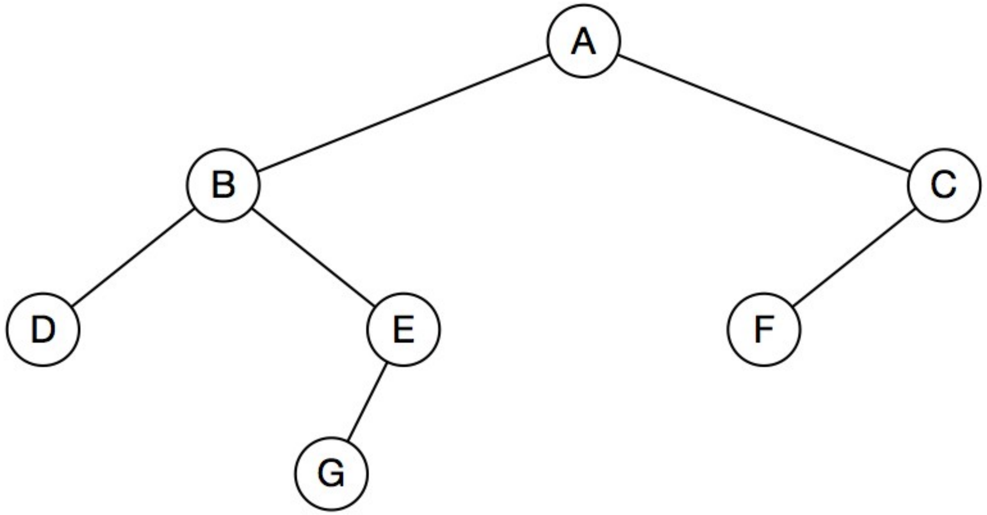
Trees

Tree Definitions and Properties

Example Tree

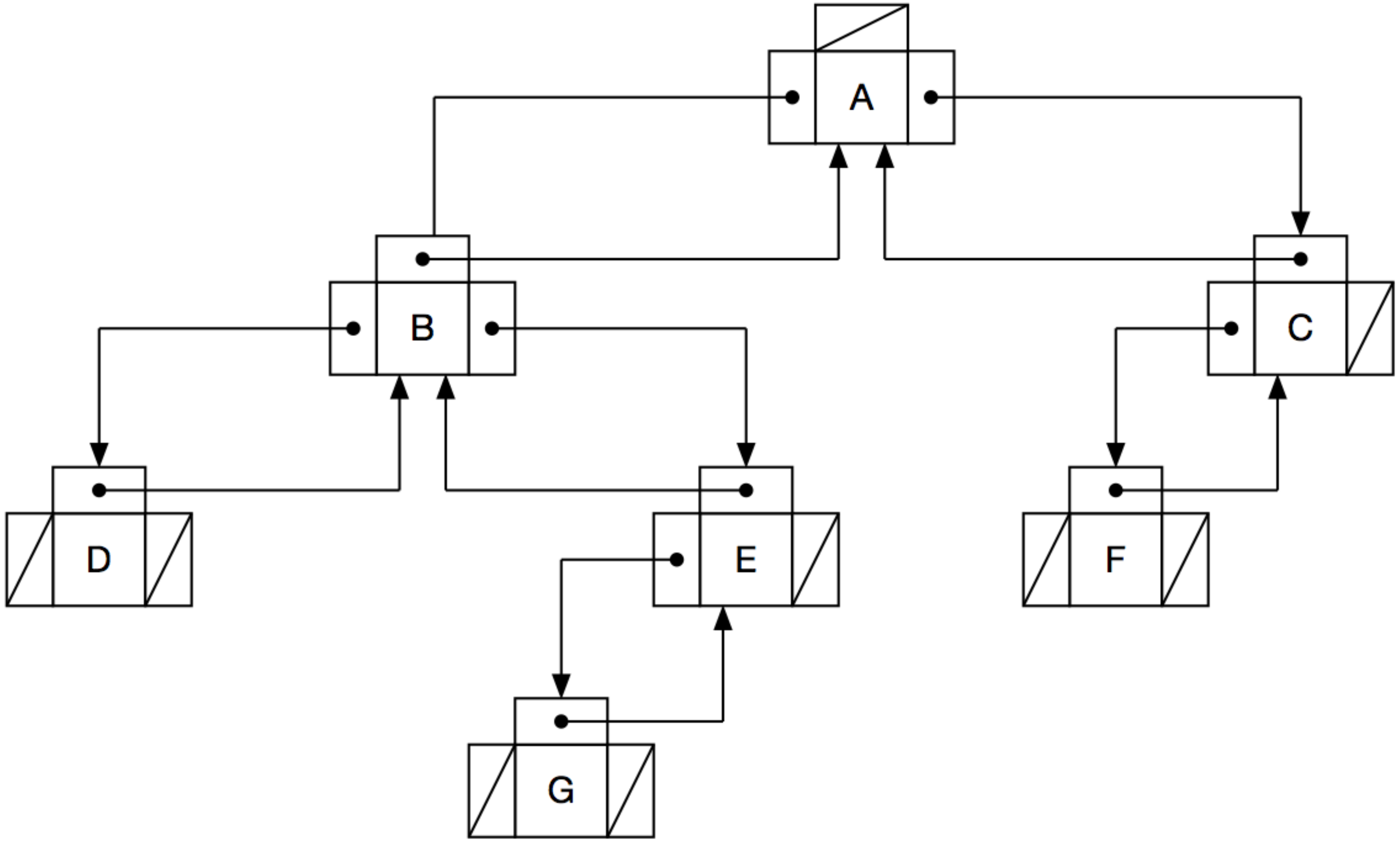


- A tree is an abstract data type that stores elements in a hierarchy.
- Every element, except the top, has a parent and zero or more children.
- The top element is called the *root*.
- A tree can be empty; no root, no parents, no children.
- A tree can have only one node—just a root.
- Children of the same parent are siblings.
- A node is called a *leaf* if it has no children.
- Nodes above leaves are called internal nodes.
- Nodes have *ancestors* and *descendants*
- A child is also a *subtree*

- An *edge* is a parent-child pair of nodes.
- An edge is implemented with links (pointers)
- A *path* is a connected set of edges—parent to child to grandchild and so on.
- Tree nodes have:
  - keys that identify them
  - associated data
- Trees are ordered if there is a linear ordering (by key) of the children of each node.
- All really useful trees are ordered—it makes them searchable.

Binary Tree

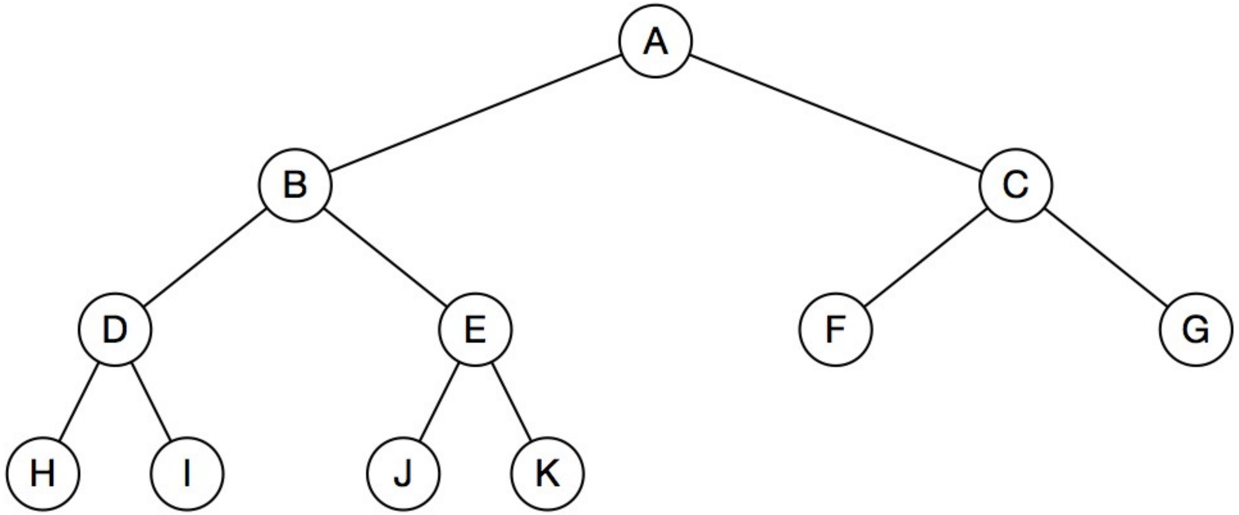
- Every node has 0, 1, or 2 children
- A child is a left child or a right child
- Tree is ordered: left children precede right children
- There are left subtrees and right subtrees


A
B
C
D
E
F
G

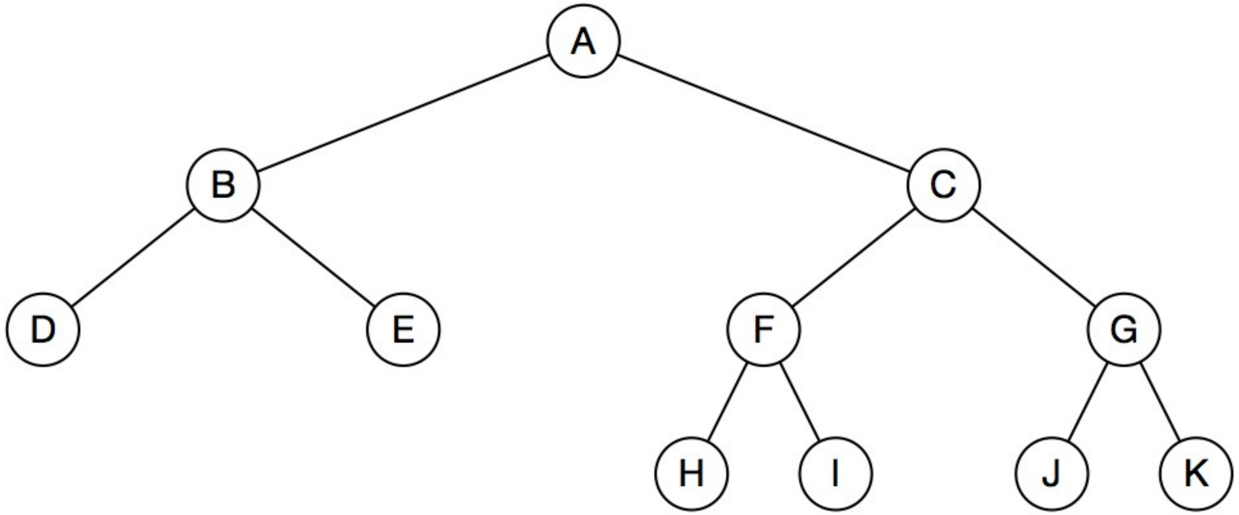
Types of Trees

- **Full**:
  - every node contains 0 or 2 children.
- **Complete:**
  - all levels, except possibly the last level, are completely full
  - all nodes in the last level are as far left as possible.
- **Perfect**:
  - all internal nodes have 2 children
  - all leaf nodes are at the same level

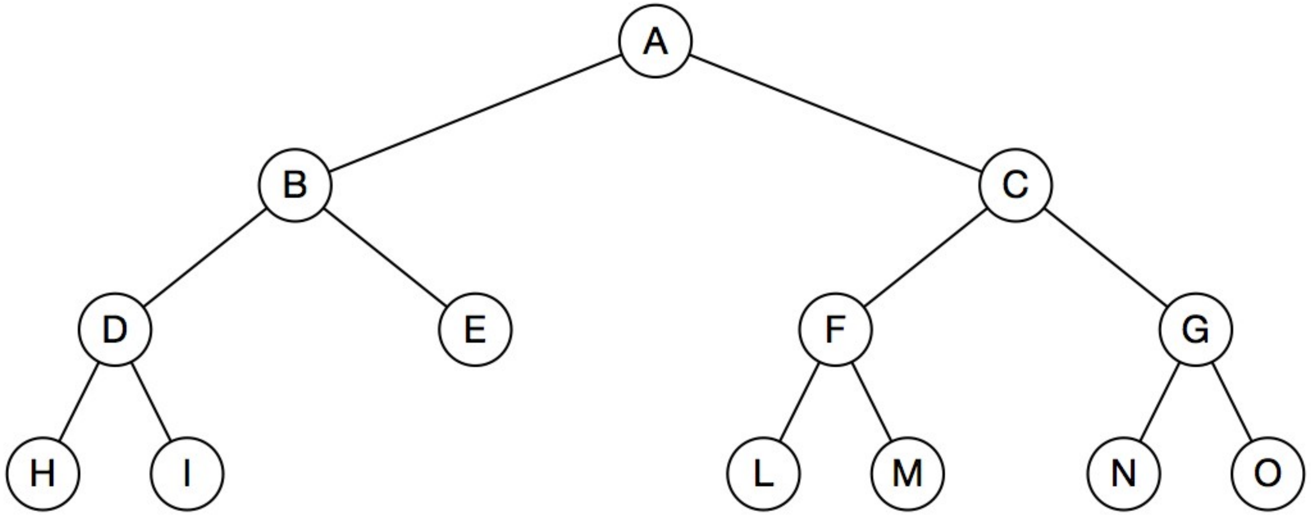
Complete Tree



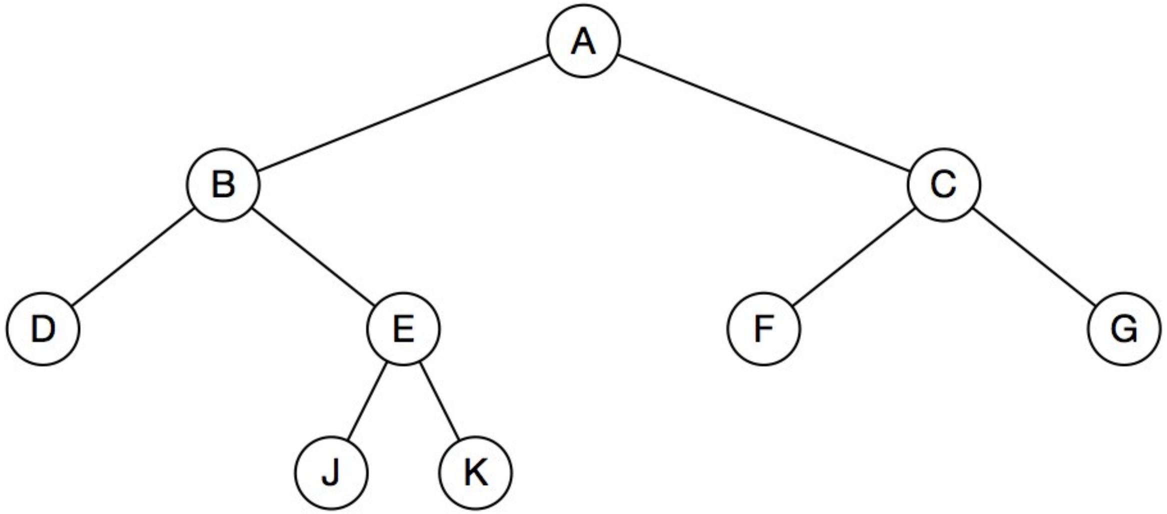
Full Tree



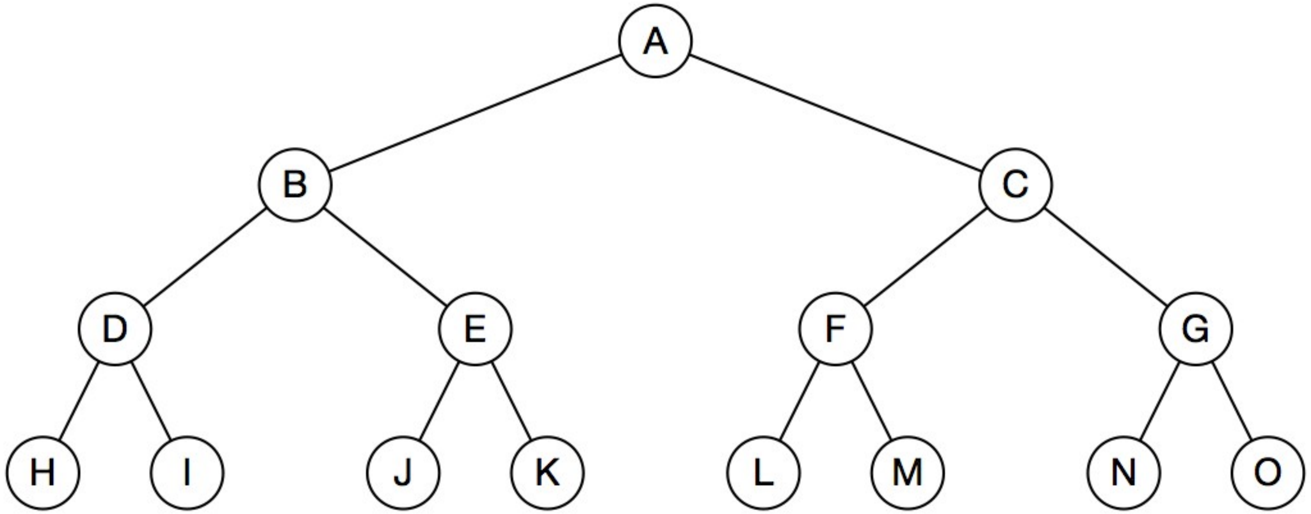
Full Tree



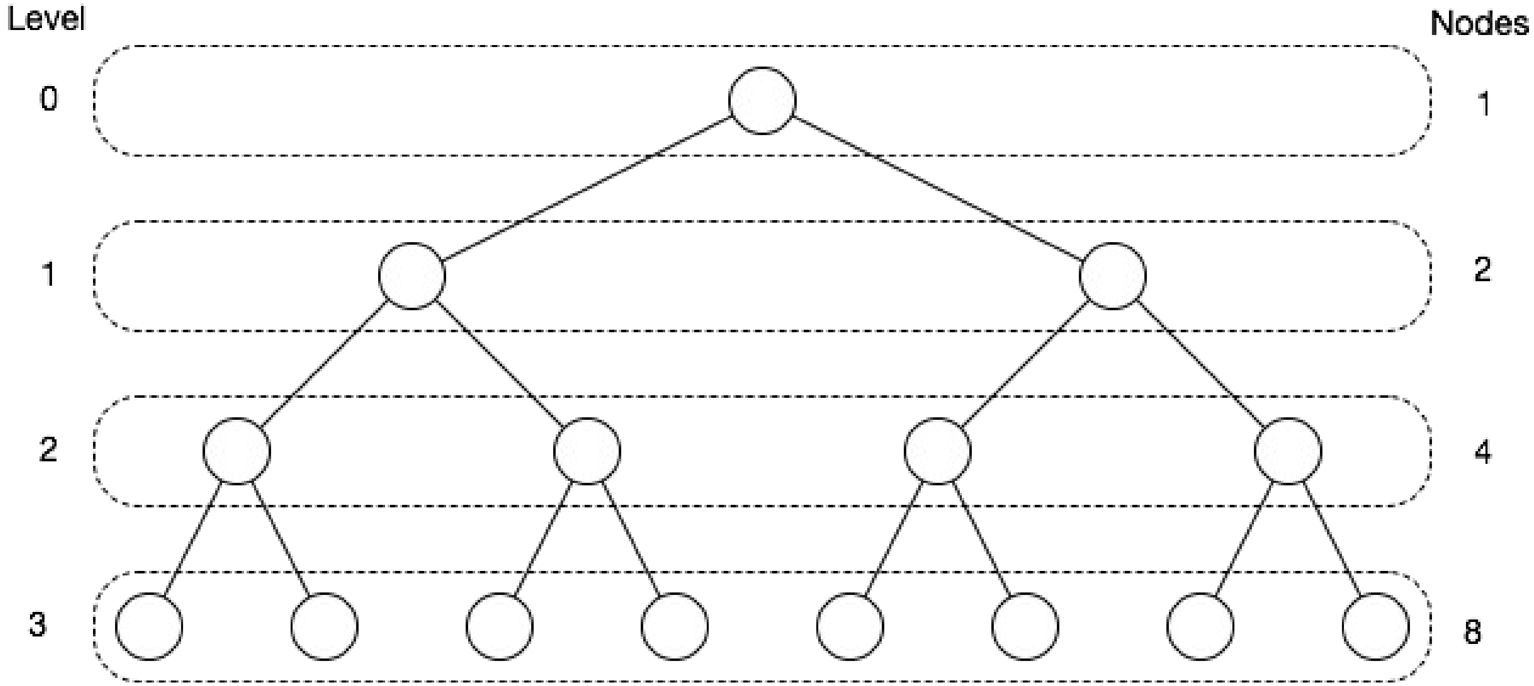
Full Tree



Perfect Tree



Properties of Binary Trees



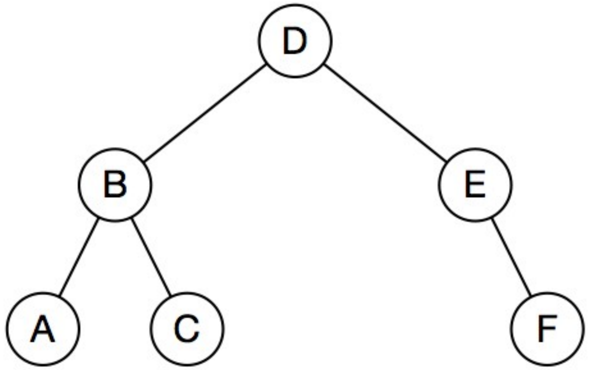
nodes = n = 15;

h = height = 4

h = ceiling(log$_2$(n)) = ceiling(log$_2$(15)) = ceiling(3.9) = 4

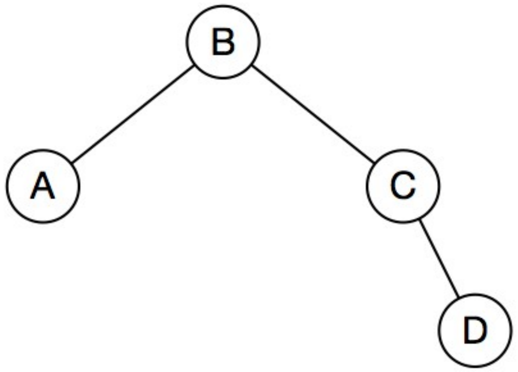
Binary Search Trees

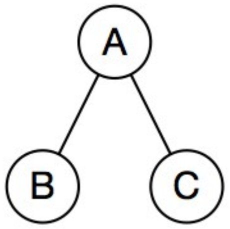
Example Search Tree

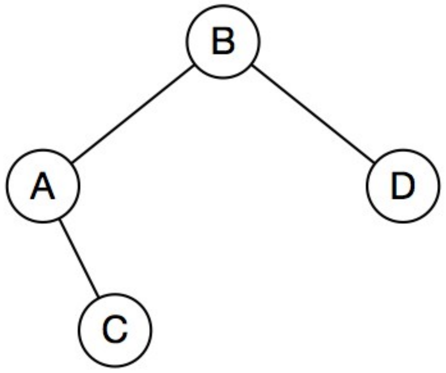


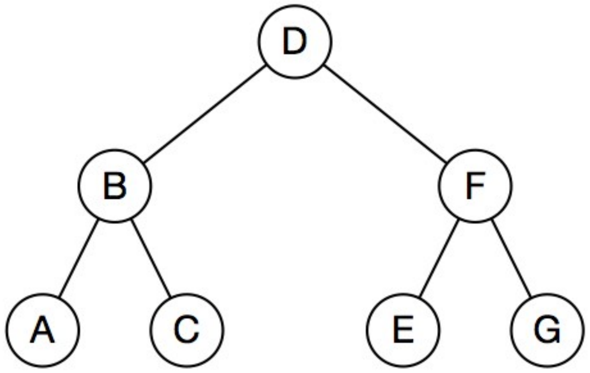
- Trees are made up of smaller trees—subtrees. Node B is the root of a subtree; so is A.
- Internal nodes (circles) and leaves (boxes) contain keys and values; keys are node identifiers that are searched for.
- Left subtree’s keys are less than the root’s key, right subtree’s keys are greater.

Examples: Binary Search Tree, Yes or No?



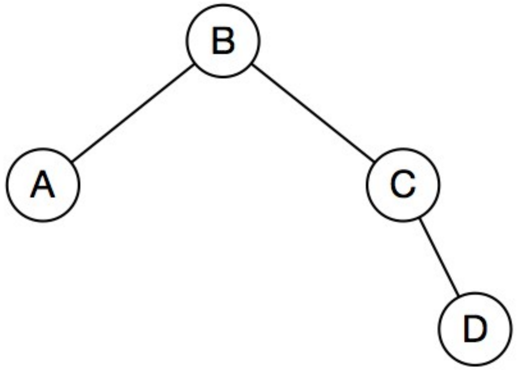




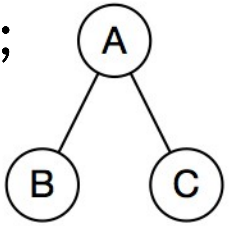


Examples: Some Are Binary Search Trees, Some Are Not

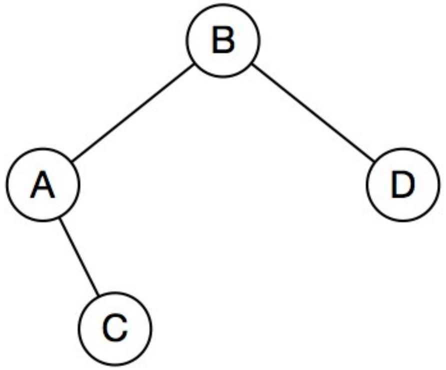
BST



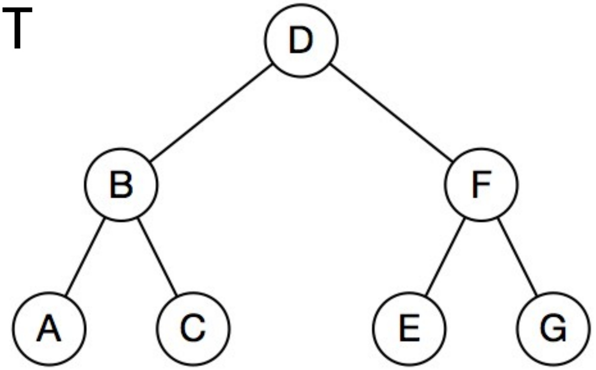
Not BST;
B > A



Not BST;
C > B



BST



C++ Implementation

See implementation examples

Tree Traversal Algorithms

Traversal Algorithms

- Traversals are systematic ways to “visit” tree nodes and their children.
- A “visit” accesses a node’s data, perhaps to display it or return it.
- Three algorithms:
  - Preorder: visit a node, then its left and right children
  - Postorder: visit a node’s left and right children, then the node itself
  - Inorder: visit a node's left child, the the node itself, then the node's right child.