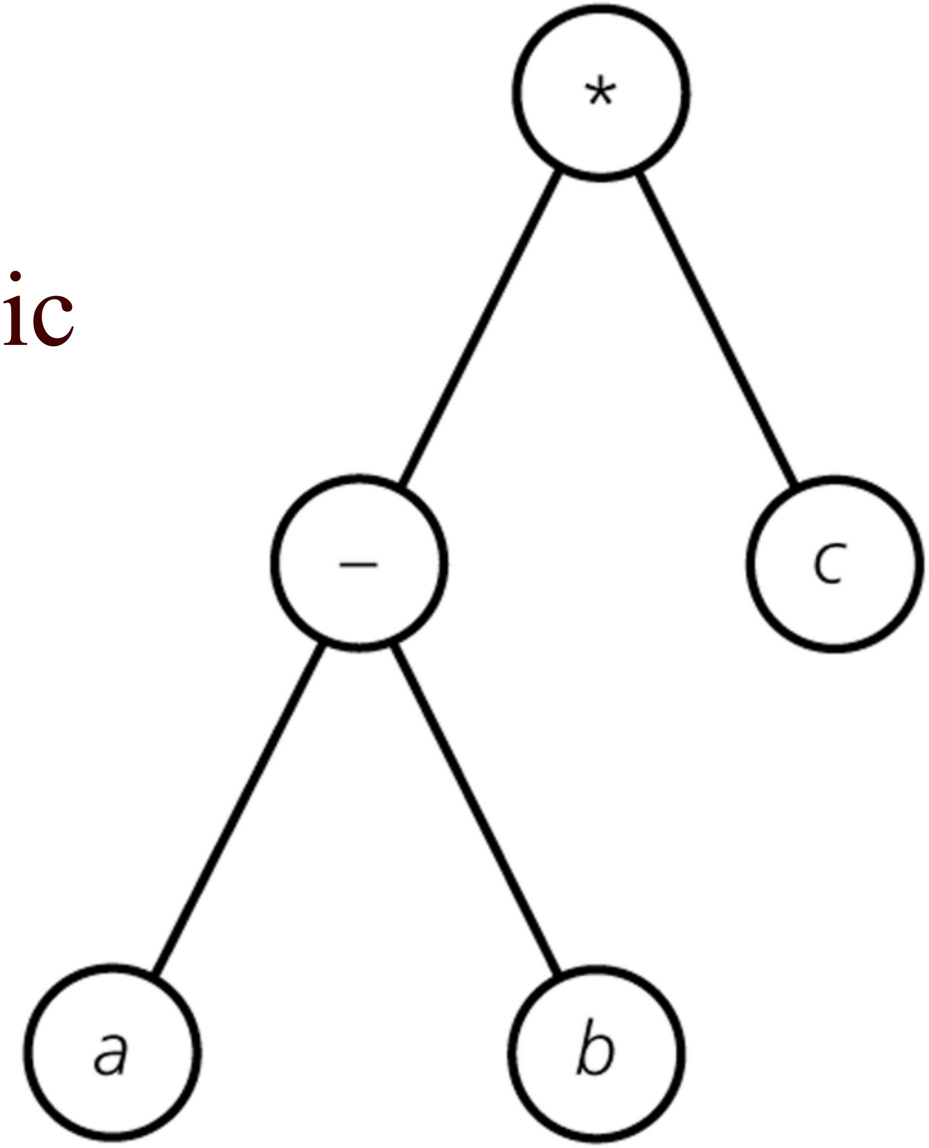
□ Recursively!!



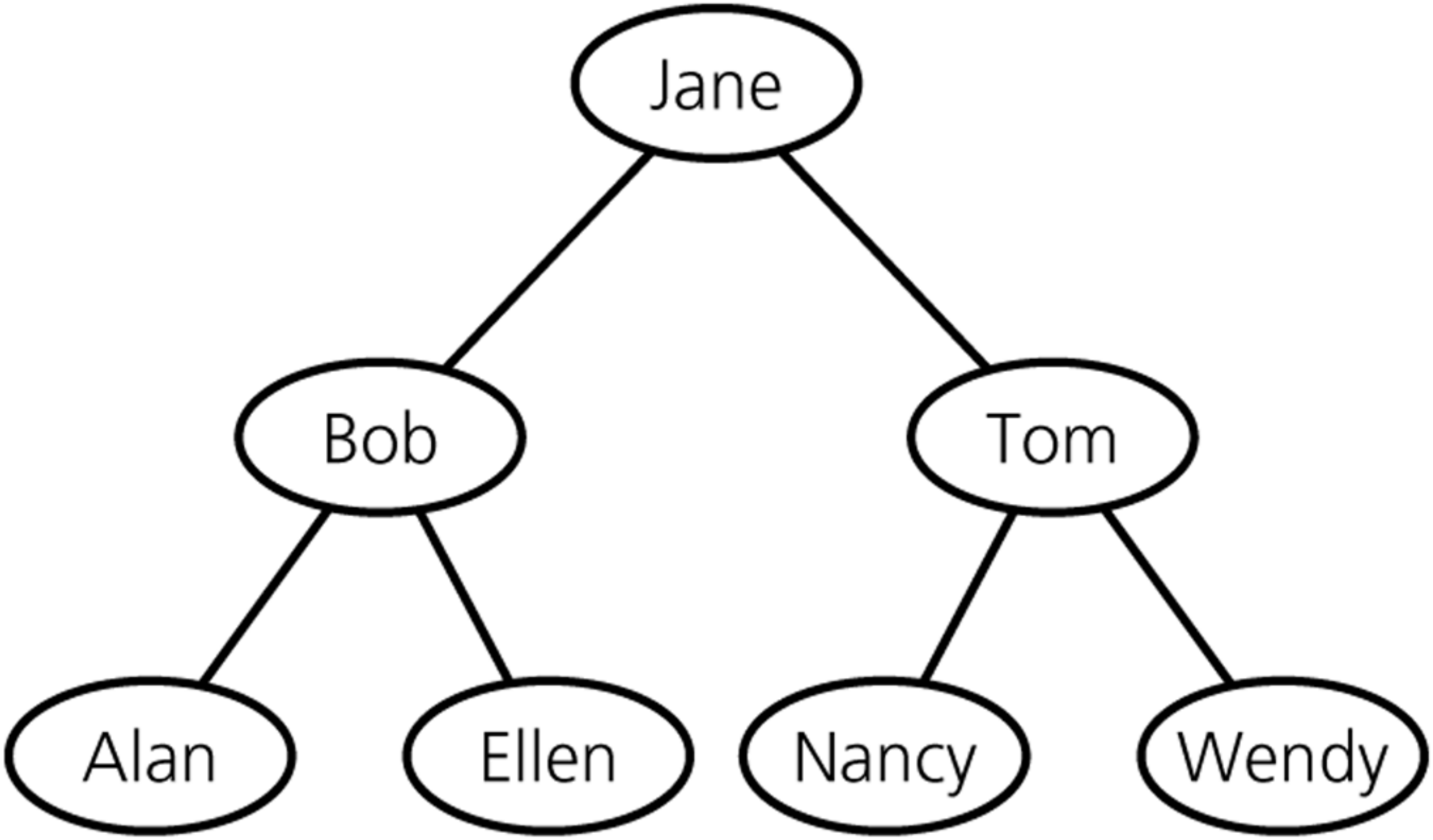
Traverse

a binary tree that represent algebraic expressions

- Preorder
- Inorder
- Postorder



Traverse a binary search tree



What is coming

- Trees
- Operations

Preorder traversal in a binary tree T

```
Preorder (T)
{
    if (T is not empty) {
        visit root;
        Preorder ($T_L$);
        Preorder($T_R$);
}
```

Inorder traversal in a binary tree T

```
Inorder (T)
{
    if (T is not empty) {
        Inorder ($T_L$);
        visit root;
        Inorder($T_R$);
}
```

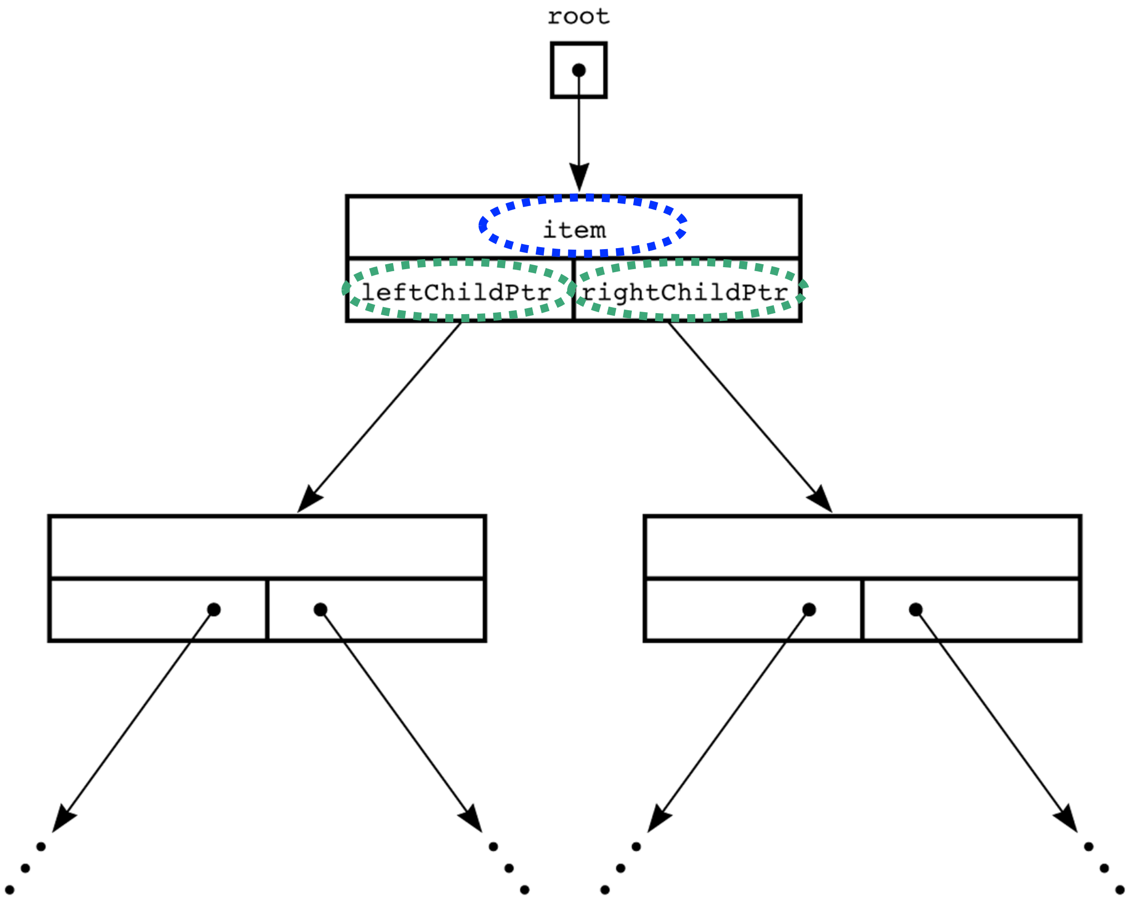
Postorder traversal in a binary tree T

```
Postorder (T)
{
    if (T is not empty) {
        Postorder ($T_L$);
        Postorder($T_R$);
        visit root;
}
```

Representations of a binary tree

- Pointer-based representation

A pointer-based implementation of a binary tree


root
item
leftChildPtr
rightChildPtr