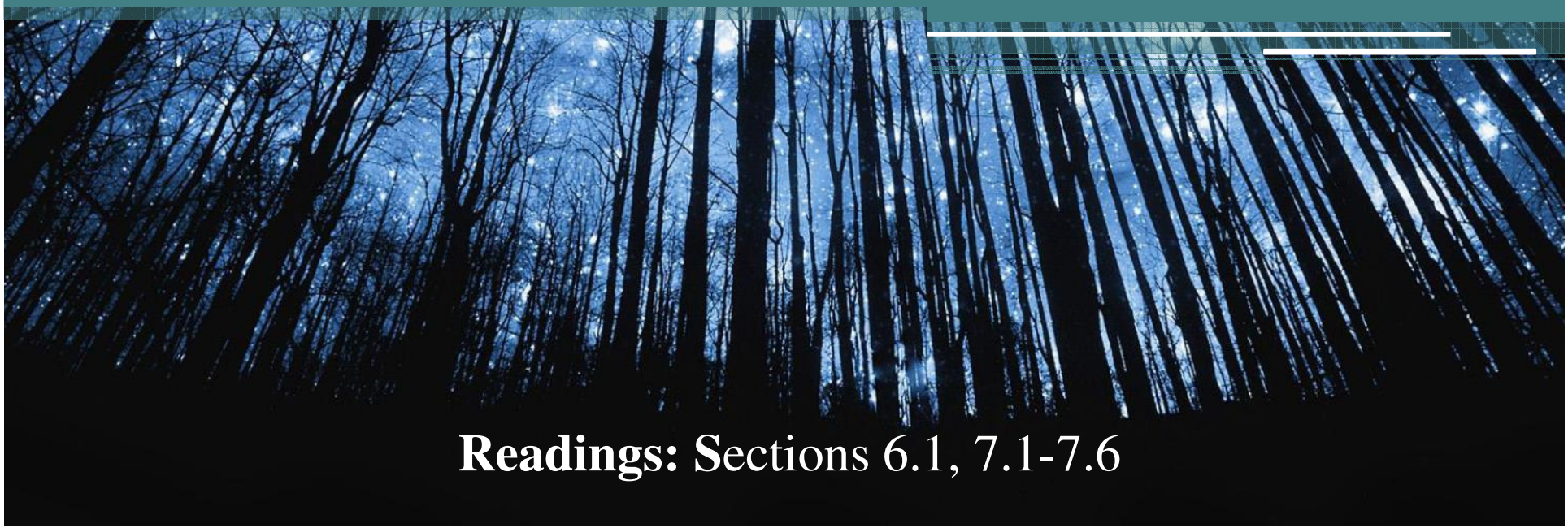
Recursive Data Structures: Trees



Readings: Sections 6.1, 7.1-7.6

Goals for this Unit

- Continue focus on data structures and algorithms
- Understand concepts of reference-based data structures (e.g. linked lists, binary trees)
  - Some implementation for binary trees
- Understand usefulness of trees and hierarchies as useful data models
  - Recursion used to define data organization
- Topics:
  - Trees
  - Tree Traversals
  - Heaps (“binary heaps”)
  - BST
  - Grammars / Tree Recursion

Taxonomy of Data Structures

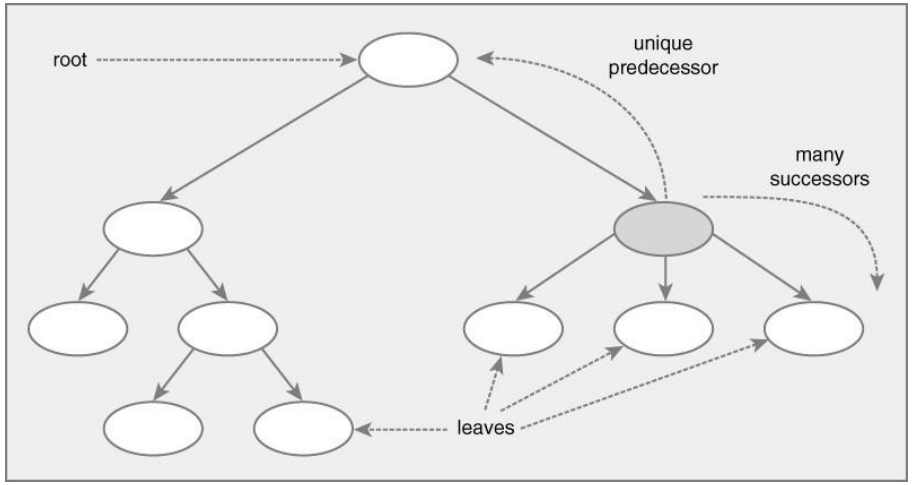
- From the text:
  - Data type: collection of values and operations
    - Compare to Abstract Data Type!
  - Simple data types vs. composite data types
- Book: Data structures are composite data types
  - Definition: a collection of elements that are some combination of primitive and other composite data types

Book's Classification of Data Structures

- Four groupings:
  - Linear Data Structures
  - Hierarchical
  - Graph
  - Sets and Tables
- When defining these, note an element has:
  - one or more *information fields*
  - relationships with other elements

Trees Represent…

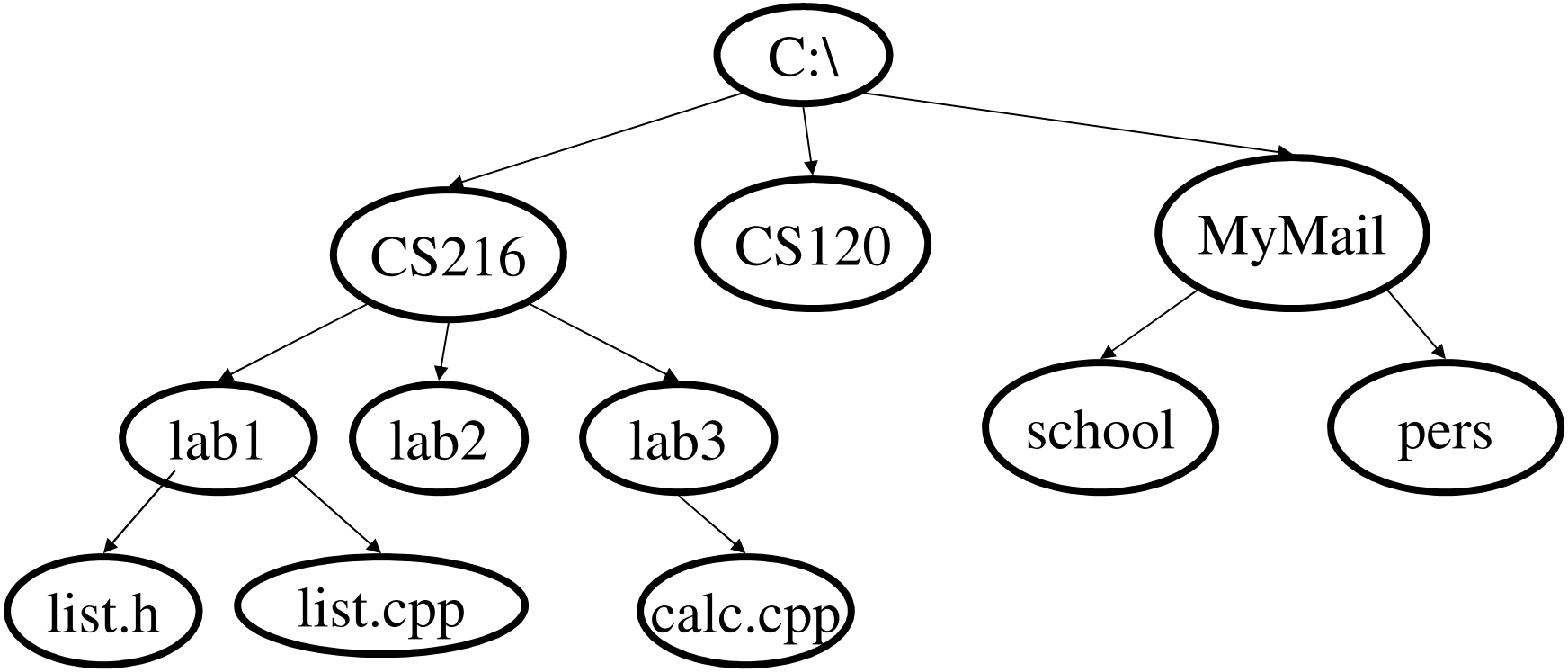
- Concept of a tree is very common and important
- Tree terminology:
  - _____ have one parent
  - A node's children
    - Can have many successors
    - _____ nodes: no children
  - _____ node: top or start; no parent
- Data structures that store trees
- Execution or processing that can be expressed as a tree
  - E.g. method calls as a program runs
  - Searching a maze or puzzle



Trees are Important

- Trees are important for cognition and computation
  - computer science
  - language processing (human or computer)
    - parse trees
  - knowledge representation (or modeling of the "*real world*")
    - E.g. family trees; the Linnaean taxonomy (kingdom, phylum, …, species); etc.

Another Tree Example: File System



- What about file links (Unix) or shortcuts (Windows)?

Another Tree Example: XML and HTML documents

```
<HTML>
<HEAD>…</HEAD>
<BODY>
 <H1>My Page</H1>
   <P> Hello… Blah Blah…
   <B>blah blah</B> The End
   </P></BODY>
</HTML>
```

How is this a tree?
What are the leaves?

Small XML example - Tree structure

```
<note>
   <to>Tim</to>
   <from>Kat</from>
   <heading>Reminder</heading>
   <body>Don't forget our road trip this weekend!</body>
</note>
```

- The first line `<note>` describes the **root** element (parent)
  e.g. "this document is a note"
- The next four lines describe 4 **child** elements of the root (`to`, `from`, `heading`, and `body`)
- Finally the last line `</note>` defines the end of the root

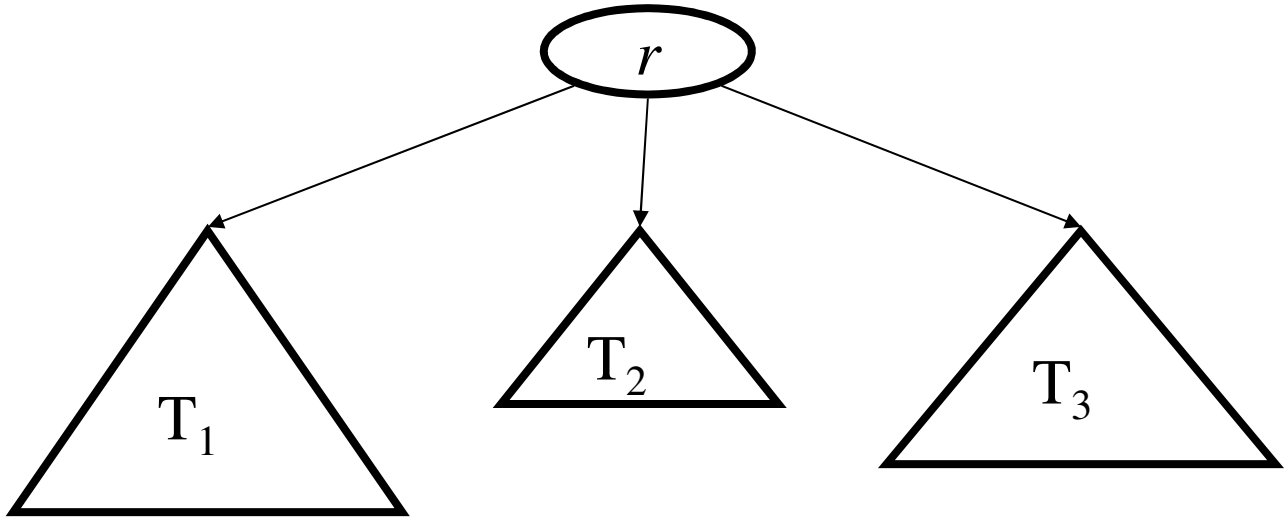
Tree Data Structures

- Why this now?
  - Very useful in coding
  - Example of recursive data structures
  - Methods are recursive algorithms

Tree Definitions and Terms

- First, general trees vs. binary trees
  - Each node in a **binary tree** has at most _____ children
- **General** tree definition:
  - Set of nodes T (possibly empty?) with a *distinguished node*, the *root*
  - All other nodes form a set of disjoint subtrees T_i
    - each a tree in its own right
    - each connected to the root with an edge
- Note the recursive definition
  - Each node is the root of a _____

Picture of Tree Definition



- And all subtrees are **recursively** defined as:
  - a node with…
  - subtrees attached to it (e.g. T_1, T_2, and T_3)

Tree Terminology

- A node's **parent**
- A node's **children**
  - Binary tree: "left child" and "right child"
  - **Sibling** nodes
  - **Descendants, ancestors**
- A node's _____(*how many children*)
- **Leaf** nodes or _____nodes
- **Internal** or _____nodes

Recursive Data Structure

- A data structure that contains a reference (or pointer) to an instance of that same type

```
public class TreeNode<E> {
    private E data;
    private TreeNode<E> left;
    private TreeNode<E> right;
      …
}
```

- Recursion is a natural way to express many data structures
- For these, natural to have recursive algorithms

ADT Tree

- Remember definition on an ADT?
  - Model of information: we just covered that
  - Operations? See pages 405-406 in textbook
- Many are similar to ADT List or any data structure
  - The "CRUD" operations: create, replace, update, delete
- Important about this list of operations
  - some are in terms of one specified node, e.g. hasParent()
  - others are "tree-wide", e.g. size(), traversal

Classes for Binary Trees

We'll use simplified versions of the book's!

- class BinTree
  - reference to root node
  - methods: tree-level operations
- class TreeNode
  - **data**: an object (of some Comparable type)
  - **left**: references root of left-subtree (or null)
  - **right**: references root of right-subtree (or null)
  - (could have) **parent**: this node's parent node
    - Could this be null? When should it be?
  - **methods**: node-level operations

Two-class Strategy for Recursive Data Structures

- Common design: use two classes for a Tree or List
- "Top" class
  - has reference to "first" node
  - other things that apply to the whole data-structure object (e.g. the tree-object)
    - both methods and fields
- Node class
  - Recursive definitions are here as references to other node objects
  - Also data (of course)
  - Methods defined in this class are recursive

Why Does This Matter Now?

- This illustrates (again) important design ideas
- The tree itself is what we're interested in
  - There are tree-level operations on it ("ADT level" operations)
- The implementation is a recursive data structure
  - There are recursive methods inside the lower-level classes that are *closely related* (same name!) to the ADT-level operation
- Principles? abstraction (hiding details), delegation (helper classes, methods)

Some Methods

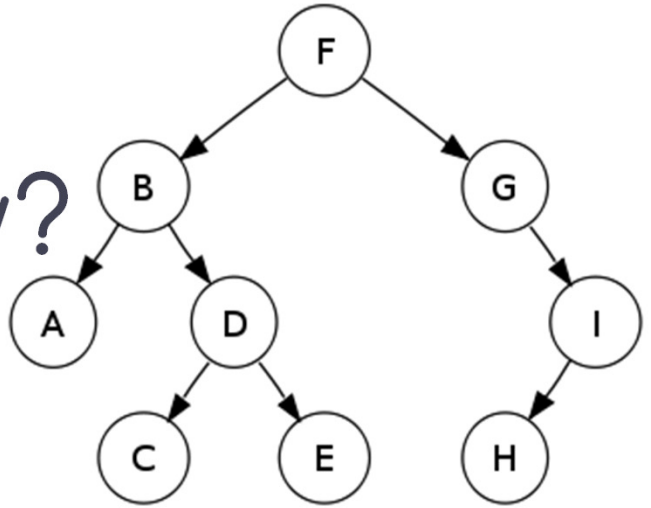
Discussion: How might we write the following methods?

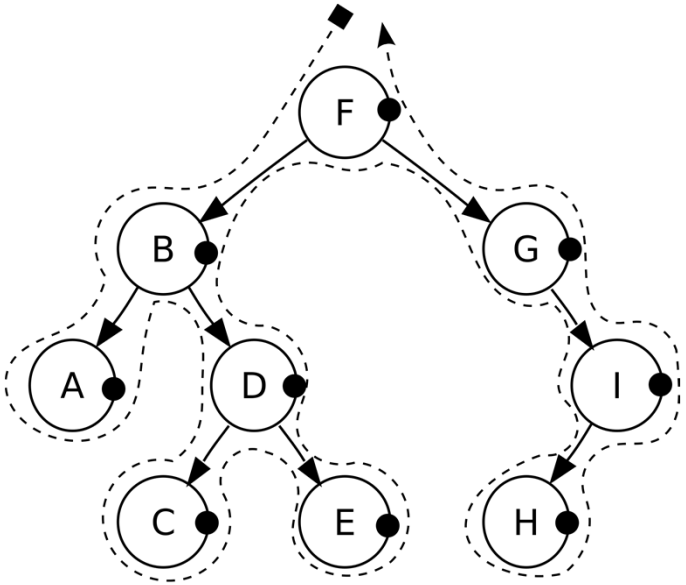
- size()
- height()
- find() [*which assumes no order of nodes in the tree*]
- …

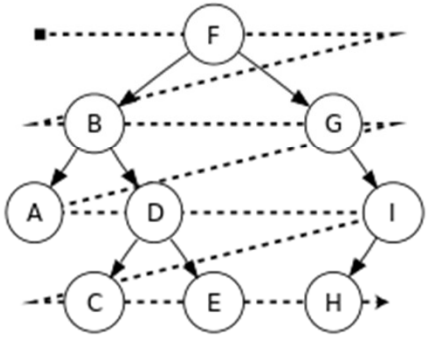
Let's Go To Eclipse

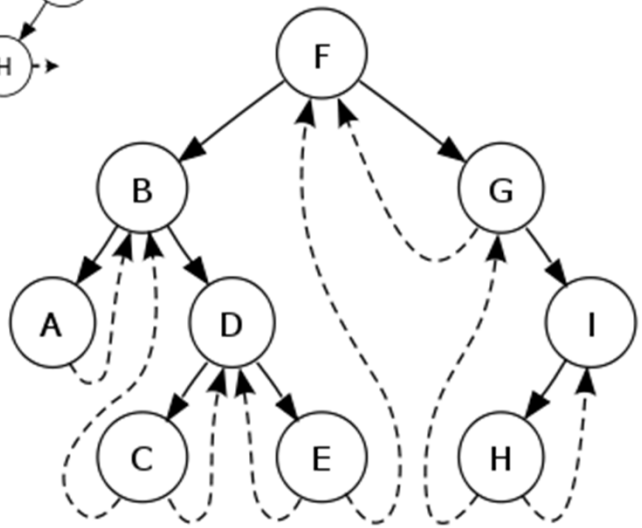
Code on Trees

Tree Traversals – How?









Tree Traversals

- It is not always clear how we should print a tree
  - Top to bottom?
  - Left to right?
- A _____ is a specific order in which to trace the nodes of a tree (visit *every* node ***once***)
- There are three common tree traversals for binary trees:
  1. pre-order
  2. in-order
  3. post-order
- This order is applied ***recursively***

Tree Traversals

- In each technique, the left subtree is traversed recursively, the right subtree is traversed recursively, and the root is visited
- What distinguishes the techniques from one another is ***the _____ of those 3 tasks***

Tree Traversals

- Visiting a node entails doing some processing at that node, but when describing a traversal strategy, we need not concern ourselves with what that processing is
  - Often times it is just printing out the node (label, or its data) but it can be more complicated than that
- For **in-order**, we must print each subtree's left branch before we print its root
- Note "**pre**" and "**post**" refer to when we visit the root (of that subtree)

Preoder, Inorder, Postorder

- In Preorder, the root is visited _____(pre) the subtrees traversals
- In Inorder, the root is visited _____left and right subtree traversal
- In Postorder, the root is visited _____(pre) the subtrees traversals

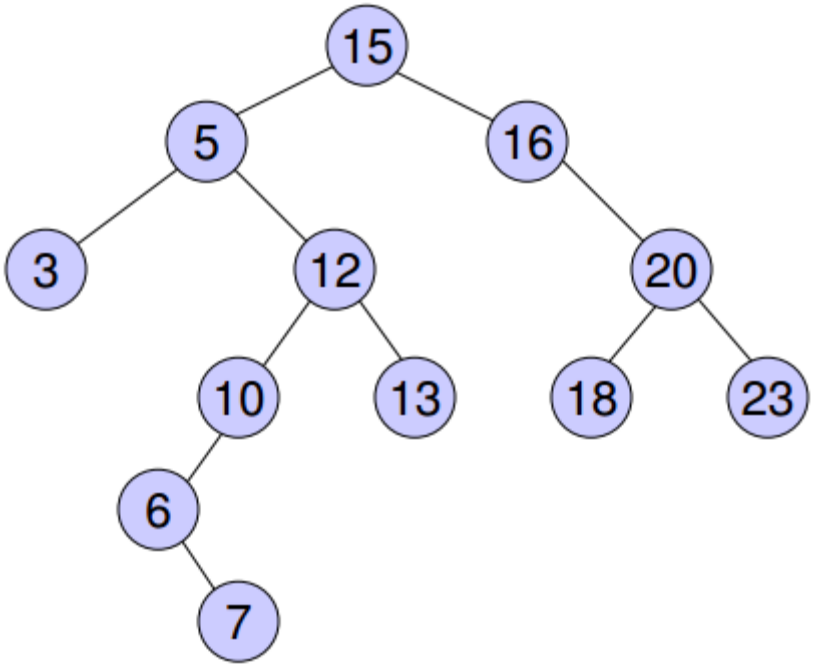
Preorder Traversal:
1. Visit the **root**
2. Traverse **left** subtree
3. Traverse **right** subtree

Inorder Traversal:
1. Traverse **left** subtree
2. Visit the **root**
3. Traverse **right** subtree

Postorder Traversal:
1. Traverse **left** subtree
2. Traverse **right** subtree
3. Visit the **root**

Tree Traversal Example

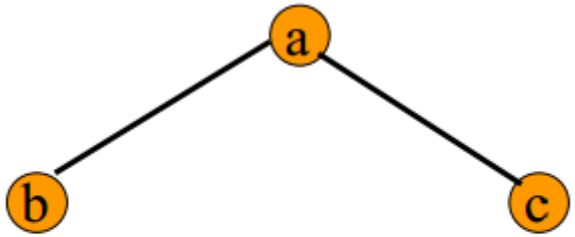
Let's do an example first…



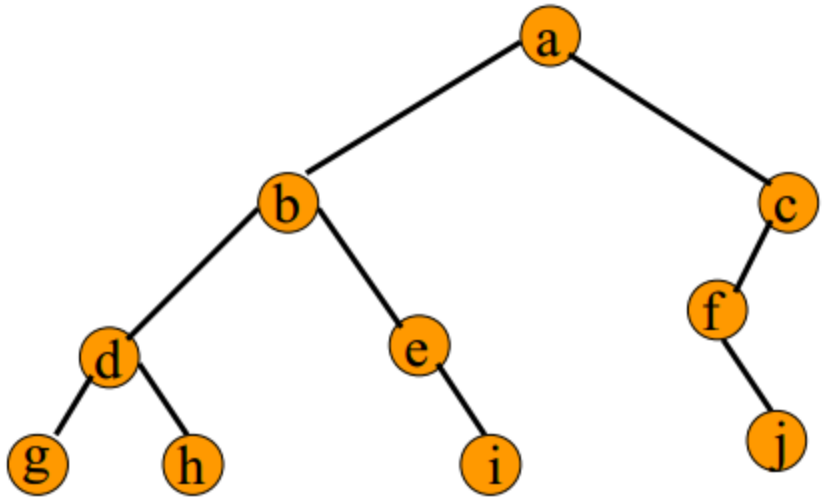
- **pre-order:** (root, left, right)
  15, 5, 3, 12, 10, 6, 7,
  13, 16, 20, 18, 23
- **in-order:** (left, root, right)
  3, 5, 6, 7, 10, 12, 13,
  15, 16, 18, 20, 23
- **post-order:** (left, right, root)
  3, 7, 6, 10, 13, 12, 5,
  18, 23, 20, 16, 15

Pre-order Traversal

- Pre-order traversal prints in order: root, left, right



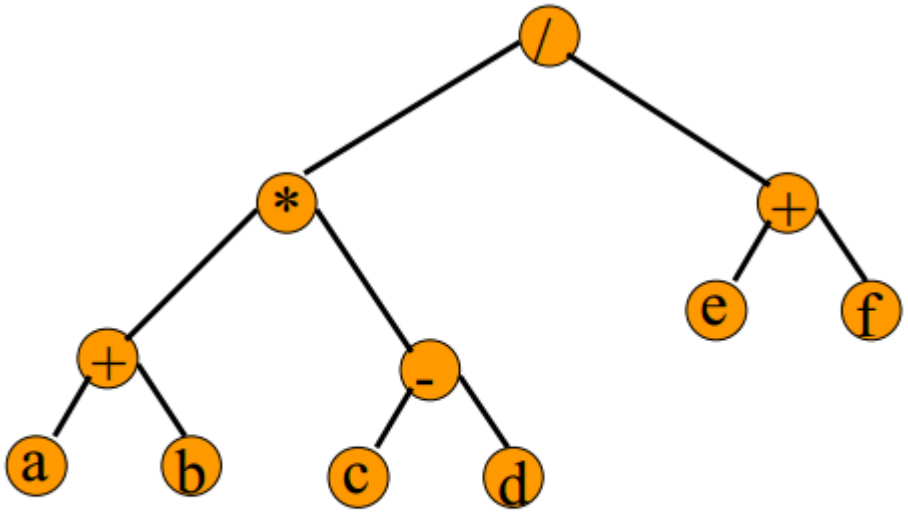
a b c




```
a  b  d
```

Pre-order Traversal

- Gives prefix form of expression!

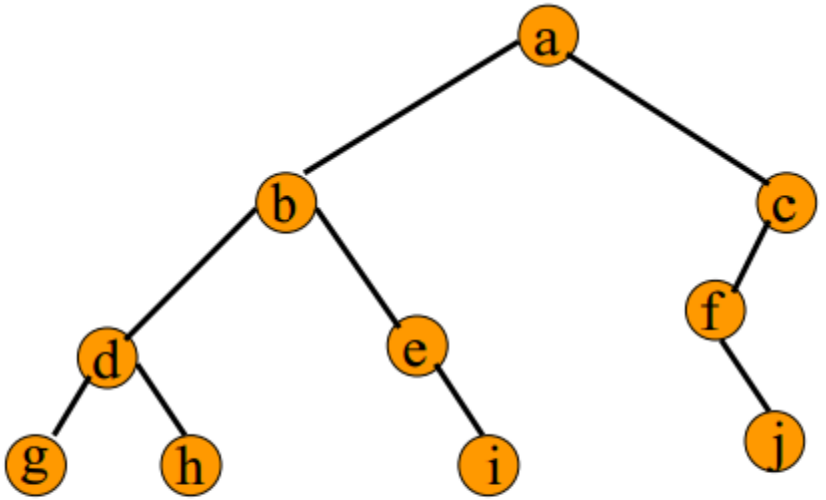


/ * + a b - c d + e f

(will revisit this later)

In-order Traversal

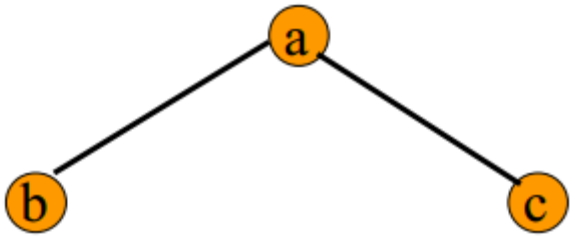
- The in-order traversal is probably the easiest to see, because it



g d h b e i a f j c

(*See also slide #26*)

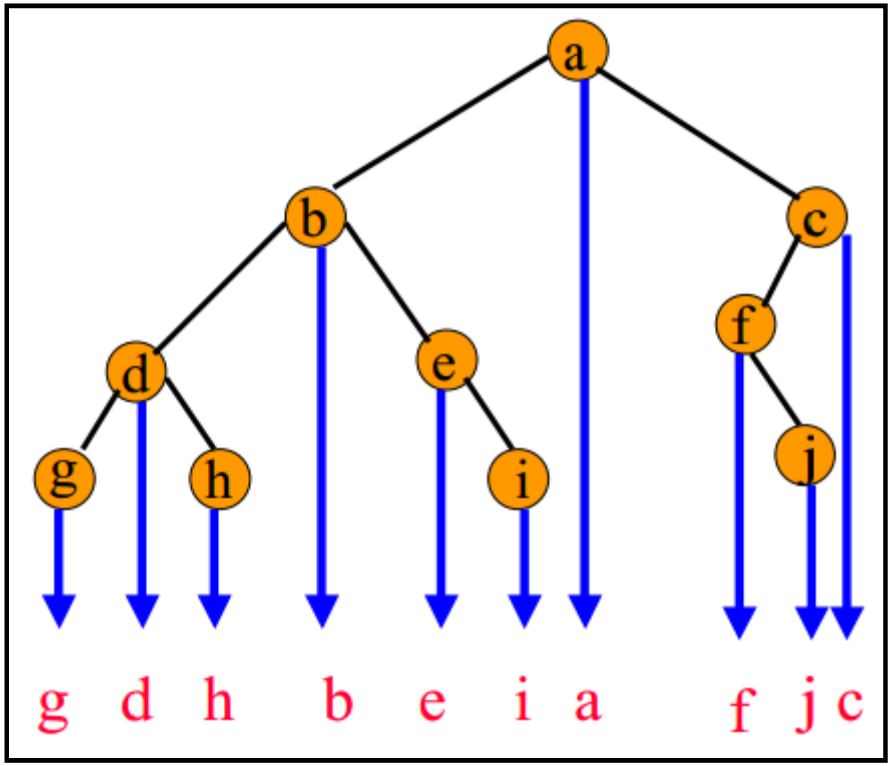
- t, right

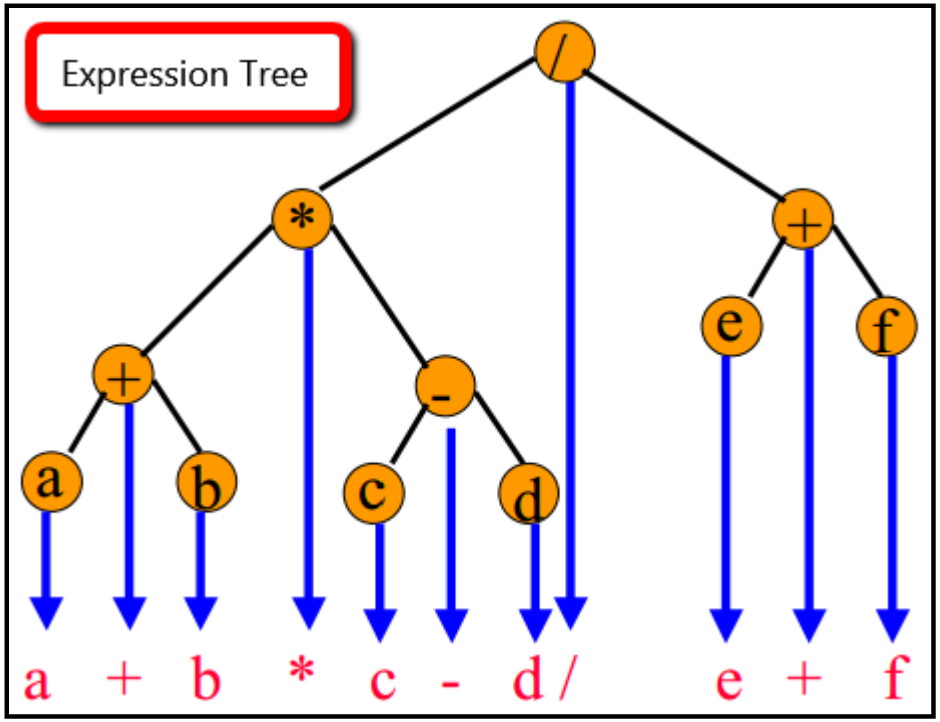


b a c

In-order Traversal (Projection)

- Gives infix form of expression (sans parenthesis)!

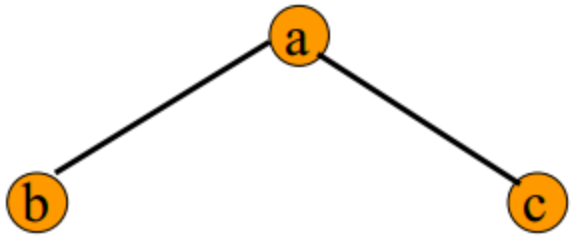




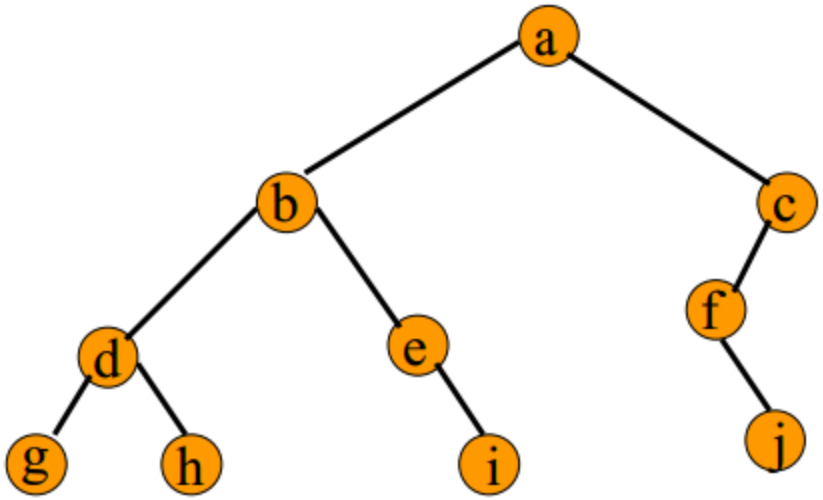
(will revisit this later)

Post-order Traversal

- Post-order traversal prints in order: left, right, root
- It is also called a



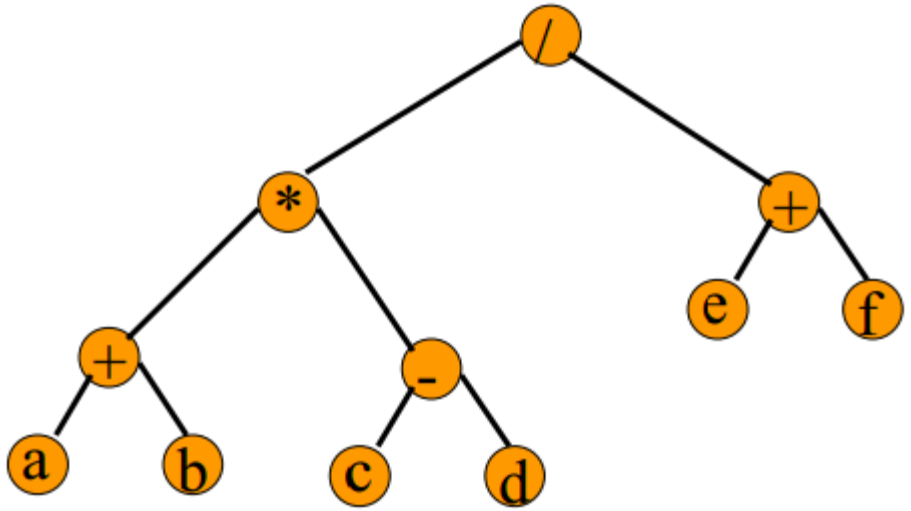
b c a



g h d i e b j f c a

Post-order Traversal

- Gives postfix form of expression!

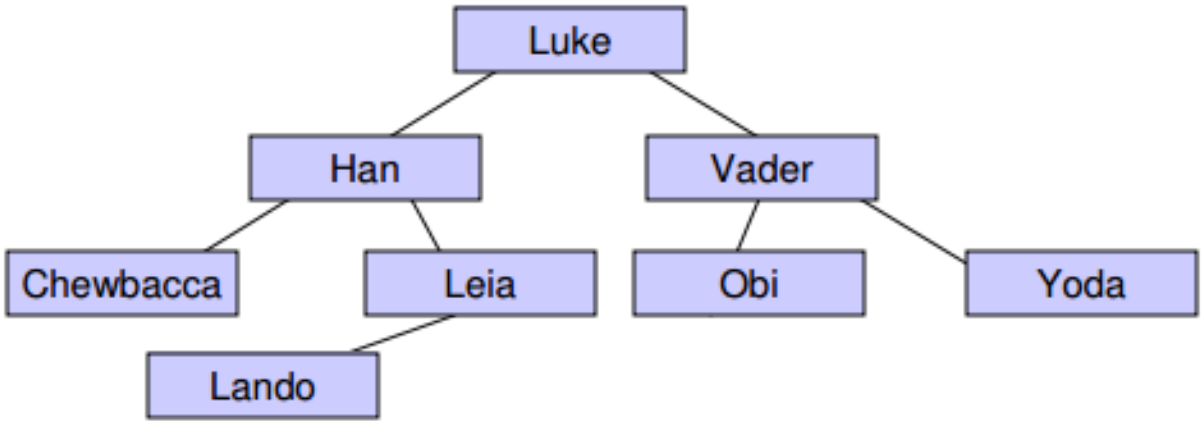


a b + c d - * e f + /

(will revisit this later)

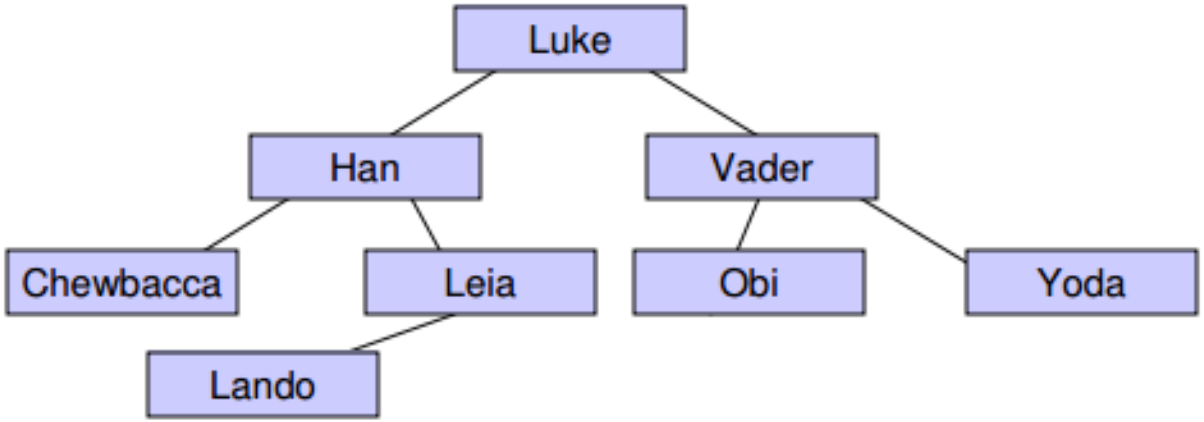
Expectations

- Given a tree, you are expected to know how to do the pre-, in-, and post-order traversals
- Exercise: Write the 3 traversals of the given tree



Exercise

- Write the pre-, in-, and post-order traversals of the following tree:



- In-order: _____
- Pre-order: _____
- Post-order: _____

Traversal Applications

- Make a clone
- Determine tree height
- Determine number of notes
- …