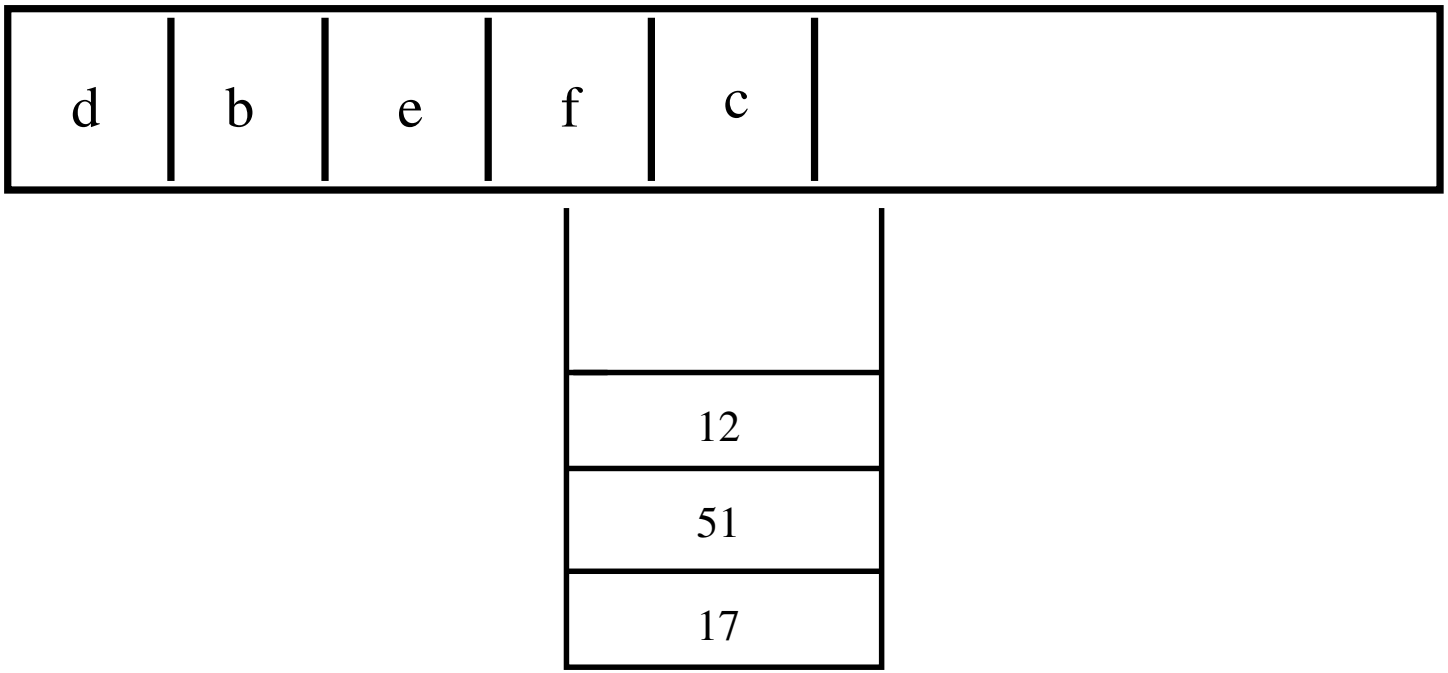
Binary Tree

Daniel Archambault

Previously in CS-115



First two ADTs and their implementations.

Previously in CS-115

- What information does a queue store? What are its four operations?

Previously in CS-115

- What information does a queue store? What are its four operations?
- What information does a stack store? What are its four operations?

Previously in CS-115

- What information does a queue store? What are its four operations?
- What information does a stack store? What are its four operations?
- Is it easier to implement a queue with a linked list?

Previously in CS-115

- What information does a queue store? What are its four operations?
- What information does a stack store? What are its four operations?
- Is it easier to implement a queue with a linked list?
- What is the easiest way to implement a stack?

Previously in CS-115

- Now it's time for the tree data structure

Trees

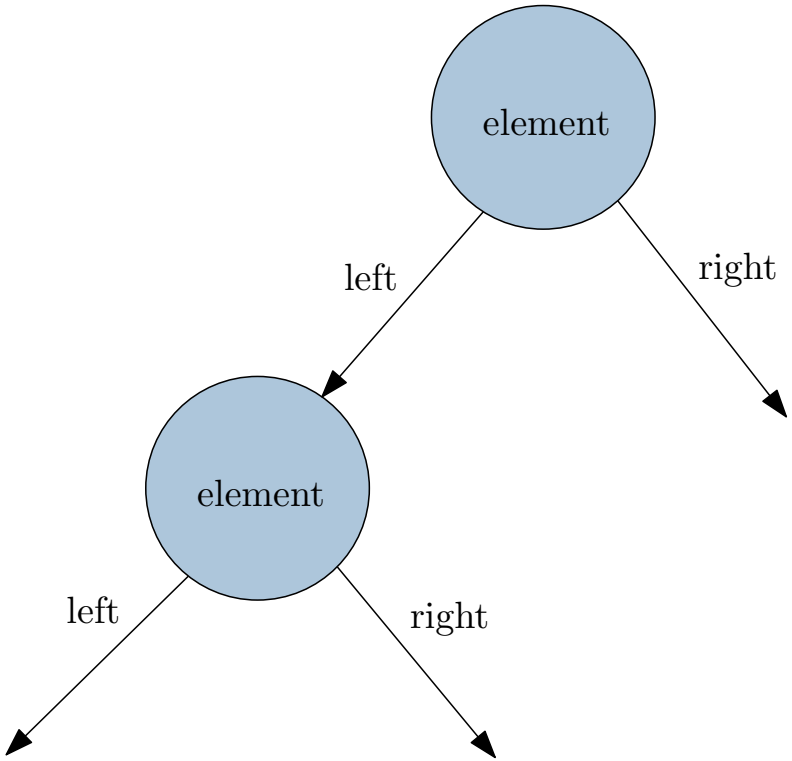
Binary Tree ADT

- A binary tree is an ADT
- Can be expressed as an array
- Can be expressed as a linked structure
- We will talk about linked structure in this class

Definition

- A node is an entity of the tree which stores data
- An edge is a reference to a node
- A binary tree is a data structure with:
  - A single node r, the root of the tree with two children
  - Each node has one parent
  - Each node has two children

Implementation of Node in Java



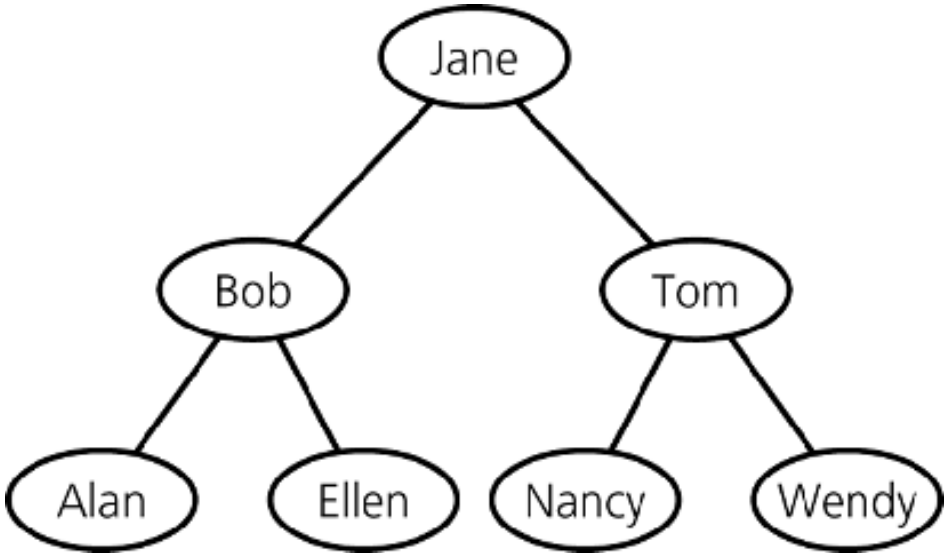
- `Object` implementation

```
public class Node
{
   private Node left;
   private Node right;
   private Object element;
}
```

- Generic implementation

```
public class Node<T>
{
   private Node<T> left;
   private Node<T> right;
   private T element;
}
```

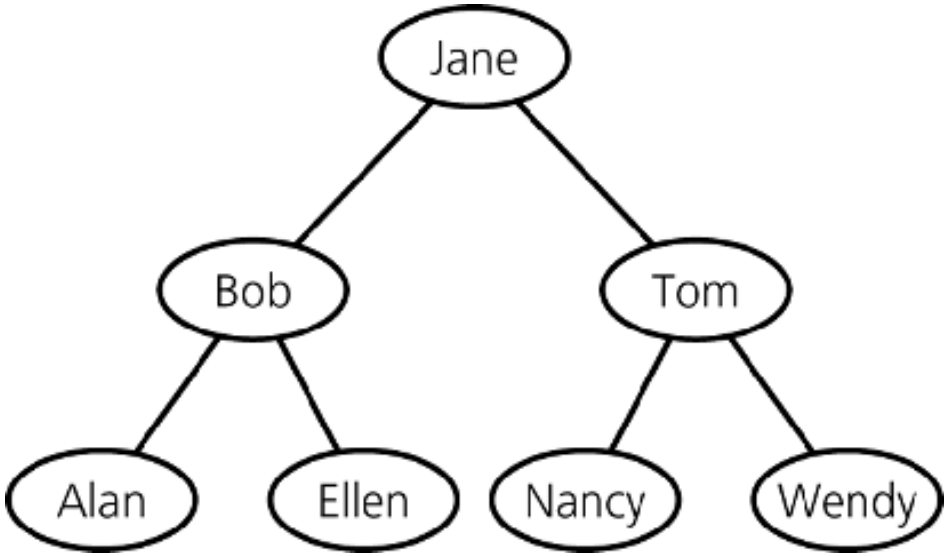
Leaf and Interior Node



Data Abstraction and Problem Solving with Java

- A leaf has zero children
  - Allen, Ellen, Nancy, and Wendy
- An interior node has one or more children:
  - Jane, Bob, and Tom
- The root has no parent: Jane
- Interior nodes have subtrees
  - The tree which exists below them
  - Bob, Allen, Ellan is a subtree below Jane

Binary Tree



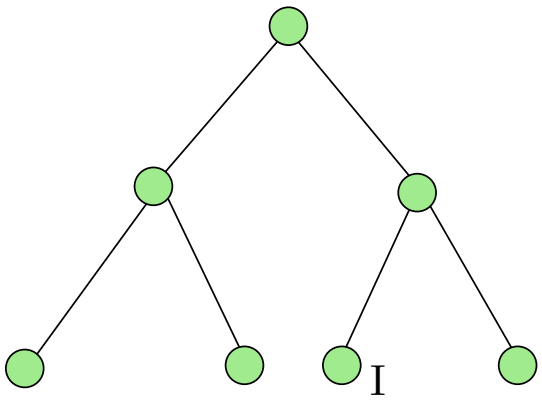
Data Abstraction and Problem Solving with Java

- A tree where each node has at most two children
- The node on the left is the left child
  - Bob is the left child of Jane
- The node on the right is the right child
  - Tom is the right child of Jane

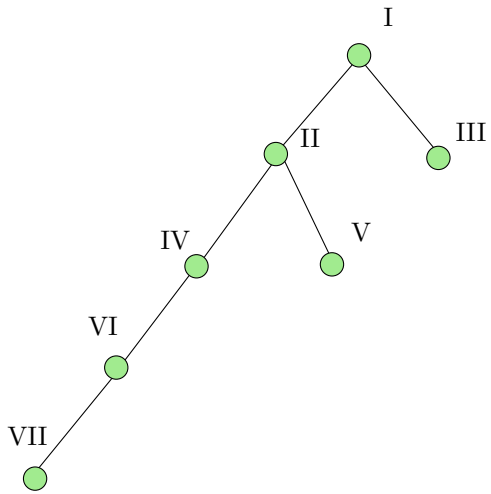
Tree Height and Level

- Level of a node n in a tree T
  - If n is the root of T, it is at level 1
  - If n is not the root of T, look at the level of the parent and add 1
- Height of a tree T defined in terms of the levels of its nodes
  - If T is empty, its height is 0
  - If T is not empty, its height is equal to the maximum level of any node in its two subtrees

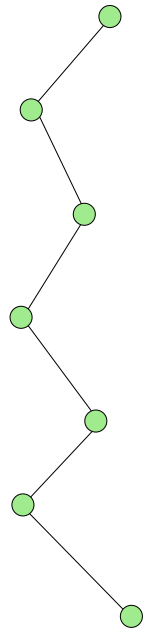
Exercise



(a)



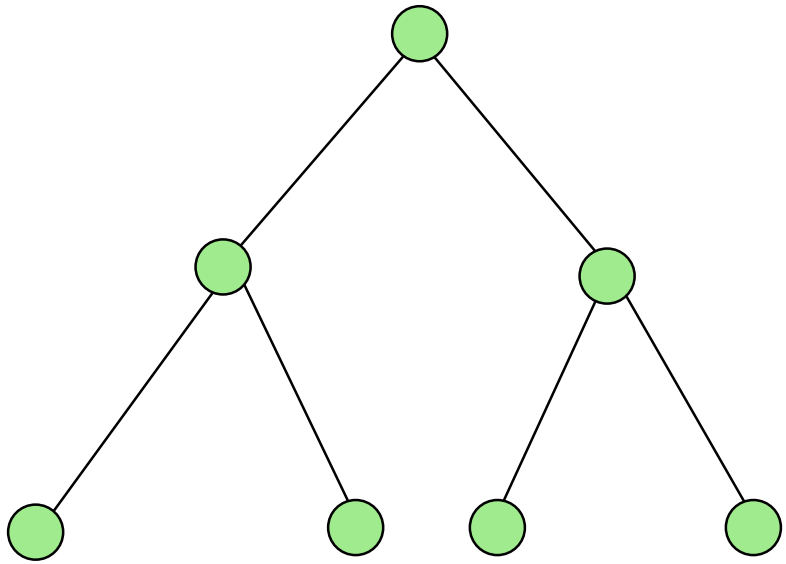
(b)



(c)

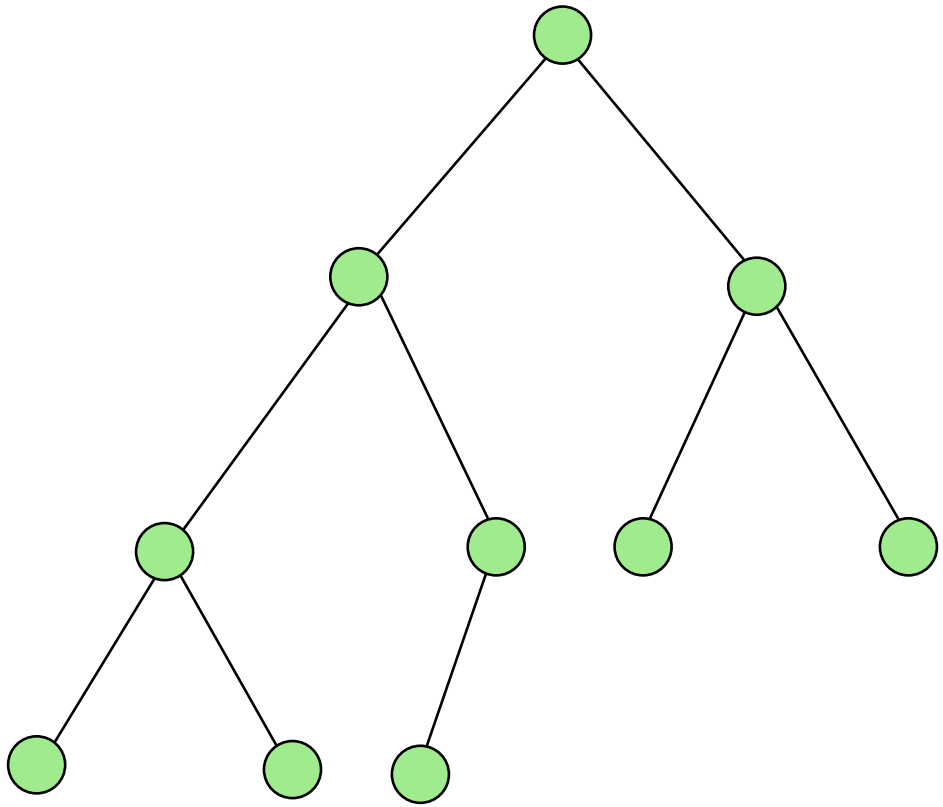
- What is the level of **(a)** I?
- What is the subtree rooted at II in **(b)**?
- What is height of tree **(c)**?

Full Binary Tree



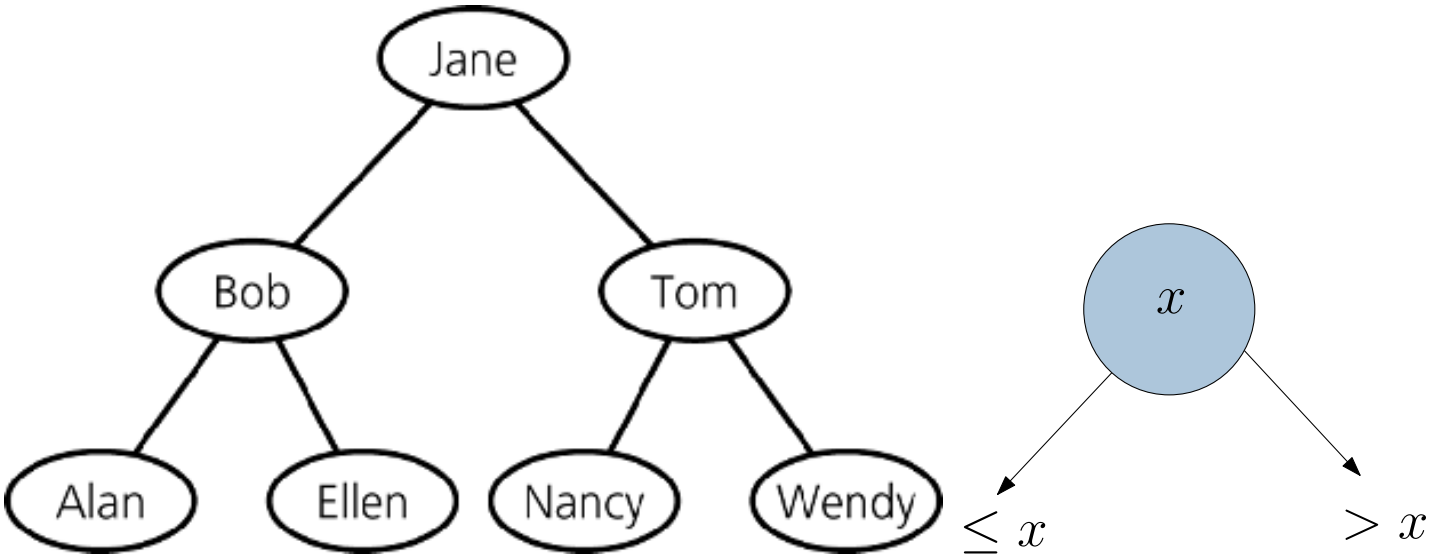
- Definition of a full binary tree
  - If T is a single root node, it is a full binary tree
  - If T is not empty, it is only full if:
    - every internal node has exactly two children
    - all leaves are at the same height

Balanced Binary Tree



- A binary tree is balanced if:
  - the heights of the left and right subtree differ by no more than 1

Binary Search Tree



- For each node in the tree
  - The left child is always less than the node
  - The right child is always greater than the node

Link Representation

- You have a separate class for a tree node
- A tree node contains two references
  - the two references are the left and right children
- Frequently a more natural representation

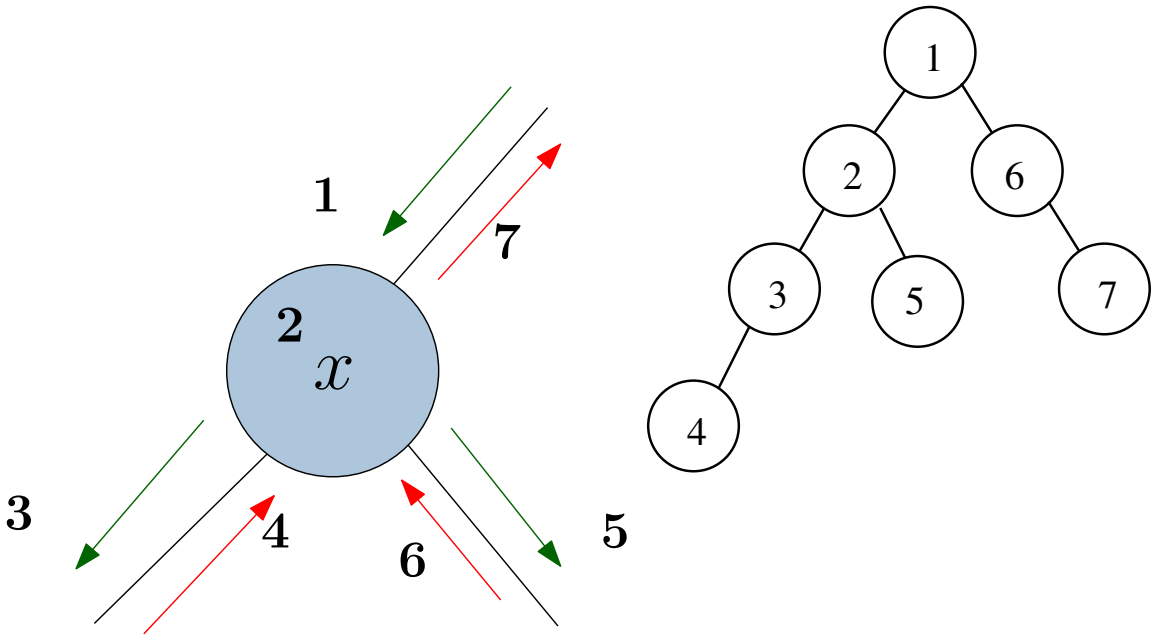
```
public class TreeNode {
   private Object element;
   private TreeNode left;
   private TreeNode right;

   ...
}
```

Traversals of Binary Trees

- Often, we would like to report on or perform a calculation with a binary tree
- To use all the data in a binary tree we must have systematic ways of visiting the data.

Preorder Traversal

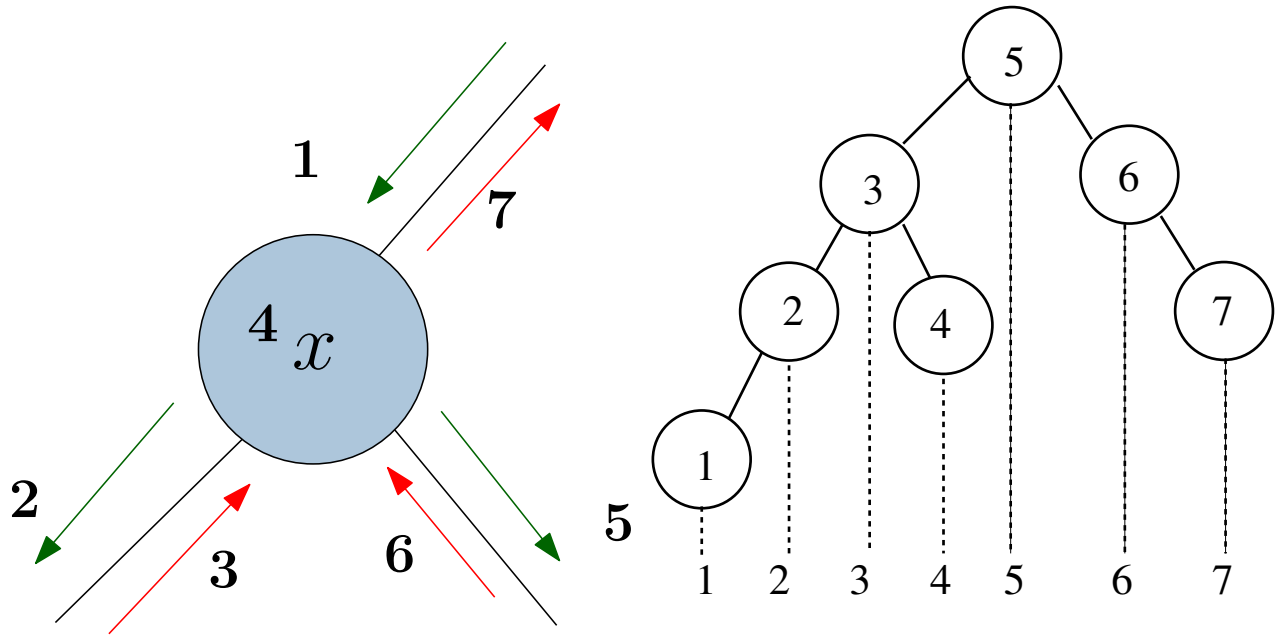
- Visit node and then its children




```
preOrder (TreeNode n)
{
   visit (n);
   if (n.left != null)
      preOrder (n.left);
   if (n.right != null)
      preOrder (n.right);
}
```

Inorder Traversal

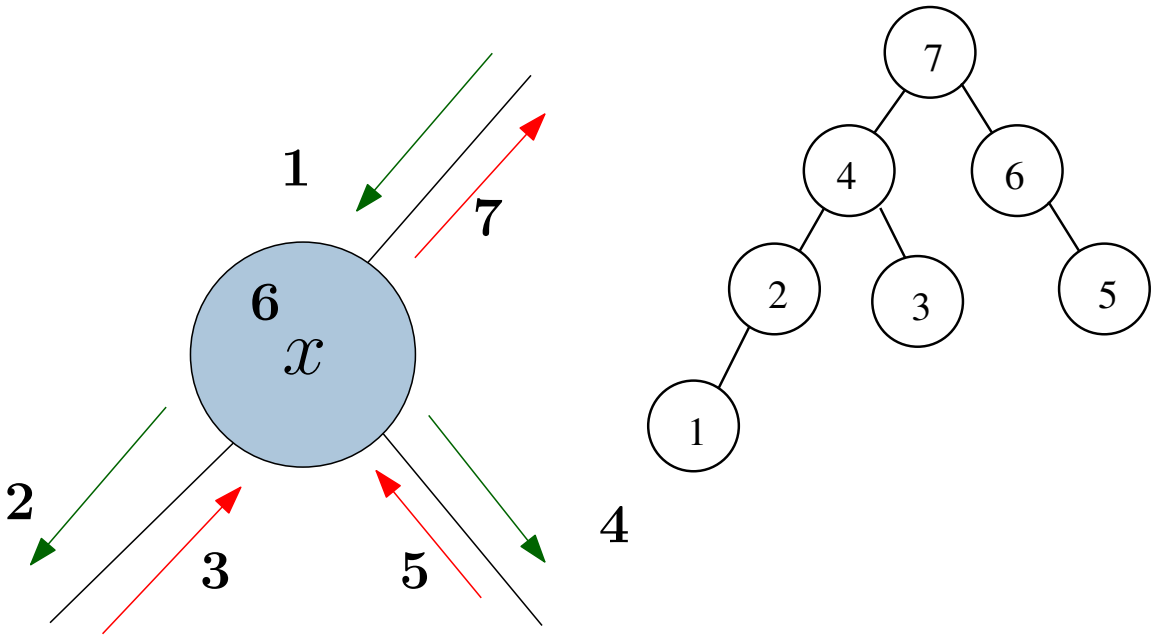
- If a binary search tree, this traversal would be in order




```
inOrder (TreeNode n)
{
   if (n.left != null)
      inOrder (n.left);
   visit (n);
   if (n.right != null)
      inOrder (n.right);
}
```

Postorder Traversal

- Visit the children and then the node




```
postOrder (TreeNode n)
{
   if (n.left != null)
      postOrder (n.left);
   if (n.right != null)
      postOrder (n.right);
   visit (n);
}
```

Summary

- Preorder: visit node and then children
- Inorder: visit left subtree, node, and then right subtree
- Postorder: visit children then node