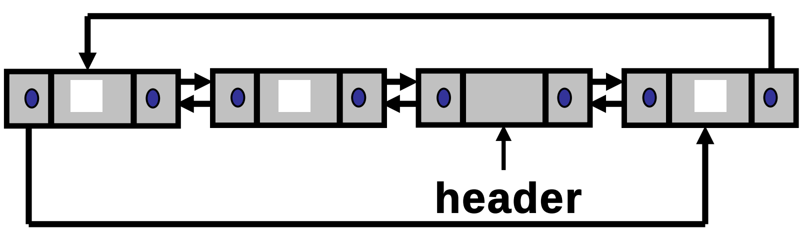
Trees

- Linear vs Nonlinear ADTs
- Tree Terminology
- Binary Tree
  - Binary Taxonomy Tree
  - Types of Binary Trees
- Binary Tree Representation
- Binary Tree Node Definition
- Binary Tree Scan Algorithms
  - Function Parameters
- Binary Search Tree

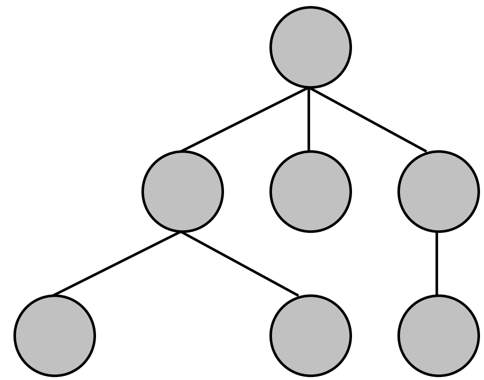
Linear ADTs

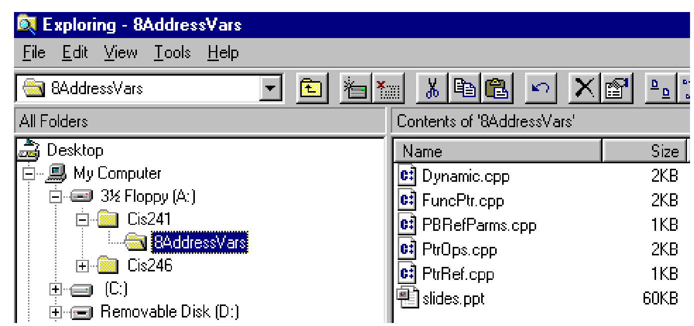
- Sequence containers store items in positional order
  - arrays, vectors, lists
- Linear ADTs have single successor



Nonlinear ADTs

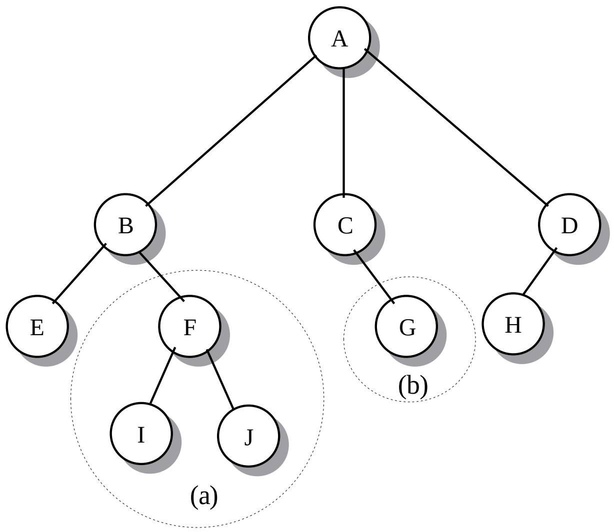
- Container components have more complex relationship
  - Often allows improved efficiency for component access
- Tree is hierarchical structure
  - Places elements in nodes along branches that originate from root
- Tree is nonlinear structure
  - Each element can have multiple successors





Tree Terminology

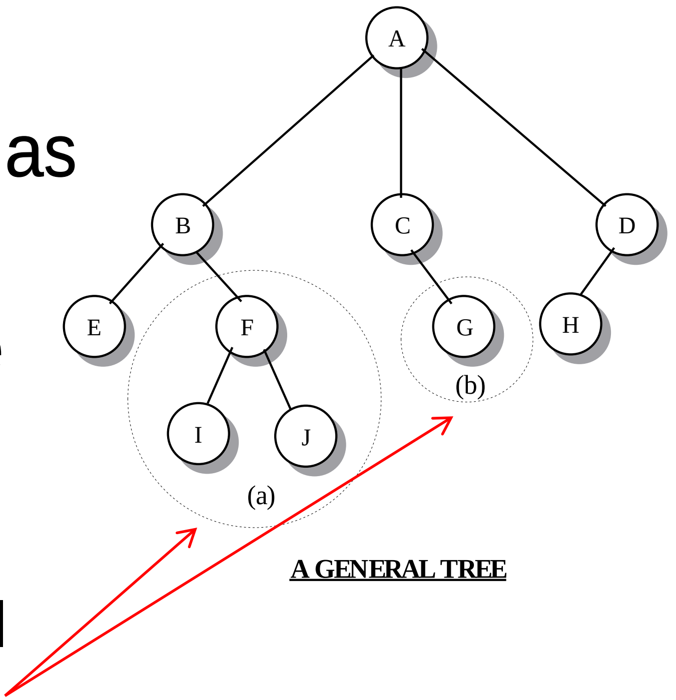
- unique component of tree is node that includes data and pointer references
  - tree with *no* nodes is empty tree
- unique node ancestor is called parent
- root node is the *unique* node of a tree with *no* ancestor
- node descendants are called children
- two children are siblings if they have the same parent
- connections from parent to child characterize the binary tree



A GENERAL TREE

Tree Terminology

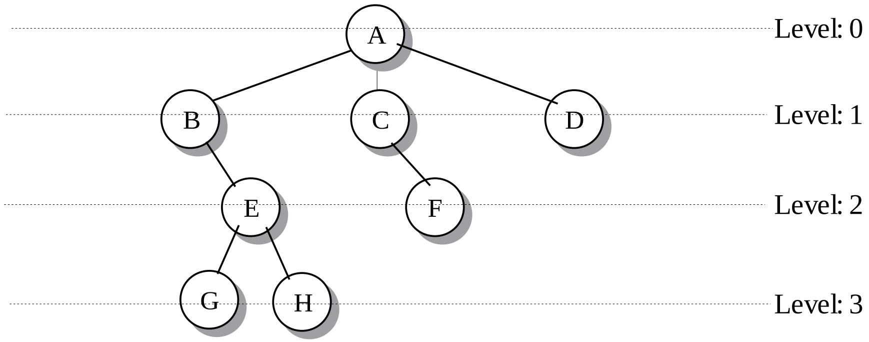
- leaf nodes are the nodes of a tree that have *no* descendant
- interior node (*nonleaf node*) has at least one child
- each node in a tree is also the root of a subtree (*recursive thinking*)
  - subtree describes a node and all its descendants



A GENERAL TREE

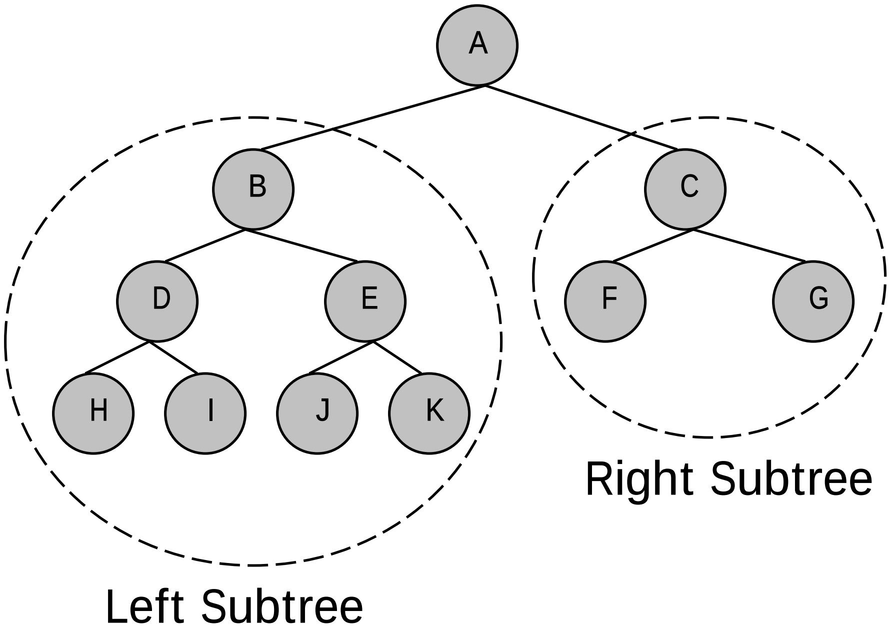
Tree Terminology

- path is linear subset of a tree (sequence of nodes)
  - there is a *unique* path from the root to any node
  - level or depth of a node is the length of its path from the root
- depth of tree is *maximum* depth of any node in the tree



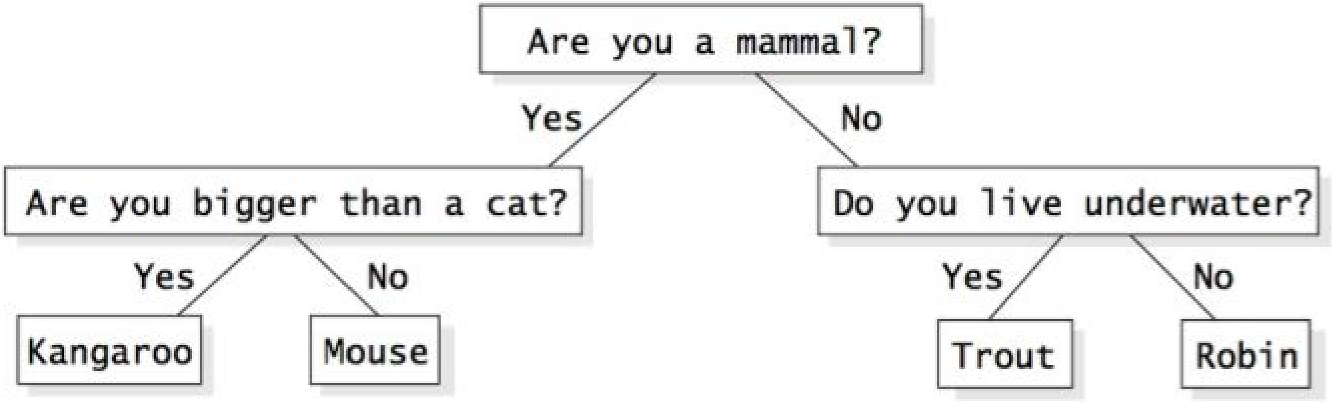
Binary Tree

- each parent has no more than two children (*descendent nodes*)
  - left child is node connected to left link
    - left subtree
  - right child is node connected to right link
    - right subtree
- each subtree is a binary tree
  - Recursive definition of binary tree
- a tree without any nodes is an empty binary tree



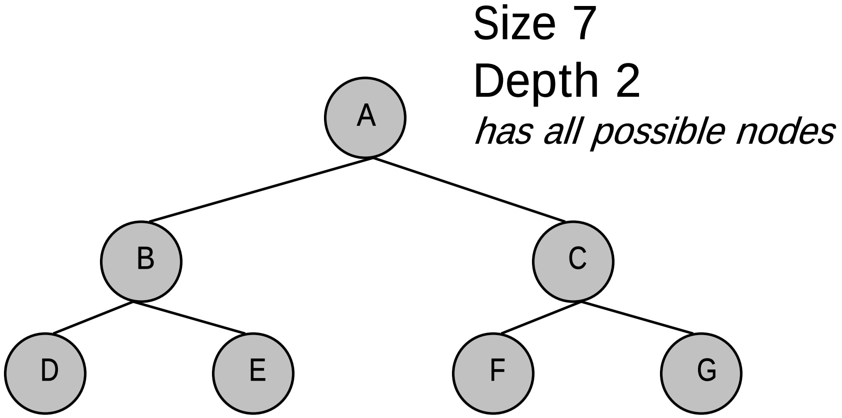
Binary Taxonomy Tree

- Stores increasing levels of detailed knowledge about a subject



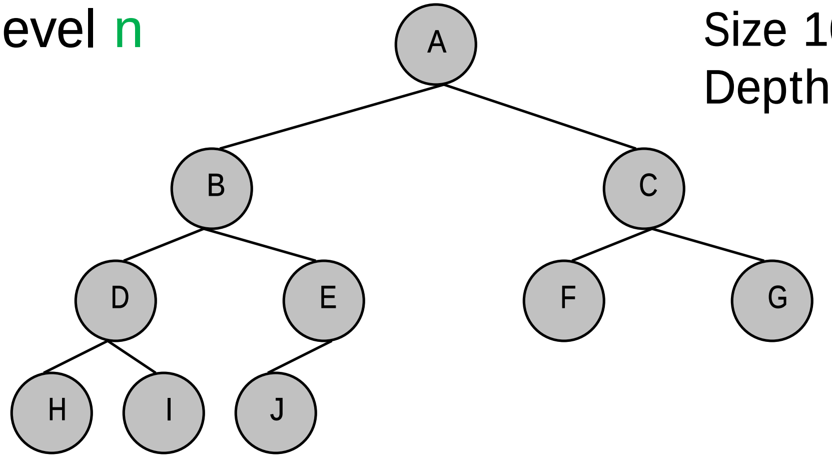
Types of Binary Trees

- full binary tree
  - every leaf node has same depth
  - every nonleaf node has two children
  - number of nodes in tree with n depth
    - $2^n + 2^{(n-1)} + \ldots + 2^0$



Types of Binary Trees

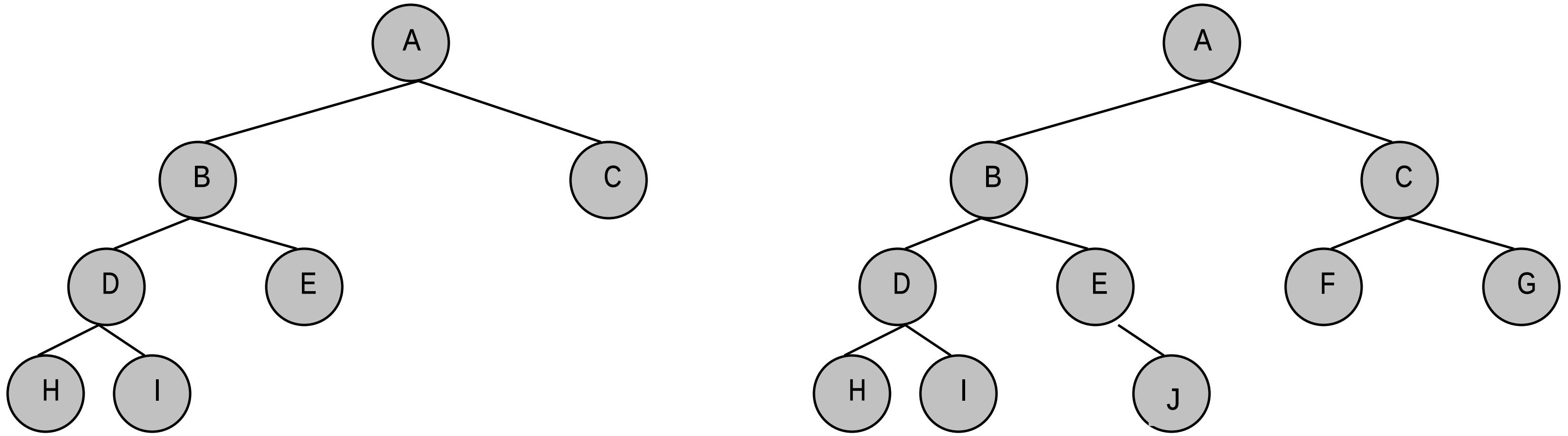
- complete binary tree
  - each level from 0 to n – 1 has all possible nodes (*full binary tree*)
  - all leaf nodes at depth n occupy leftmost position
  - number of nodes that occupy the first n – 1 levels
    - $2^n - 1$
  - possible number of nodes at level n
    - 1 to 2^n



Size 10
Depth 3

Types of Binary Trees

- non-complete binary trees

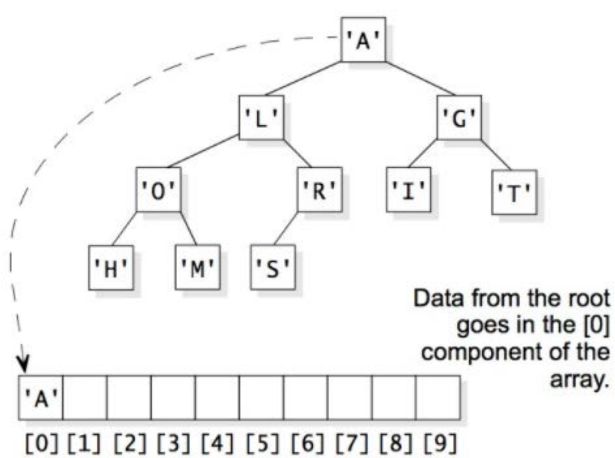


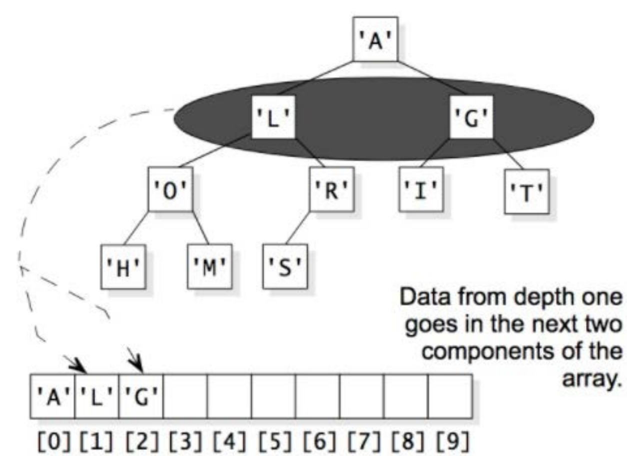
Size 7
Depth 3
Level 2 missing nodes

Size 10
Depth 3
Nodes at level 3 do not occupy leftmost position

Binary Tree Representation

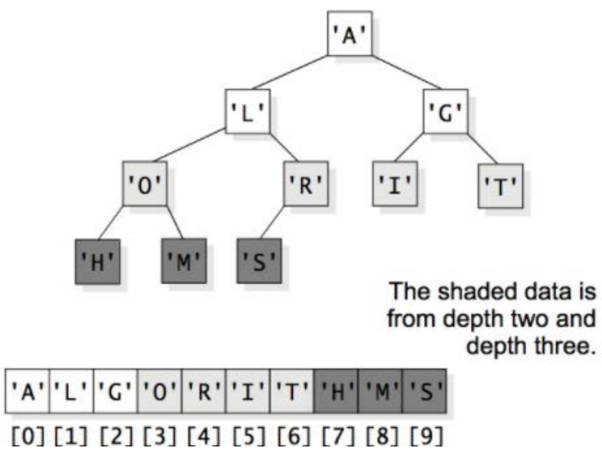
- complete binary tree can be represented using arrays





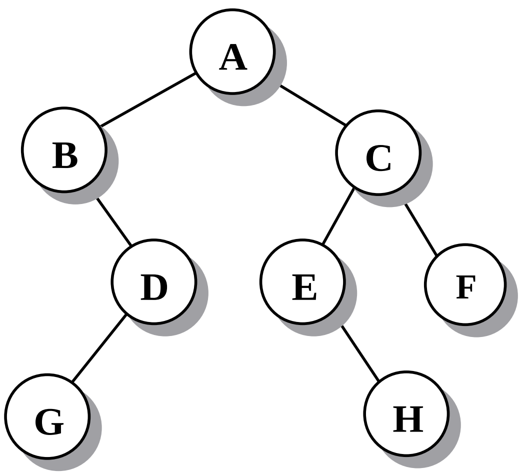
Binary Tree Representation

- complete binary tree can be represented using *dynamic* or *static* arrays
  - root in [0]
  - node data at [i]
    - parent at [(i-1)/2]
    - left child at [2i+1]
    - right child at [2i+2]

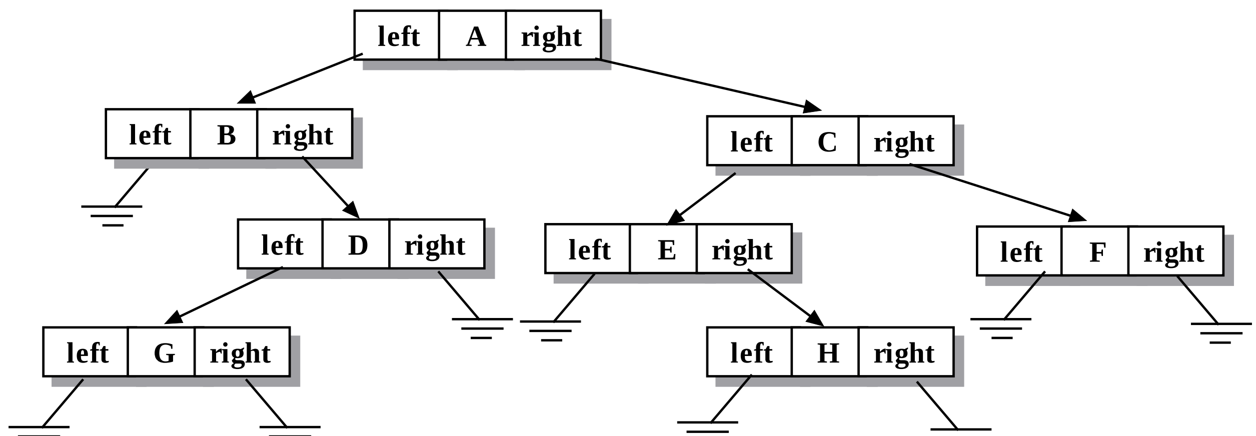


Binary Tree Representation

- *any* binary tree can be represented using connected node objects



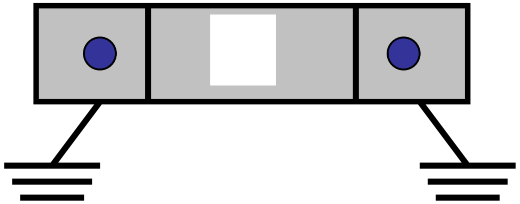
Abstract Tree Model



Tree Node Model

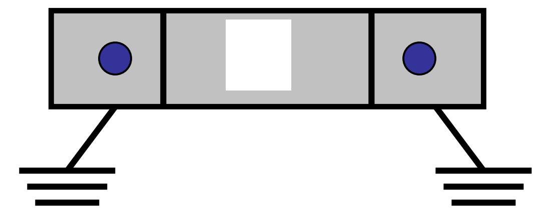
Binary Tree Node Definition

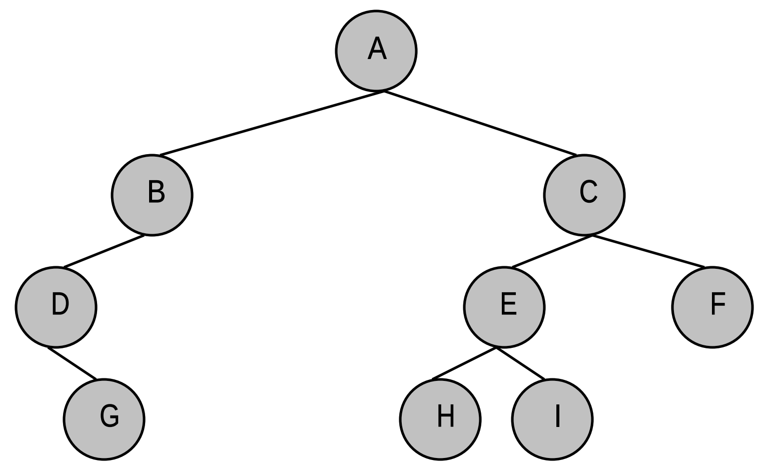
- Template binary_tree_node class
- Private data members include:
  - data_field
  - left_field
  - right_field
- Public methods include:
  - parameter constructor with defaults
  - accessor/mutator methods for private data
  - check for leaf status *(no children)*



Binary Tree Node Definition

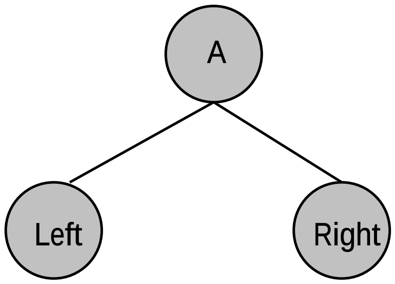
- Non-member toolkit functions:
  - tree_clear
  - tree_copy
  - tree_size
  - Tree Scanning
    - preorder
    - inorder
    - postorder
  - print
  - pretty_print





Binary Tree Scan Algorithms

- Since tree is non-linear structure, cannot scan nodes sequentially
- Recursive structure of binary tree allows traversal down subtrees
- Scanning strategies depend upon ordering of
  - Visit left child (*traverse left subtree*)
  - Perform action at node (*visit node*)
  - Visit right child (*traverse right subtree*)



Function Parameters

- Parameter to function can be function itself

```
ret_type parm_name(ptype_1, ptype_2, ...)
```

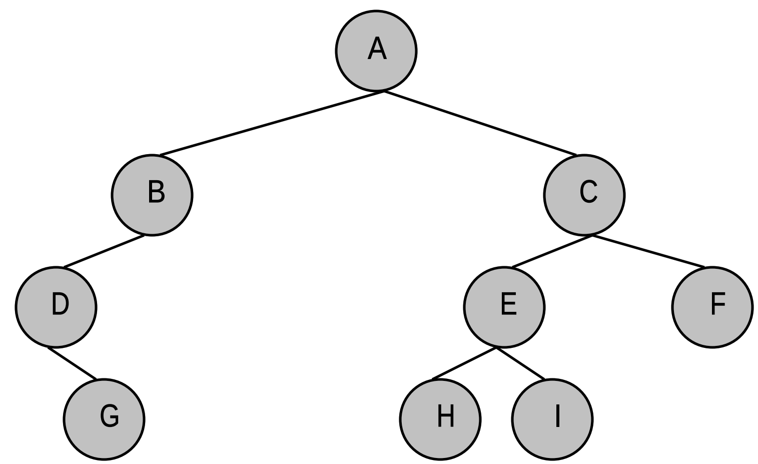
  - Return type
  - Parameter name
  - Parentheses with parameter type
- Binary Tree Scanning Algorithms
  - First parameter is void function with template parameter

```
void func_name(Item)
void func_name(Item&)
```

Binary Tree Scan Algorithms

- preorder (NLR→recursive)
  - visit node
  - traverse left subtree
  - traverse right subtree

pre-order (NLR)
A B D G C E H I F

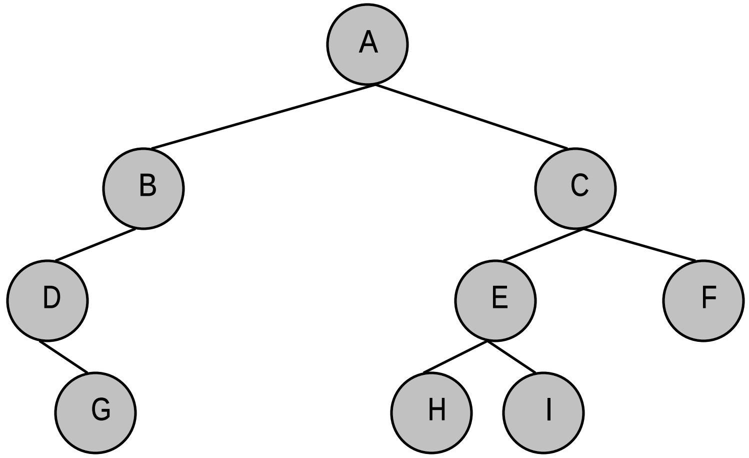


Binary Tree Scan Algorithms

- inorder (LNR→recursive)
  - traverse left subtree
  - visit node
  - traverse right subtree

in-order (LNR)

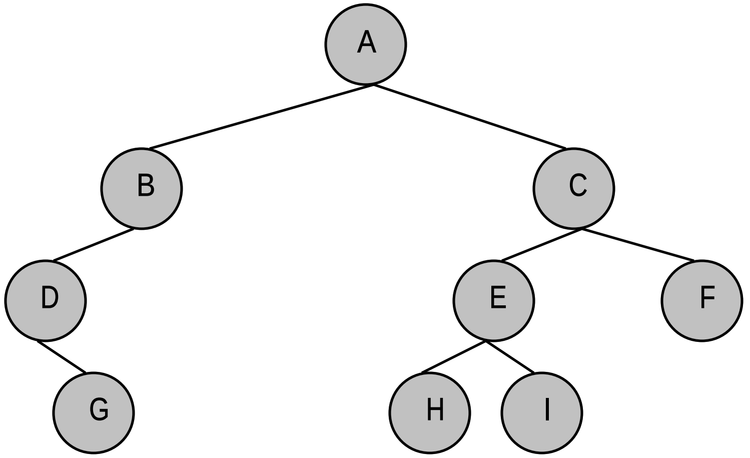
D G B A H E I C F



Binary Tree Scan Algorithms

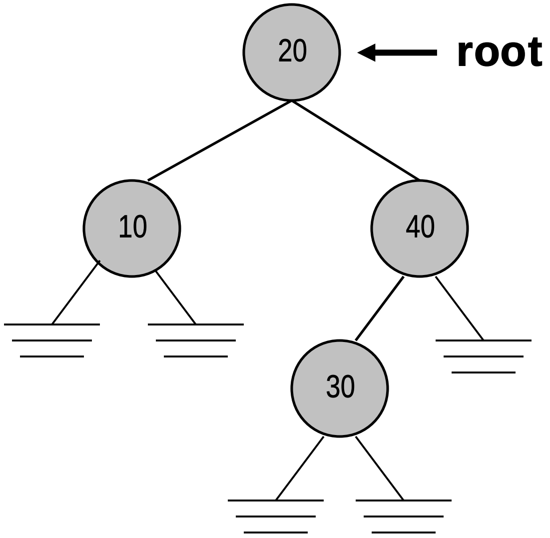
- postorder (LRN→recursive)
  - traverse left subtree
  - traverse right subtree
  - visit node

postorder (LRN)
G D B H I E F C A



Binary Search Tree

- Binary Tree whose ordered elements provide very efficient access and update operations
  - Elements ordered by relational operator < for the template type
    - data values in **left** subtree are less than or equal to value of the node
    - data values in **right** subtree are greater than value of node

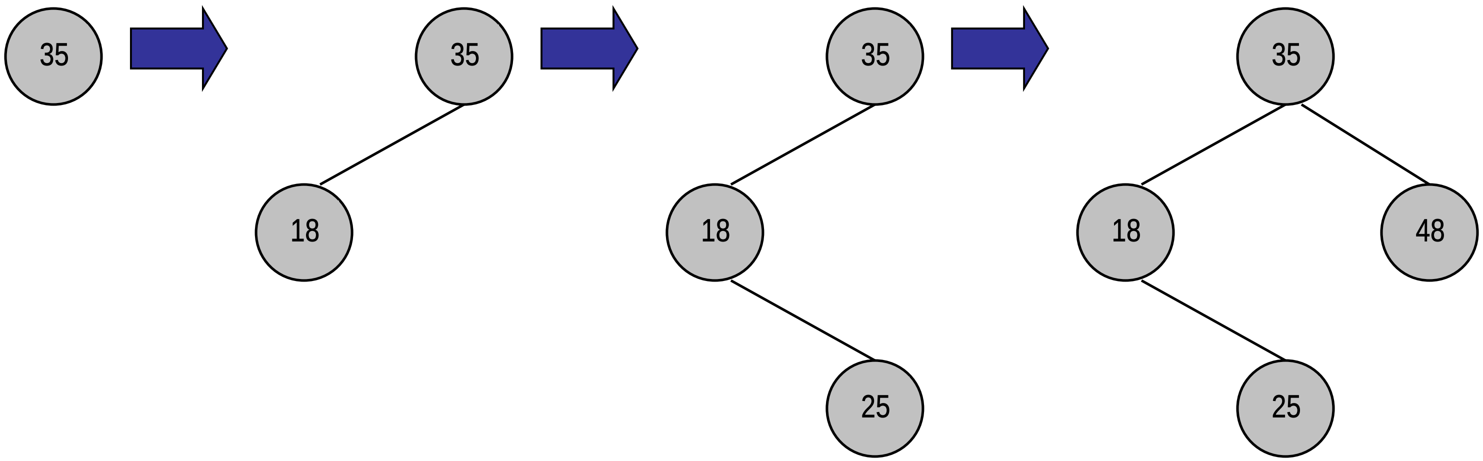


Building a BST

- elements enter BST via specific ordering strategy
- first element becomes root
- subsequent elements enter at end of path
  - if current >= new
    - go left
      - left empty, add node
      - left not empty, go left and repeat
  - if current < new
    - go right
      - right empty, add node
      - right not empty, go right and repeat

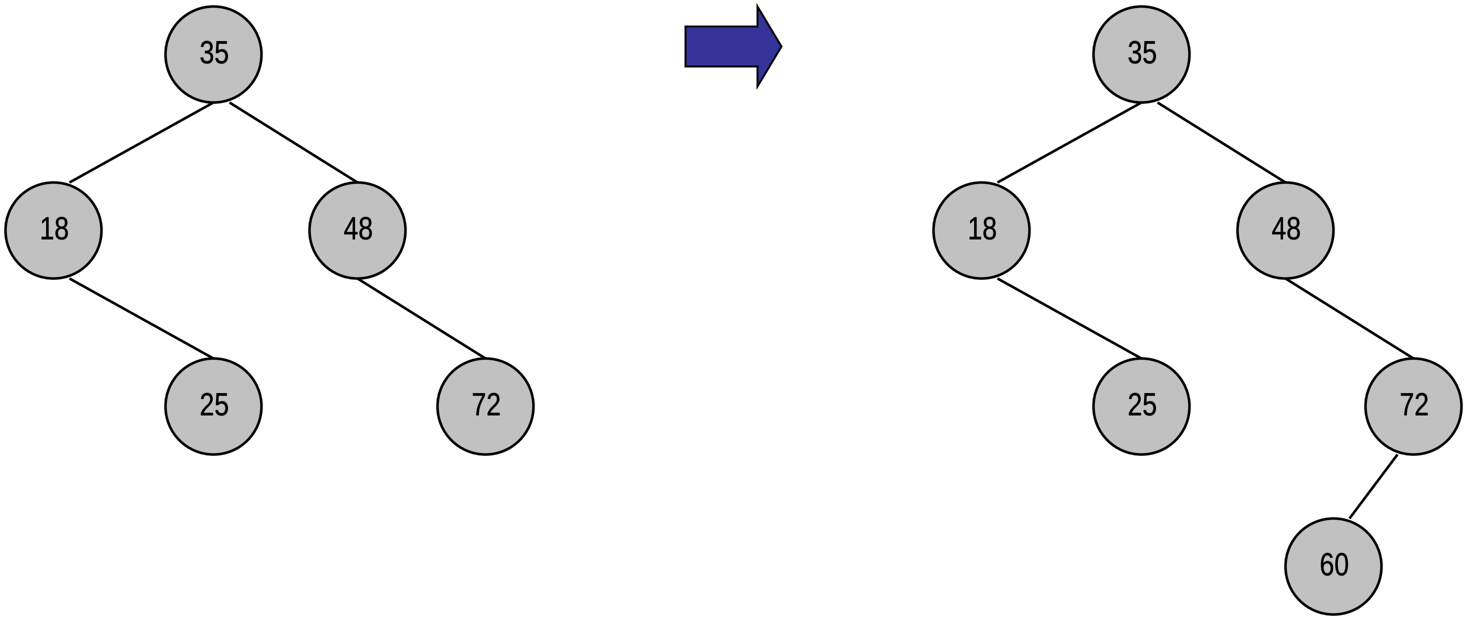
Building a BST

- Build binary search tree with elements
  - 35, 18, 25, 48, 72, 60



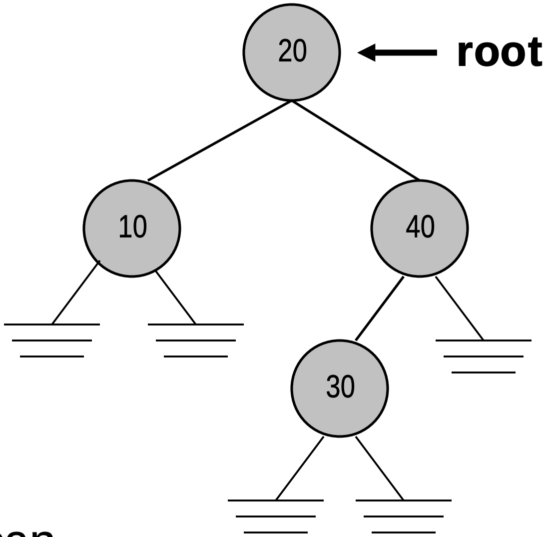
Building a BST

- Build binary search tree with elements
  - 35, 18, 25, 48, 72, 60



Binary Search Tree

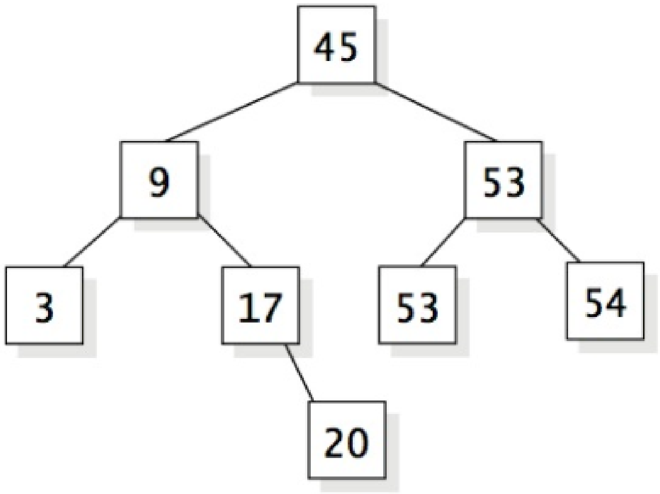
- Inorder scan (LNR) of BST ensures smaller nodes visited first
  - nodes visited in ascending order



Inorder Scan
10 20 30 40

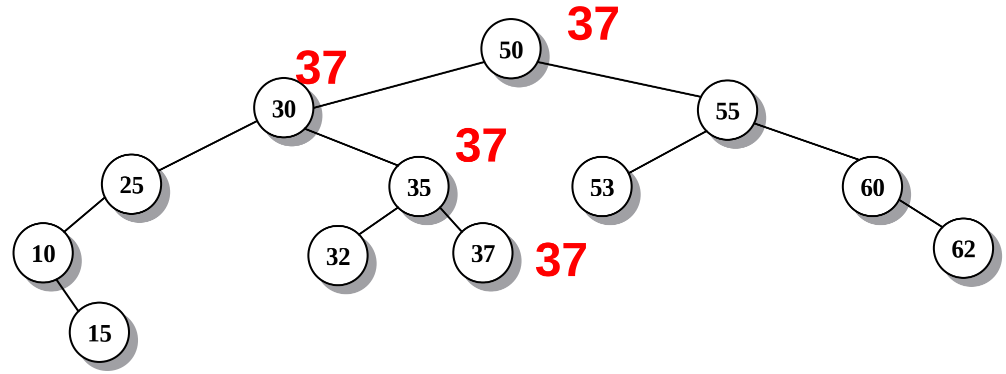
Binary Search Tree

- Improved search strategies for locating an element; path no greater than tree depth
  - scan left subtree if key is <=
    - duplicates found in left subtree
  - otherwise scan right subtree



Locating Item in BST

- template type has operators == and < defined
- compare item with node (loop until match or all nodes visited)
  - if item == current
    - match found
  - else
    - if item < current
      - go left
    - if item > current
      - go right



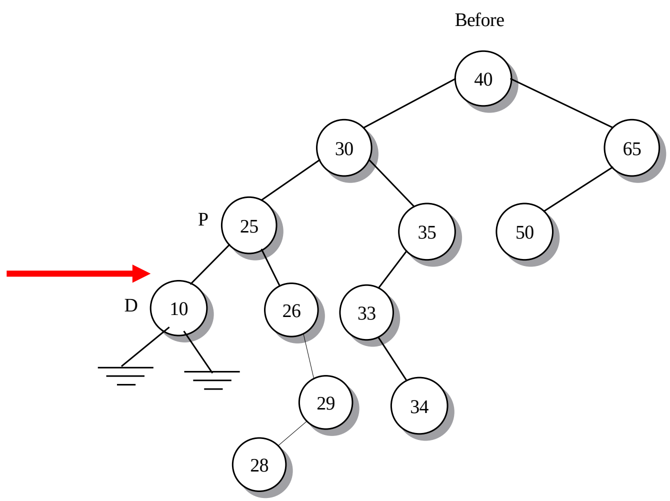
Current Node	Action (looking for node 37)
Root = 50	Compare items 37 and 50 37 <= 50, move to the left subtree
Node = 30	Compare items 37 and 30 37 > 30, move to the right subtree
Node = 35	Compare items 37 and 35 37 > 35, move to the right subtree
Node = 37	Compare items 37 and 37. Item found.

Removing Item from BST

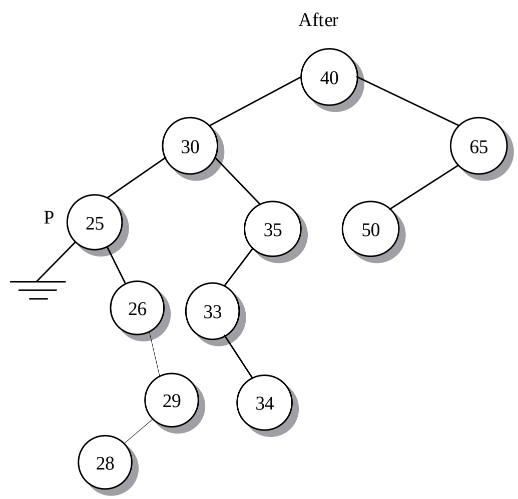
- must preserve BST node relationships
  - current == nullptr
    - not found
  - item == current
    - current_left == nullptr && current_right == nullptr
    - current_left == nullptr
      - new_root = current_right
    - current_left != nullptr
      - new_root = largest entry in left subtree
    - remove current
  - item < current
    - current = current_left, repeat
  - item > current
    - current = current_right, repeat

Removing Item From BST

- item == current
  - current_left == nullptr && current_right == nullptr
  - remove current



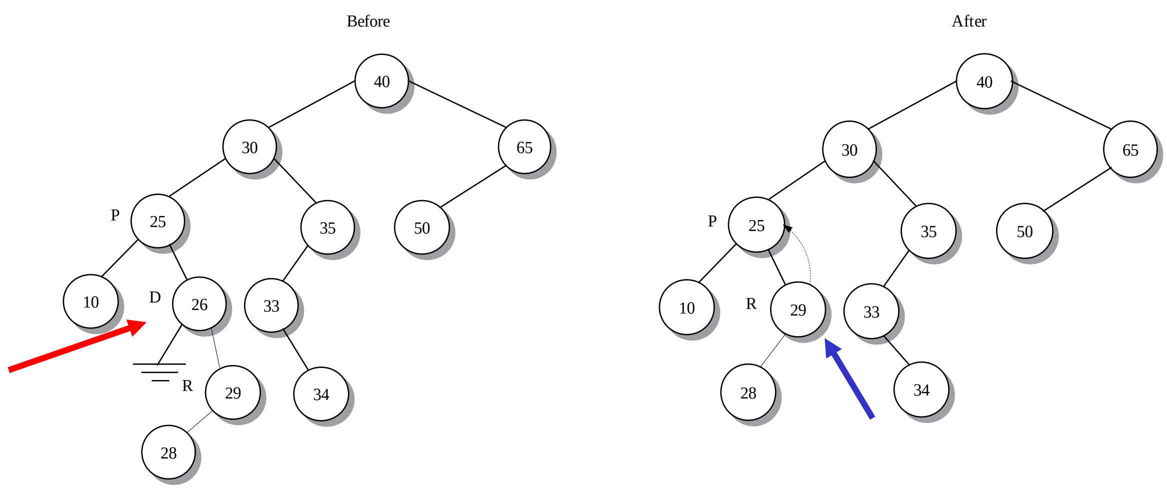
Delete leaf node 10.
pNodePtr->left is dNode



No replacement is necessary.
pNodePtr->left is NULL

Removing Item From BST

- item == current
  - current_left == nullptr
    - new_root = current_right
  - remove current



Delete node 26 with only a right child: Node R is the right child.

Attach node R to the parent.

Removing Item From BST

- item == current
  - current_left != nullptr
    - new_root = largest entry in left subtree
  - remove current