

Lecture 16: Trees and their Iterators

CS 2110

October 20, 2026

Today's Learning Outcomes

60. Identify parent, child, root, leaf, ancestor, and descendent nodes in a tree.

61. Given a tree, identify its size and height along with the depth and arity of its nodes.

62. Write recursive methods on general and binary trees.

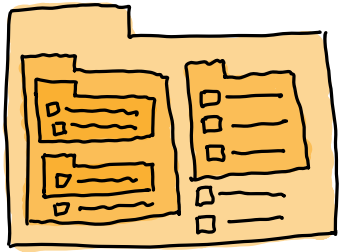
63. Given an image of a binary tree, write out the pre-order, in-order, and post-order traversals of its nodes, and vice versa.

64. Implement iterators on trees.

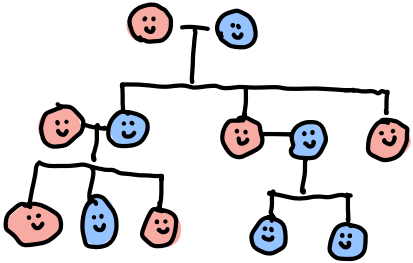
Hierarchical Data

Sometimes, a linear organization of data fails to capture its structure.

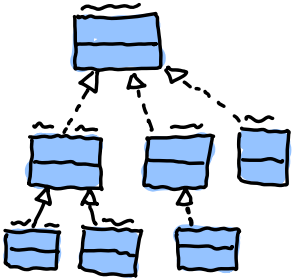
File Systems



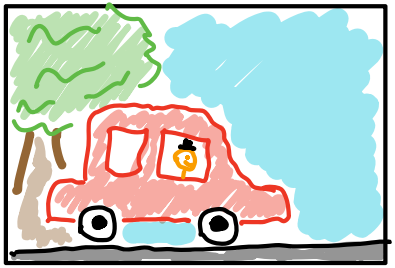
Family Trees



Inheritance Hierarchies

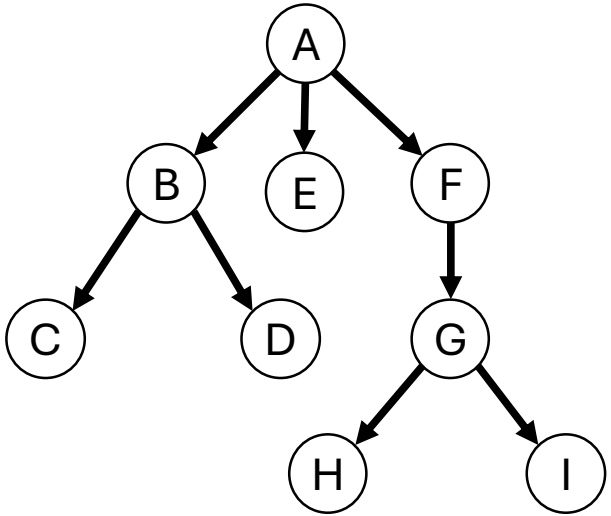


Graphics Rendering



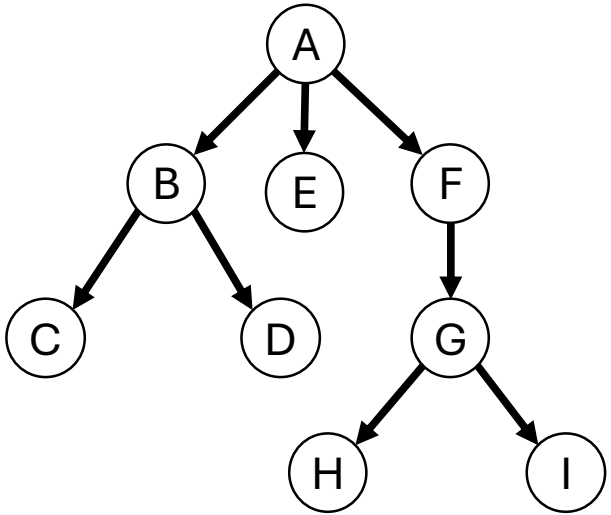
Visualizing a Tree

A **tree** is a linked data structure with nodes arranged in a branching structure.



Subtrees

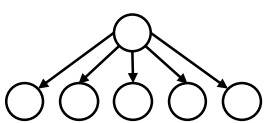
The **subtree** rooted at a node v consists of v and all other nodes reachable by following connections out of v.

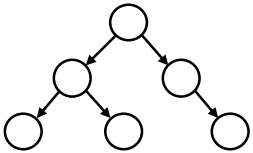


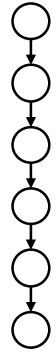
Measuring Trees

The **size** of a tree is the number of nodes that it contains.

Different branching structures allow for many trees with the same size.







Binary Tree Representation

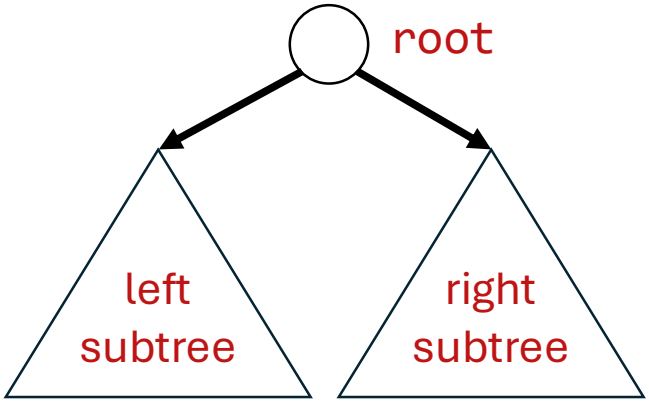
First Idea: Model as a de-centralized linked structure with an auxiliary Node class, this time with two pointers to its child nodes.

```
public class LinkedBinaryTree<T> {
  private static class Node<T> {
    T data;  // The element stored in this tree node
    Node<T> left; // The left child of this node
    Node<T> right; // The right child of this node
    // Node constructor
  }

  private Node<T> root; // The root node of this tree
  // LinkedBinaryTree methods
}
```

The Recursive View of Trees

A `BinaryTree` consists of the element at its root, plus two (possibly `null`) references to its left and right *subtrees*.





Coding Demo: `BinaryTree` class



Key Ideas:

- `protected` visibility
- `abstract` `left()` and `right()` methods (as opposed to fields)

- Nested constructor calls to build `ImmutableBinaryTrees`

Recursive Methods on Trees

Big Idea: Combine the answers from the `left()` and `right()` subtrees (along with the `root`) to get the answer for the whole tree.

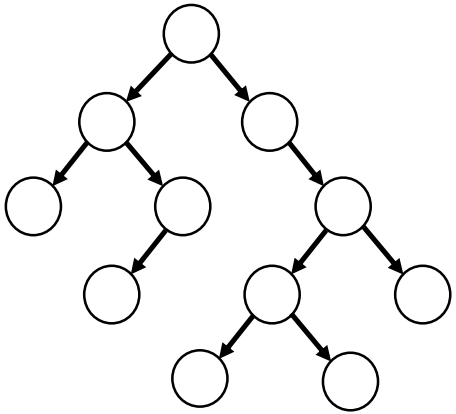
Be careful about `null` children!

BinaryTree.java

```
/** Returns the number of nodes in this tree. */
public int size() {




}
```



Exercise:

Work with your neighbors to complete the definition of this method.

BinaryTree.java

```
/** Returns the number of leaves in this binary tree. */
public int numLeaves() {




}
```

Tree Traversals

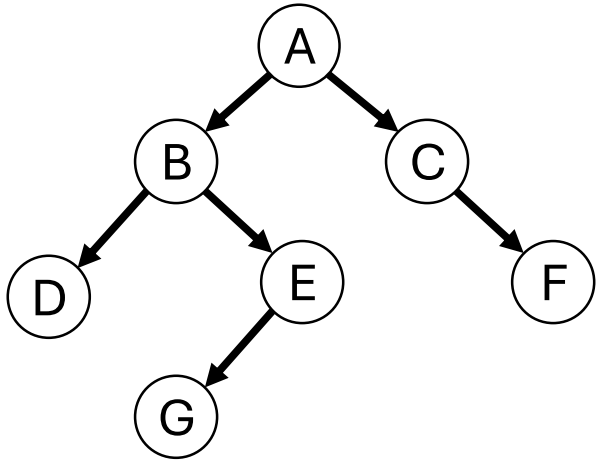
When we **traverse** a tree, we "visit" each of its nodes once in some systematic order.

For branching (non-linear) structures, this order becomes non-obvious…

Pre-Order Traversal

Visiting the root happens ***before*** traversing the subtrees.

1. Visit the root
2. Pre-order traverse the left subtree
3. Pre-order traverse the right subtree



Binary Tree Iterators

We can develop inner `Iterator` classes for `BinaryTree` that yield its elements in each of the three traversal orders.

We'll need to introduce field to the iterator to trach the "traversal state"

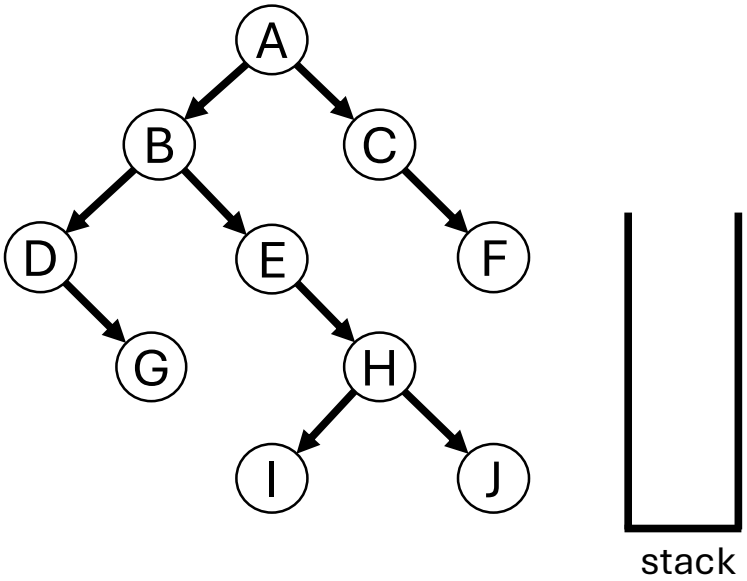


Coding Demo: Pre-Order Iterator



Designing an In-Order Iterator

New Challenge: Need to "drill down" to locate the first node to return (etc.)





Coding Demo: In-Order Iterator

