Poll Everywhere

PollEv.com/javabear text **`javabear`** to **`22333`**

Suppose we push`()` 1, 2, 3, 4 onto a `Stack` (in that order), calling `pop()` at various points during and after this.

In which of the following orders could the elements *not* have been `pop()`ped?

1, 2, 3, 4 **(A)**	1, 3, 2, 4 **(B)**	2, 1, 4, 3 **(C)**
4, 3, 2, 1 **(D)**	3, 1, 4, 2 [✓ **(E)**]	3, 2, 4, 1 **(F)**

Lecture 16: Trees and their Iterators

CS 2110

March 17, 2026

Today's Learning Outcomes

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

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

67. Write recursive methods on general and binary trees.

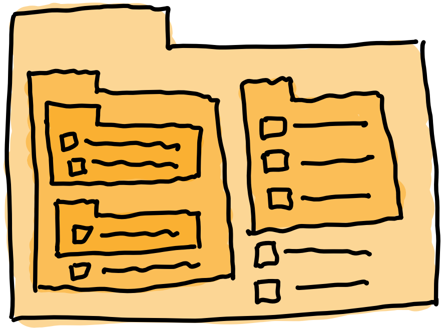
68. Given an image of a binary tree, write out the pre-order, in-order, and post-order traversals of its nodes, and vice versa.

69. Implement iterators on trees.

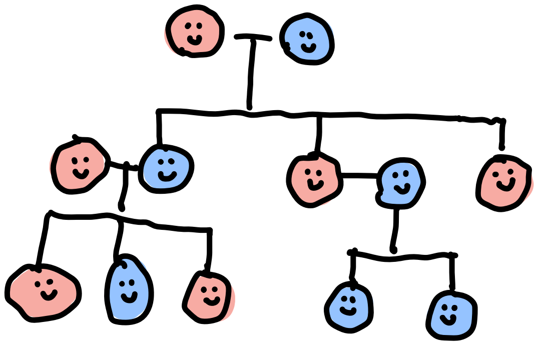
Hierarchical Data

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

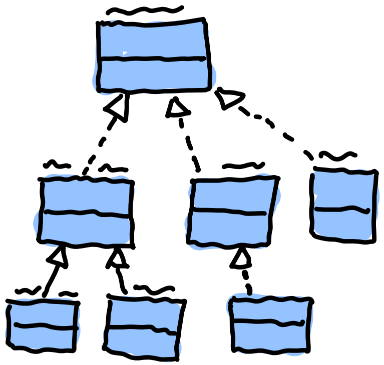
File Systems



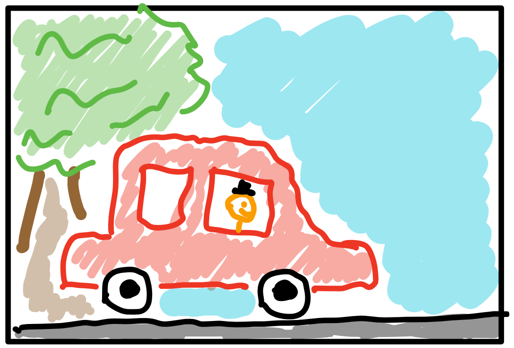
Family Trees



Inheritance Hierarchies

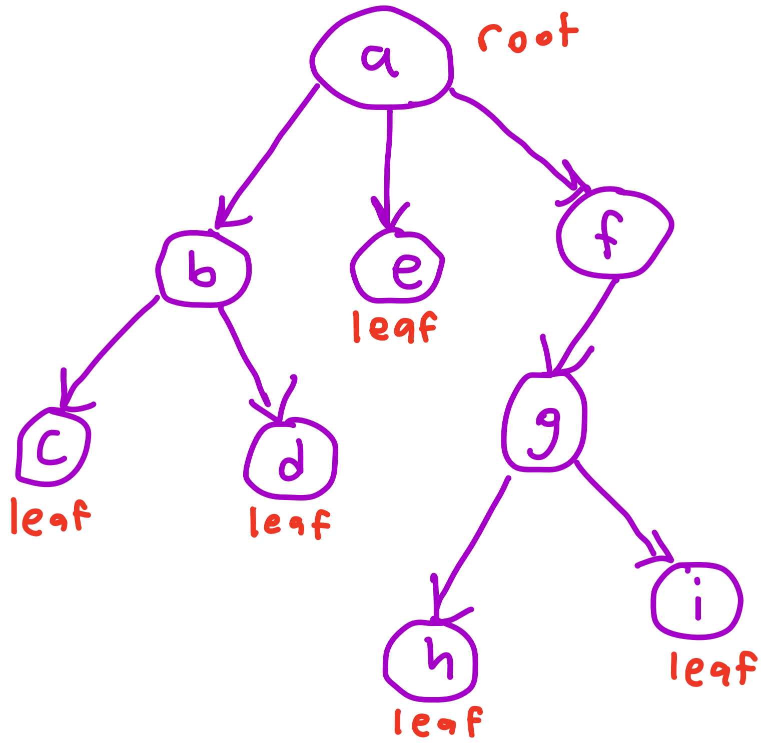


Graphics Rendering



Visualizing a Tree

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

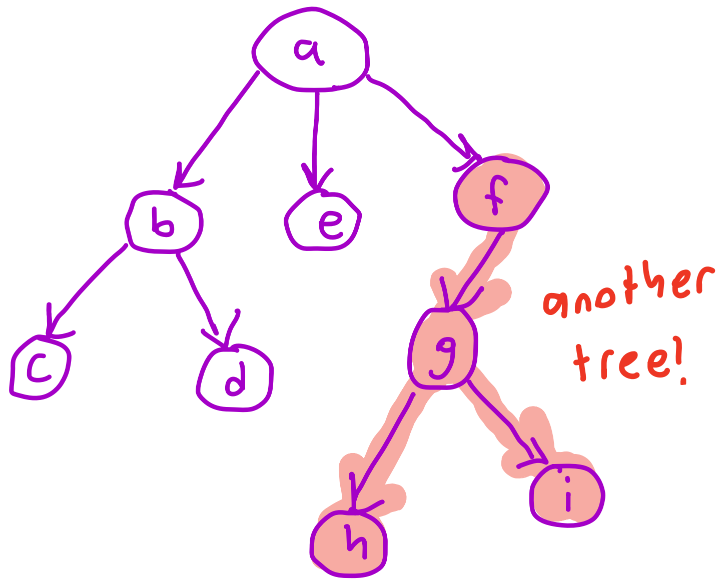


root node = source of branching (no incoming connections)

- all connections oriented away from root
- every other node has one incoming connection from its parent
- nodes each have outgoing connections to zero or more children
- leaf nodes have no children

Subtrees

The <u>subtree</u> rooted at node v consists of v and all other nodes reachable by following connections out of v.



Node u is an <u>ancestor</u> of node v (equiv. v is a <u>descendant</u> of u) if v belongs to the subtree rooted at u.

f is an ancestor of g, h, i (and f)

h is a descendant of g, f, a (and h)

hmm... seems like recursion is coming

Poll Everywhere

PollEv.com/javabear text **javabear** to **22333**

Which of the following statements is *not necessarily true* about nodes u, v, and w in a tree?

If u is an ancestor of v and v is an ancestor of w, then u is an ancestor of w. True transitivity **(A)**

If u is an ancestor of w and v is an ancestor of w, then either u is an ancestor of v or v is an ancestor of u. True w's lineage is a linked chain **(B)**

If u is an ancestor of v and u is an ancestor of w, then either v is an ancestor of w or w is an ancestor of v. False

$$\begin{tikzcd} & u \arrow[dl] \arrow[dr] & \\ v & & w \end{tikzcd}$$

[✓ (C)]

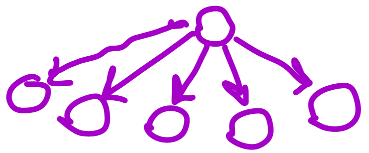
Measuring Trees

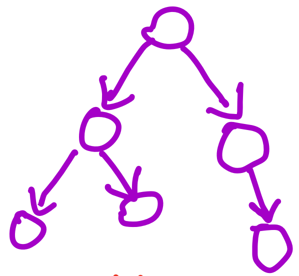
Size = # of nodes in a tree

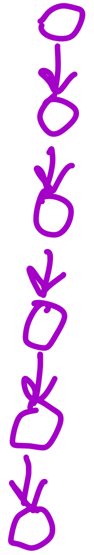
Different branching structures allow many trees with same size

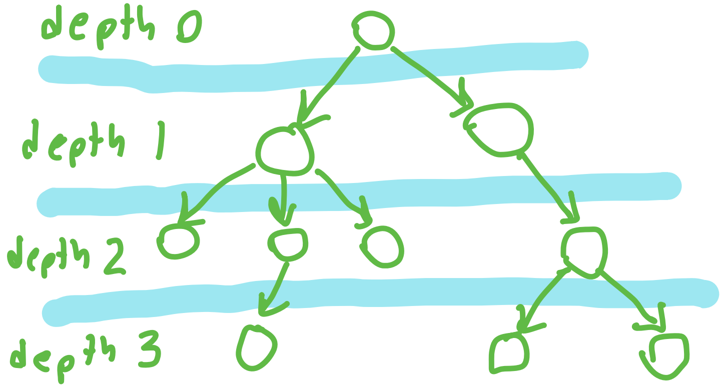
Height = max # connections to get from root ⟶ leaf

Depth of node = # connections needed to get to it from root









Arity = max # of outgoing connections from a node

Binary trees have arity ≤ 2.

General trees have any arity.

Binary Tree Representation

First Idea: Model as a de-centralized linked structure with an auxiliary Node class (this time with two pointers)

```java
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
}
```

every Node reachable from here

Issue: To visit all nodes, we need to worry about branching and backtracking

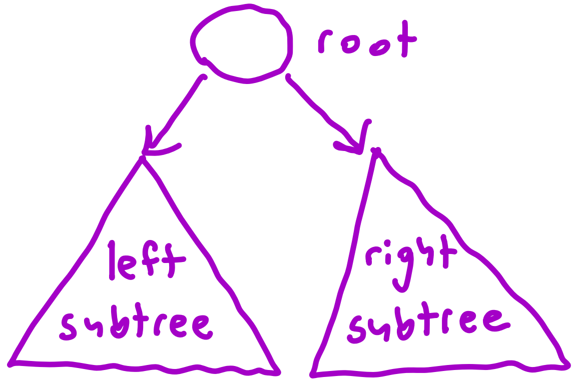
... clunky to track state ...

Big idea: Use runtime stack and recursion

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.

↑ not children!



Tree class itself (not inner node class) becomes recursive.

Client calls methods on the root subtree (i.e., whole tree) which recursively delegates computation to its subtrees.

Coding Demo: `BinaryTree` class

Key Ideas:

- protected visibility
- abstract left() and right() methods (vs. fields)

Recursive Methods on Trees

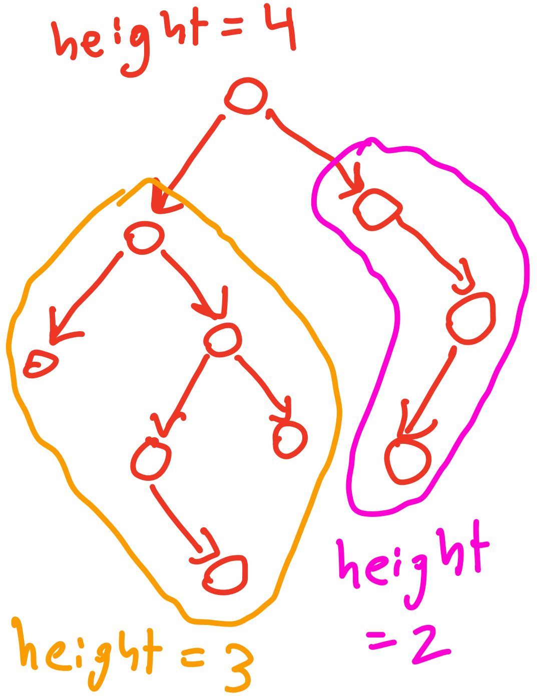
General Idea: Combine the answers from the left() and right() subtrees (with root value) to get answer for whole tree. Be careful about null children!

```
/** Returns the height of this binary tree. */
public int height() {
  return 1 + Math.max(
    left()==null ? -1 : left().height(),
    right()==null ? -1 : right().height()
  );
}
```

Time: O(size)

Space: O(height)

↑ tree with just root should have height 0



Your Turn!

```
/** Returns the number of leaves in this binary tree. */        this is a leaf
public int numLeaves() {
  if (left() == null && right() == null) { return 1; }
  return (left() == null ? 0 : left().numLeaves())
       + (right() == null ? 0 : right().numLeaves());
}
```

Tree Traversals

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

For a branching (non-linear) structure, order becomes non-obvious...

From the recursive view, we can split traversal into 3 "steps":

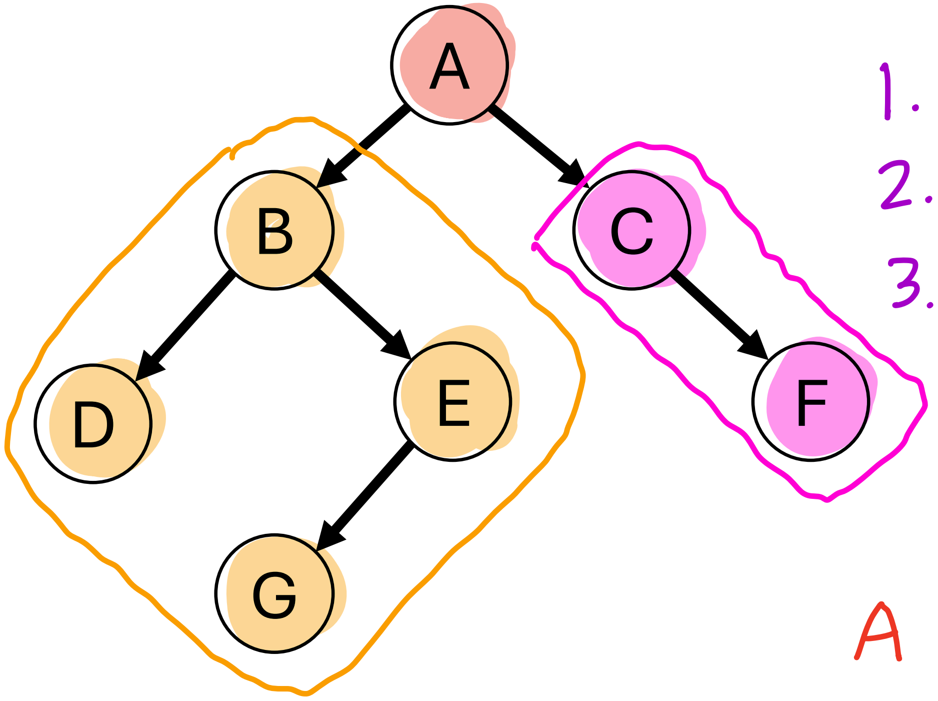
- visit the tree root
- recursively traverse the left subtree
- recursively traverse the right subtree

Different ways to "slot" root visit in lead to three different traversal strategies.

* Different traversal orders useful for different applications (e.g., sorting, parsing)

Pre-Order Traversal

↖ visiting root happens before traversing subtrees



1. visit root
2. pre-order traverse left
3. pre-order traverse right

A $\boxed{\text{B D E G}}$ $\boxed{\text{C F}}$

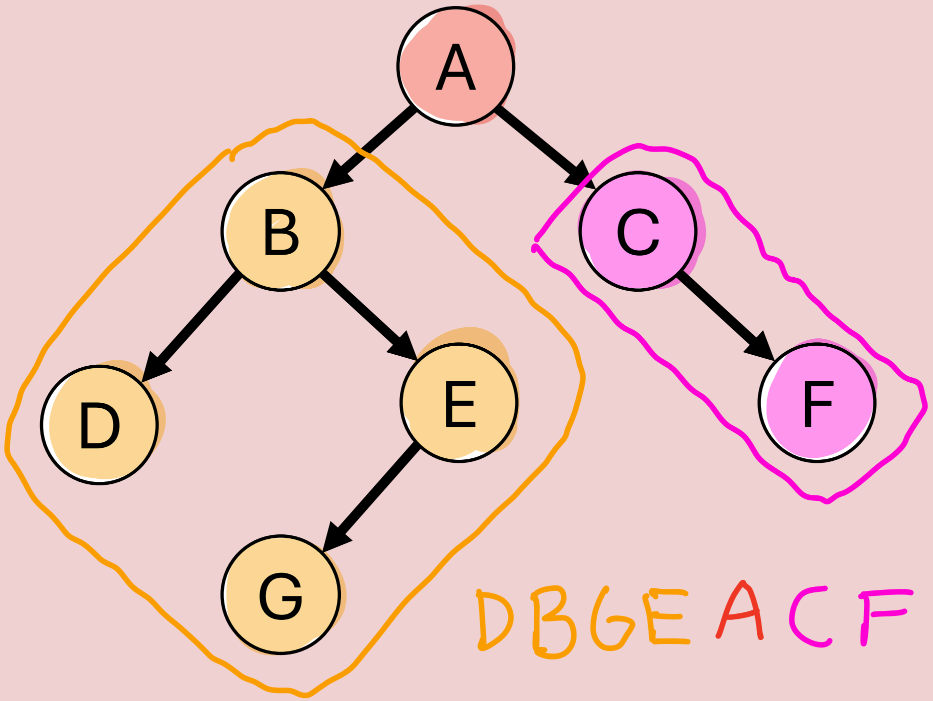
left right

Poll Everywhere

PollEv.com/javabear text **`javabear`** to **22333**

In an *In-Order Traversal* we visit the root after traversing the left subtree and before traversing the right subtree.

Which of the following is the in-order traversal of this tree?



A B D E G C F ✗ **(A)**

D B E G A C F **(B)**

D G B E A C F ✗ **(C)**

D B G E A C F [✓ **(D)**]

Poll Everywhere

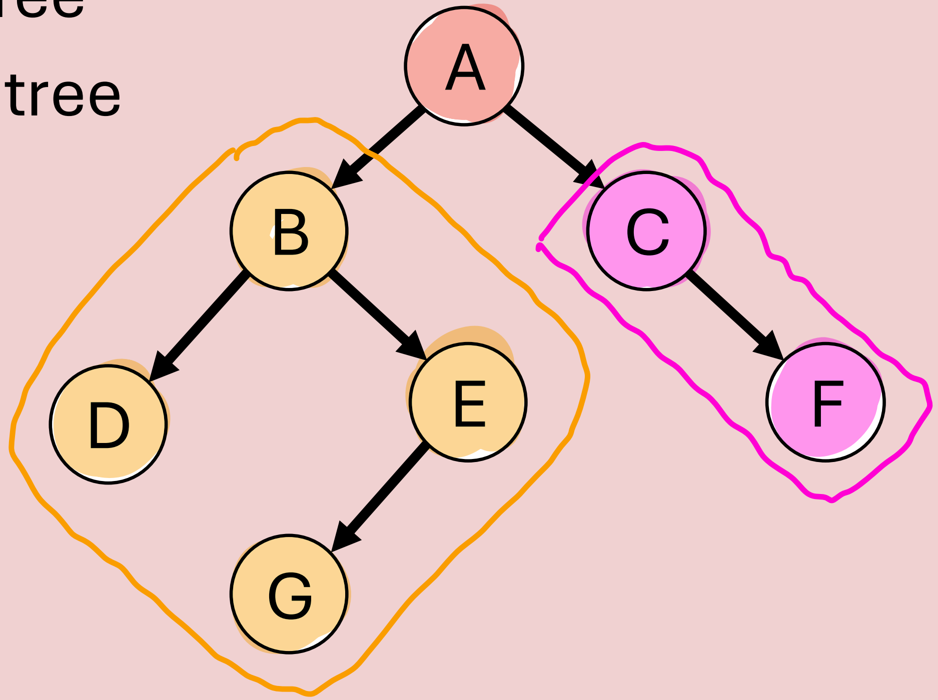
PollEv.com/javabear text **javabear** to **22333**

In a *Post-Order Traversal*, we:

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

Write out the post-order traversal of this tree.

$\boxed{DGEB}\ \boxed{FC}\ A$



Binary Tree Iterators

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

Need to keep track of "state of traversal"

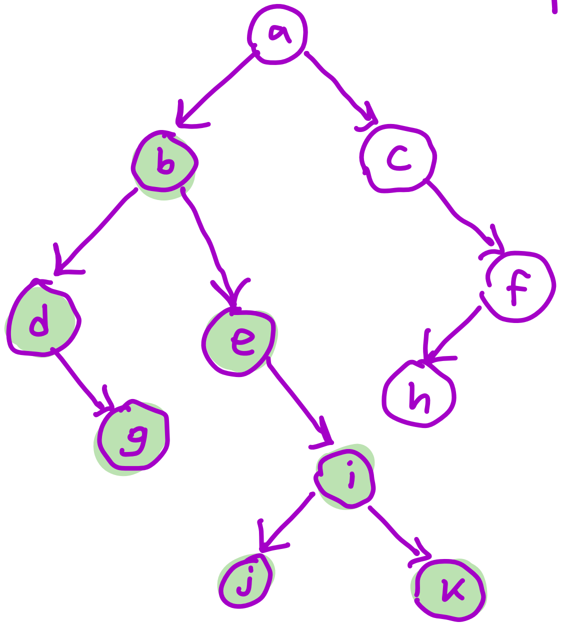
- status at each level of the tree (have we visited root, left, right yet)
- model using a <u>stack</u> (if we're traversing lower levels, need to finish this before returning to root)

<u>Common invariant</u>: Top of stack will always be node with next() element to return. Empty stack = done!

Coding Demo: Pre-Order Iterator

Designing an In-Order Iterator

New challenge: Need to "drill down" to locate first node to return (and similar later in traversal)



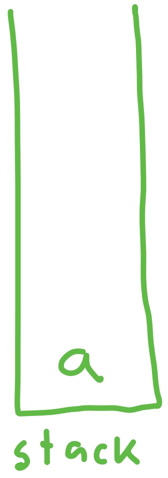
First node to return = d ("leftmost" node)

Idea: keep following left() pointers until we reach a node without one.

cascadeLeft() ^ helper method

iterative (since it lives in main tree's iterator)

dgbejik...



strategy for next():

pop() then cascadeLeft() on right subtree

Coding Demo: In-Order Iterator