Data Structure

Lecture#10: Binary Trees (Chapter 5)

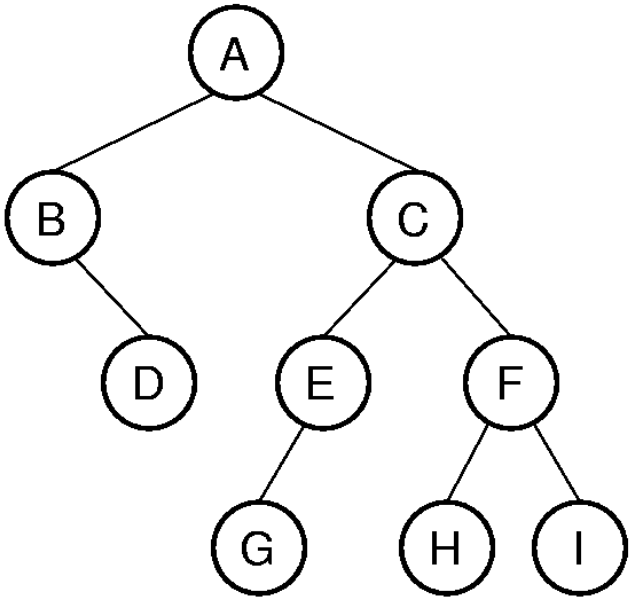
U Kang
Seoul National University

In This Lecture

- The concept of binary tree, its terms, and its operations
- Full binary tree theorem
- Idea and implementation of traversals for tree

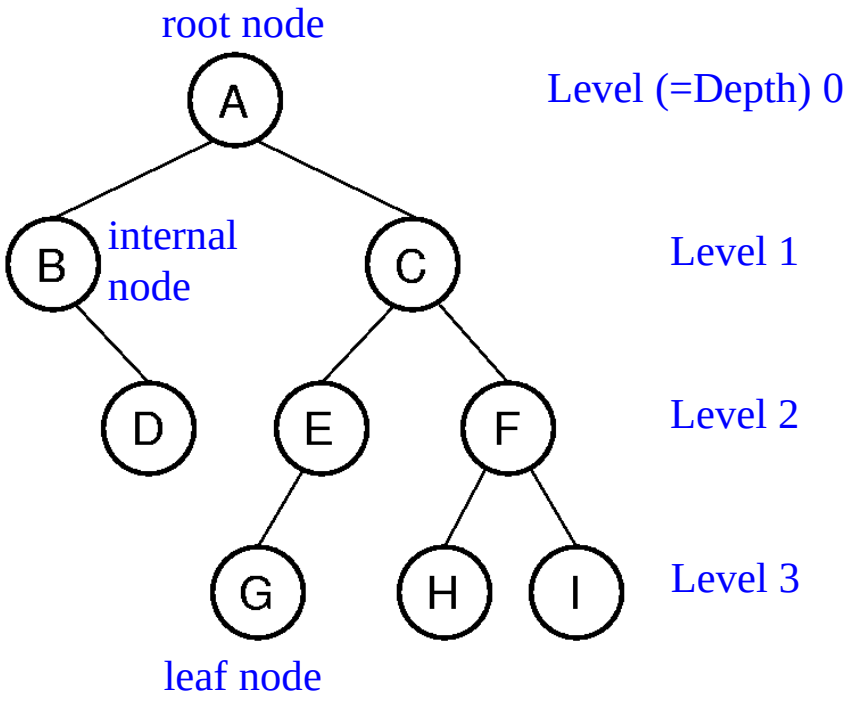
Binary Trees

- A binary tree is made up of a finite set of nodes that is either empty or consists of a node called the root together with two binary trees, called the left and right subtrees, which are disjoint from each other and from the root.



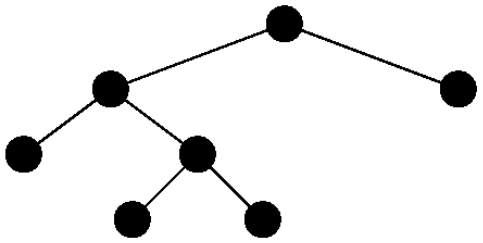
Binary Tree Example

- Notation: node, children, edge, parent, ancestor, descendant, path, depth, height, level, root node, leaf node, internal node, subtree.

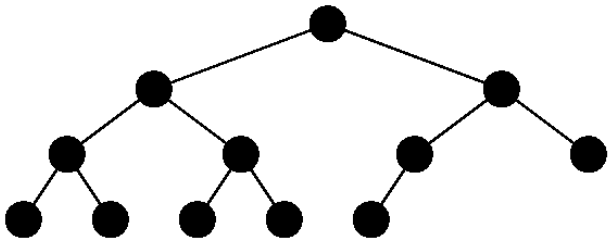


Full and Complete Binary Trees

- Full binary tree: Each node is either a leaf or internal node with exactly two non-empty children.
- Complete binary tree: If the height of the tree is *d*, then 1) all levels except possibly level *d-1* are completely full, and 2) the bottom level has all nodes to the left side.



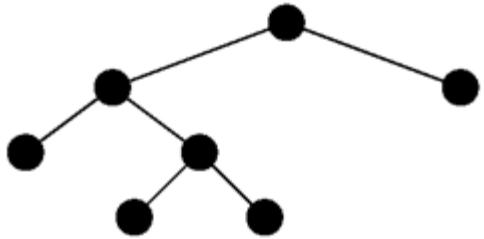
Full binary tree



Complete binary tree

Full Binary Tree Theorem (1)

- **Theorem**: The number of leaves in a non-empty full binary tree is one more than the number of internal nodes.



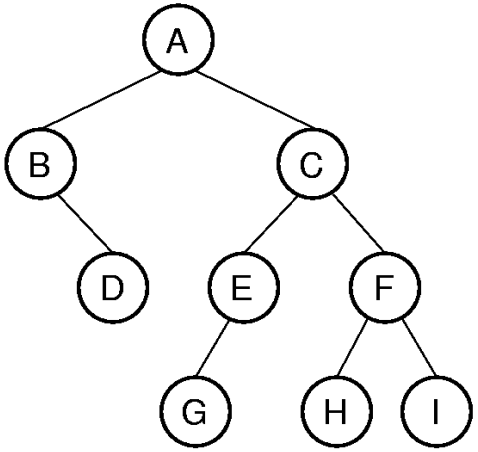
- **Proof** (by Mathematical Induction):
- **Base case**: A full binary tree with 1 internal node must have two leaf nodes.
- **Induction Hypothesis**: Assume any full binary tree **T** containing *n*-1 internal nodes has *n* leaves.

Full Binary Tree Theorem (2)

- **Induction Step**: given tree **T** with n internal nodes, pick internal node *I* with two leaf children. Remove *I*'s children, call resulting tree **T'**.
- By induction hypothesis, **T'** is a full binary tree with n leaves.
- Restore *I*'s two children. The number of internal nodes has now gone up by 1 to reach *n*. The number of leaves has also gone up by 1.

Full Binary Tree Corollary

- **Theorem**: The number of null pointers in a non-empty tree is one more than the number of nodes in the tree.



# of null pointers = 10
# of nodes = 9

- **Proof**: Replace all null pointers with a pointer to an empty leaf node. This is a full binary tree.

Binary Tree Node Class

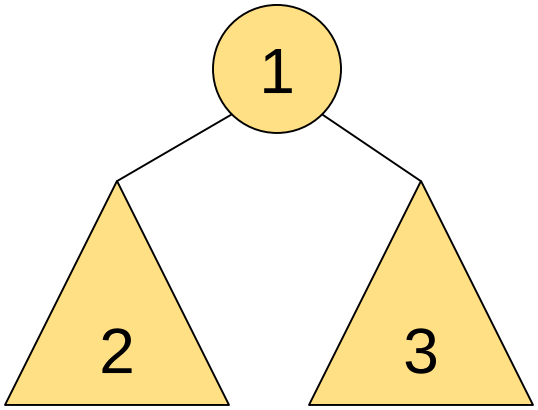
```
/** ADT for binary tree nodes */
public interface BinNode<E> {
  /** Return and set the element value */
  public E element();
  public E setElement(E v);

  /** Return the left child */
  public BinNode<E> left();

  /** Return the right child */
  public BinNode<E> right();

  /** Return true if this is a leaf node */
  public boolean isLeaf();
}
```

Traversals (1)

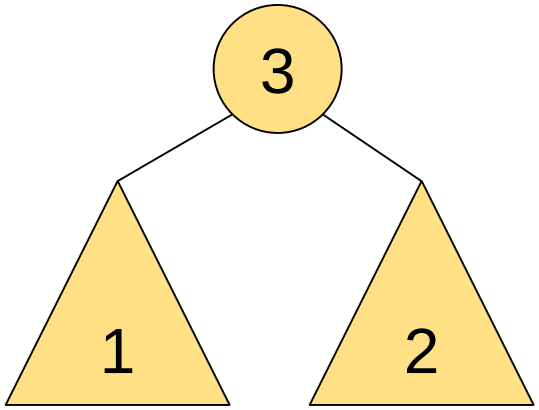
- Any process for visiting the nodes in some order is called a traversal.
- Any traversal that lists every node in the tree exactly once is called an enumeration of the tree's nodes.

Traversals (2)

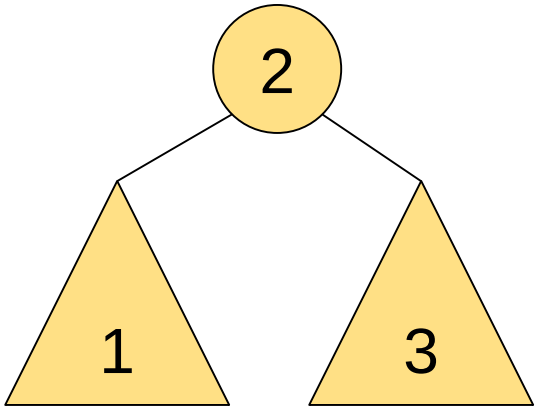
- Preorder traversal: visit each node before visiting its children.
- Postorder traversal: visit each node after visiting its children.
- Inorder traversal: visit the left subtree, then the node, then the right subtree.
- Apply the rule recursively inside children



Preorder Traversal



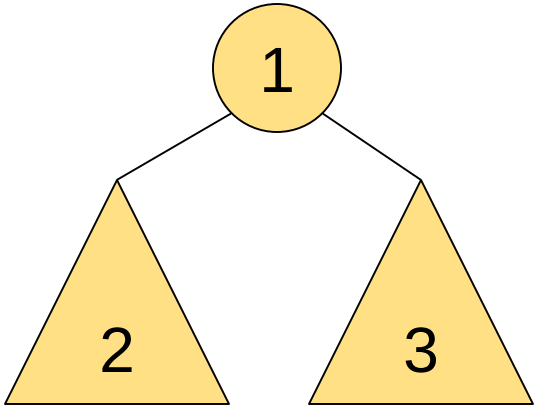
Postorder Traversal



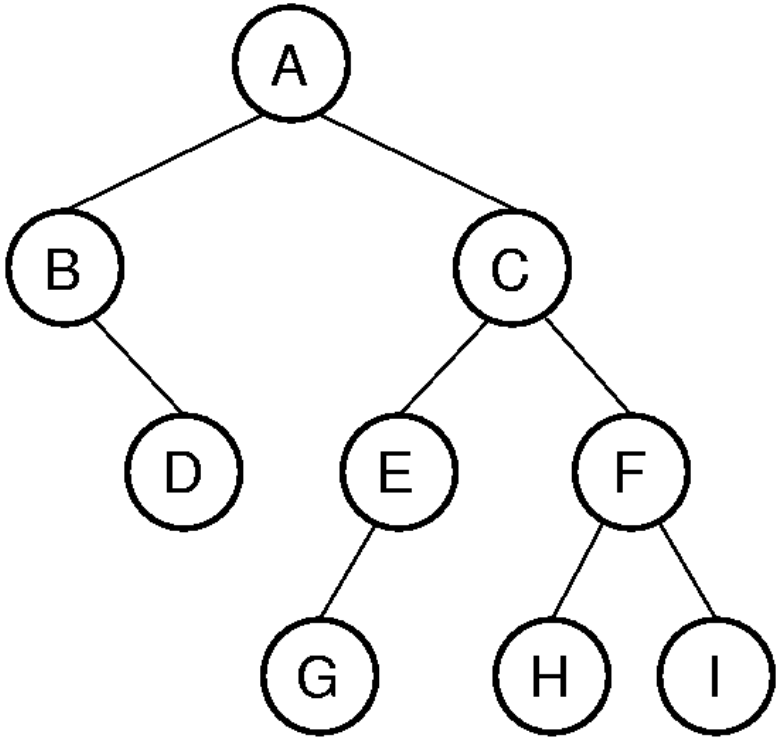
Inorder Traversal

Preorder

- Preorder traversal: visit each node before visiting its children.
  - E.g.) ?

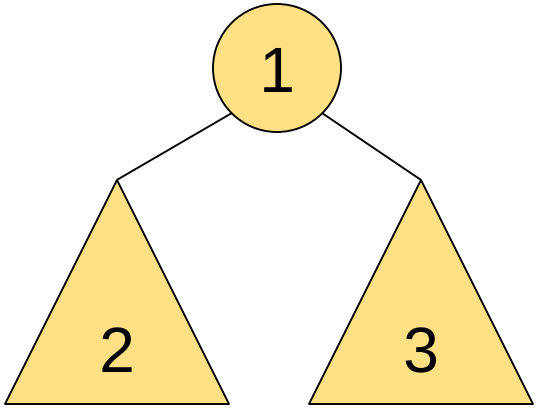


Preorder Traversal

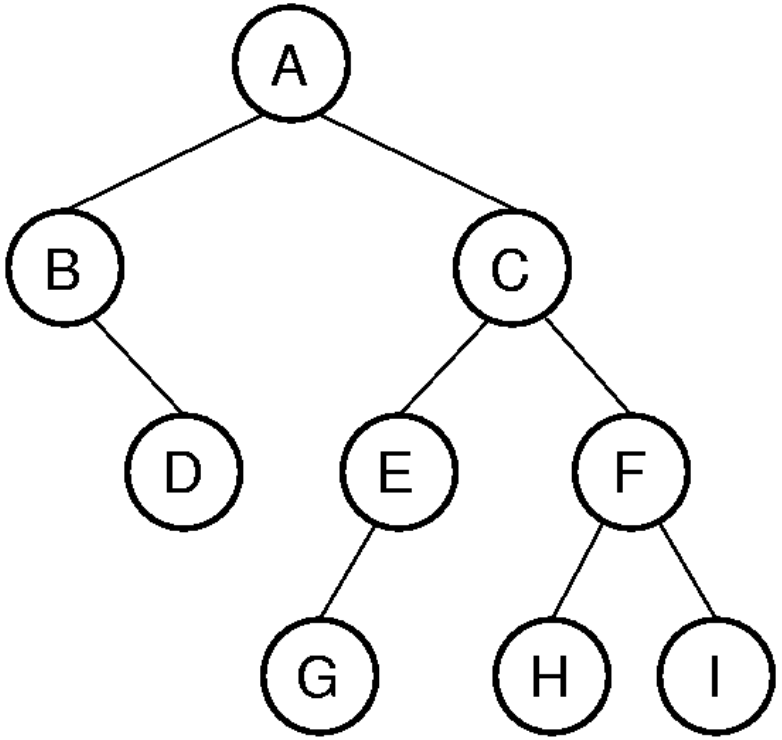


Preorder

- Preorder traversal: visit each node before visiting its children.
  - E.g.) ABDCEGFHI

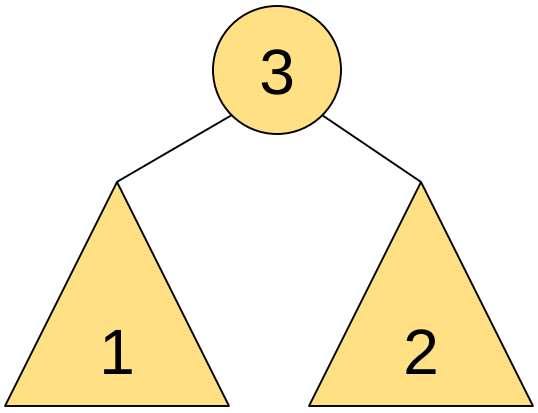


Preorder Traversal

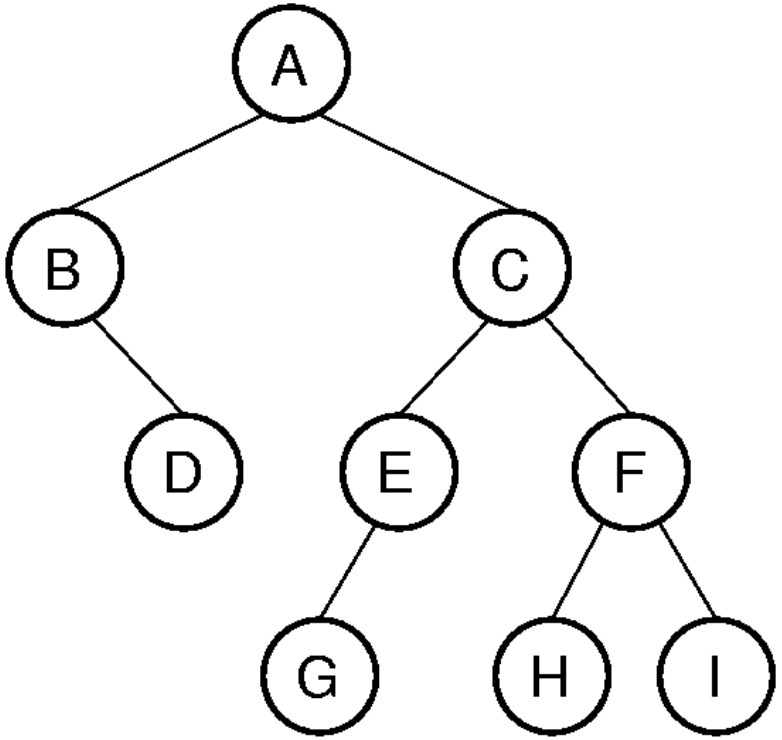


Postorder

- Postorder traversal: visit each node after visiting its children.
  - E.g.) ?

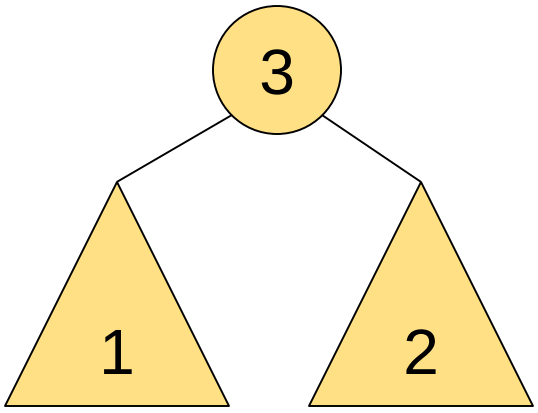


Postorder Traversal

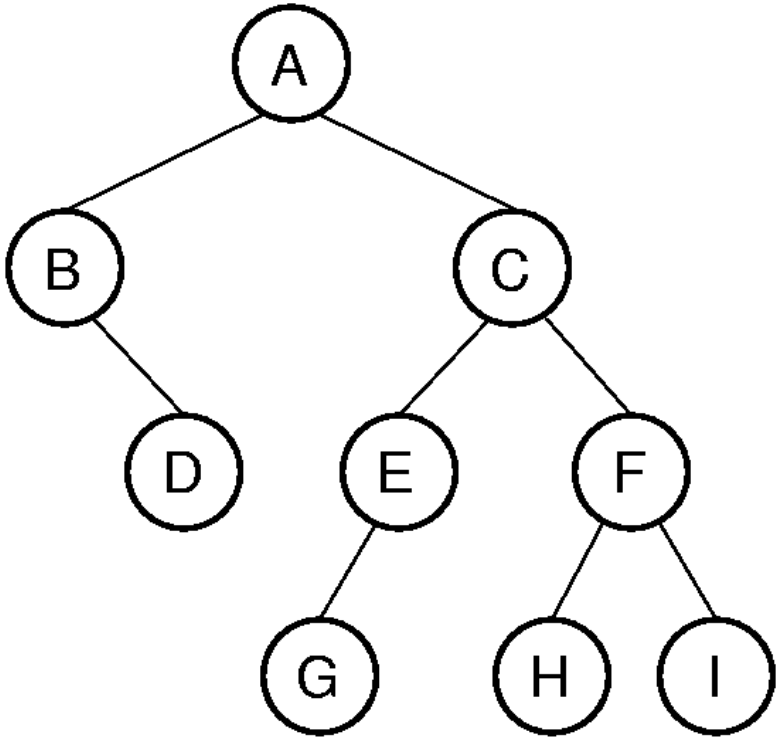


Postorder

- Postorder traversal: visit each node after visiting its children.
  - E.g.) DBGEHIFCA

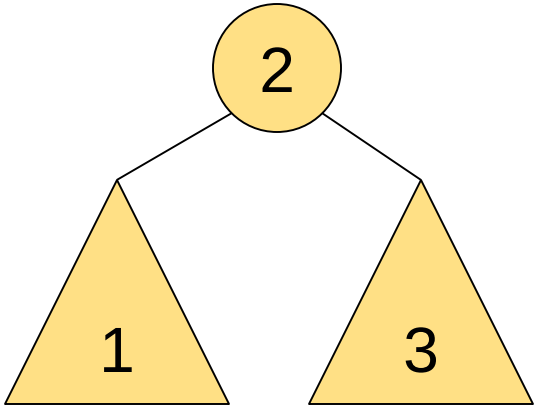


Postorder Traversal

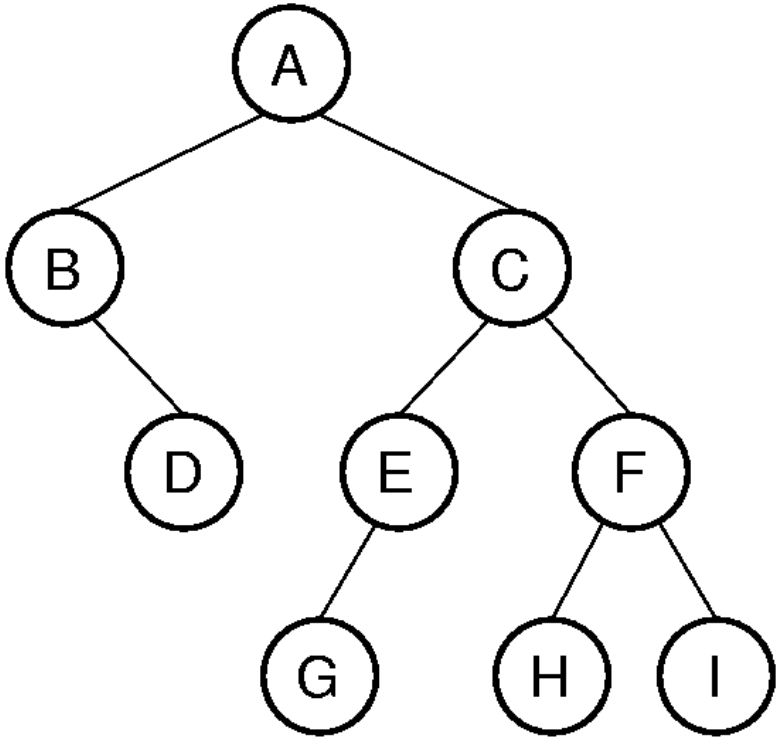


Inorder

- Inorder traversal: visit the left subtree, then the node, then the right subtree.
  - E.g.) ?

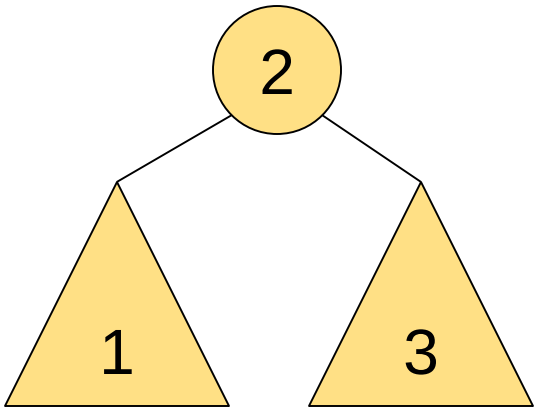


Inorder Traversal

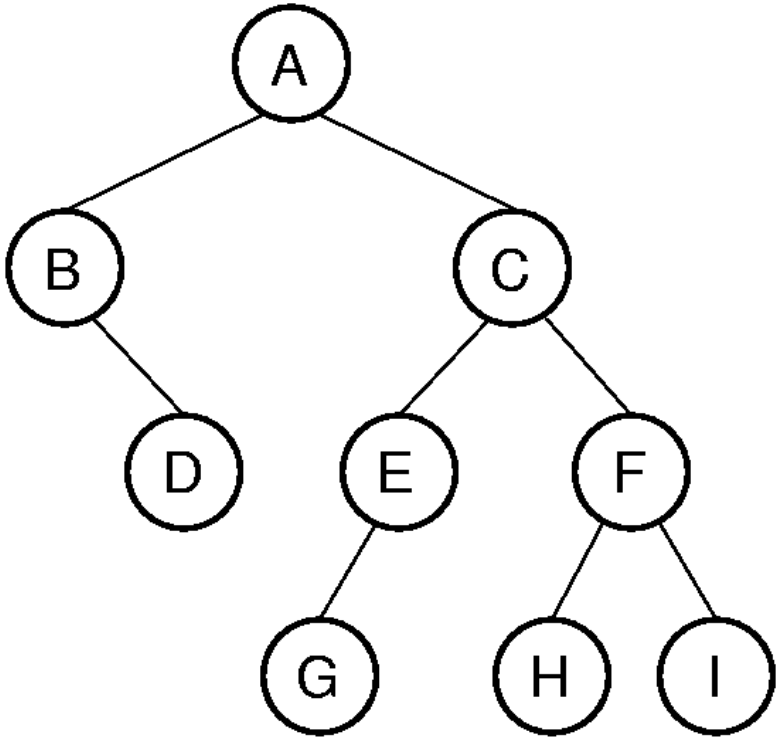


Inorder

- Inorder traversal: Visit the left subtree, then the node, then the right subtree.
  - E.g.) BDAGECHFI



Inorder Traversal



Implementing Traversals

```
/** @param rt The root of the subtree */
void preorder(BinNode rt)
{
  if (rt == null) return; // Empty subtree
  visit(rt);
  preorder(rt.left());
  preorder(rt.right());
}

// This implementation is error prone
void preorder2(BinNode rt) // Not so good
{
  visit(rt);
  if (rt.left() != null)  preorder2(rt.left());
  if (rt.right() != null) preorder2(rt.right());
}
```

What you need to know

- The concept of binary tree, its terms, and its operations
- Idea and proof of full binary tree theorem and its corollary
- How to perform three main traversals for a given tree; how to implement the traversals

Questions?