# Recursive Data Structures: Trees
## -- Tree Traversals --

UNIVERSITY OF VIRGINIA
DATA SCIENCE INSTITUTE

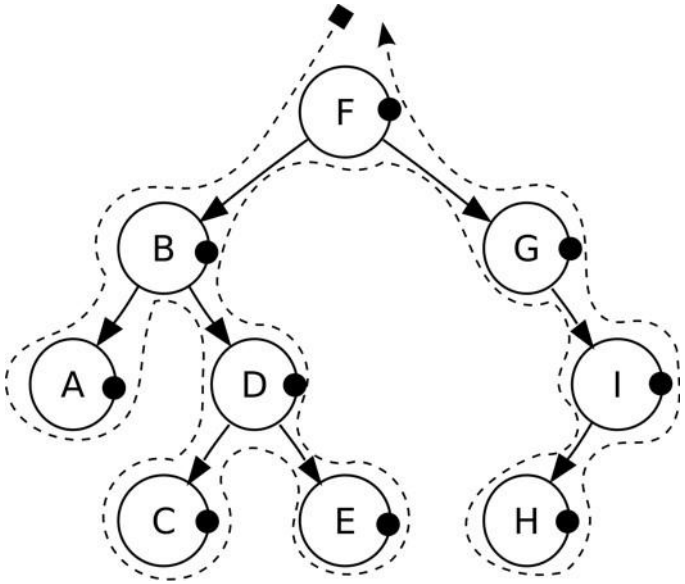
# Tree Traversals

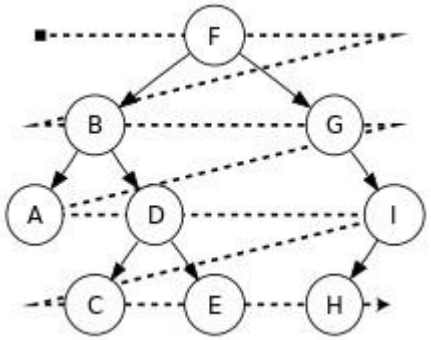
- On a recursive data structure, here are some recursive methods that allow us to traverse a binary tree

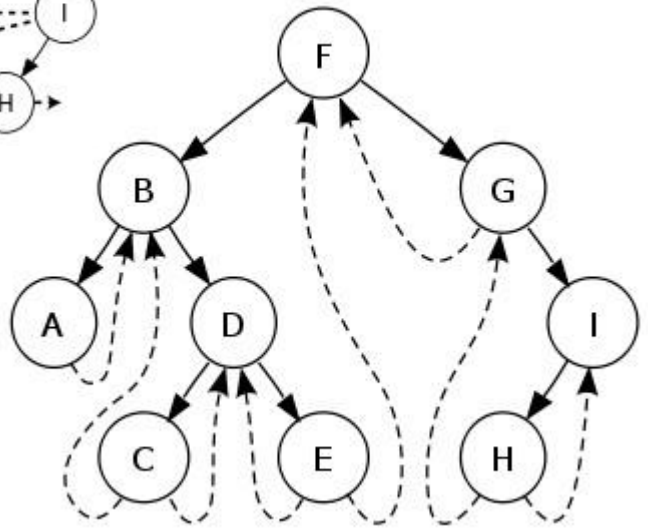
# Some Motivation…

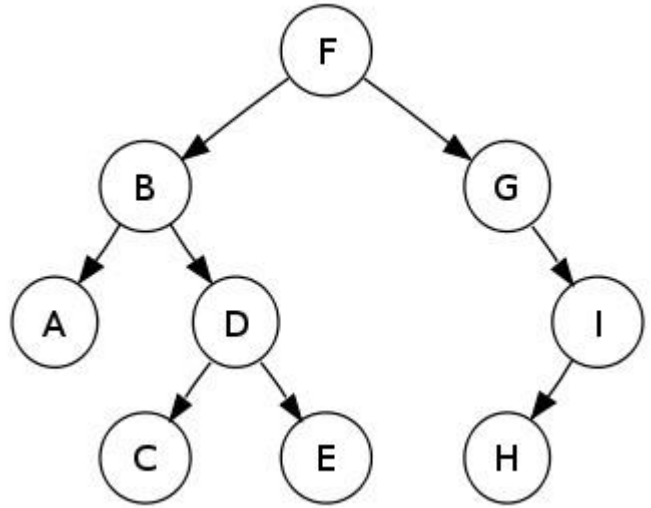
- Lists are great for keeping objects in order. They're less useful for searching
- Searching an unsorted list → O(n) (e.g. linear search)
- Searching a sorted list → O(lg n) (e.g. binary search)
  - However, takes O(n lg n) to sort…
  - And must be re-sorted as the list changes

# Tree Traversals – How?



# Tree Traversals

- A **tree traversal** 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: (**depth-first**)
  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 order of those 3 tasks***
- Visiting a node entails doing some processing at that node (often it is just **printing** – node label or its data)
- Note "**in**", "**pre**", and "**post**" refer to when we visit the root (of that subtree)

# Preoder, Inorder, Postorder

- In Preorder, the root is visited **before** (pre) the subtrees traversals
- In Inorder, the root is visited **in-between** left and right subtree traversal
- In Postorder, the root is visited **after** (post) the subtrees traversals

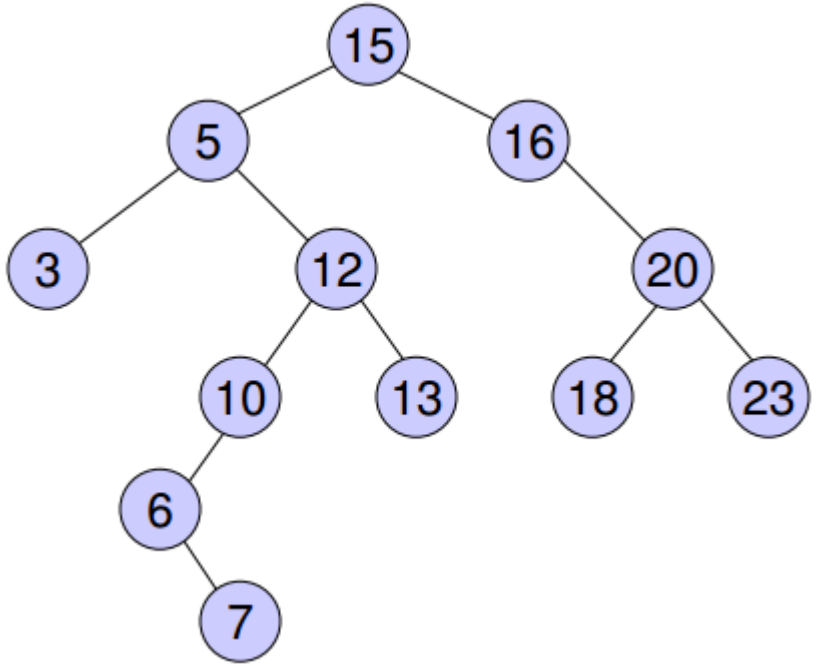
**Preorder Traversal**:
1. Visit the **root**
2. Traverse **left** subtree
3. Traverse **right** subtree

**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 [***3 methods***]

Let's do an example first…




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

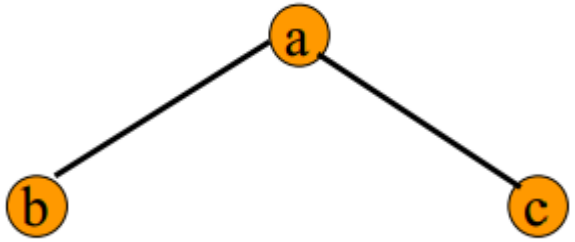
**in-order**: (left, root, right)
3, 5, 6, 7, 10, 12, 13, 15, 16, 18, 20, 23

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

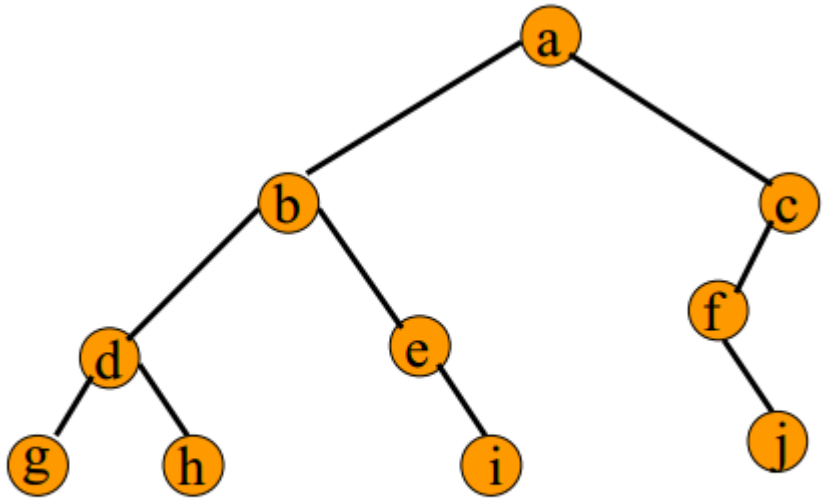
# Pre-order Traversal

- Prints in order: **root**, left, right
- It is also the simple

***depth-first search***



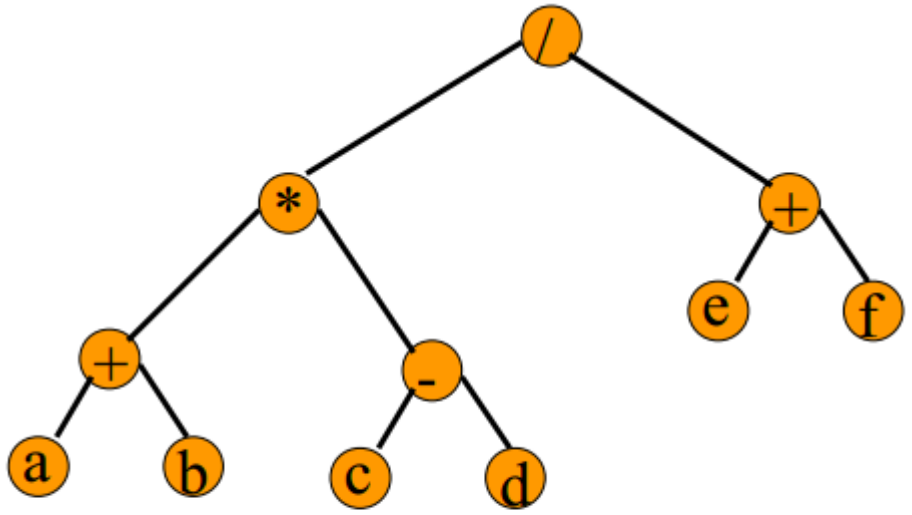

a b c




a b d g h e i c f j

# Pre-order Traversal

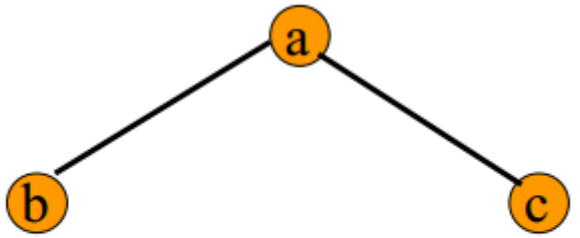
- Gives **prefix** form of expression



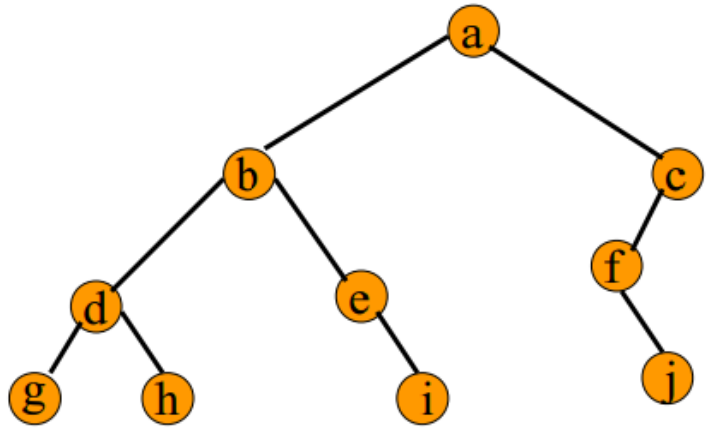

/ * + a b - c d + e f

(will revisit this later)

# In-order Traversal




b a c

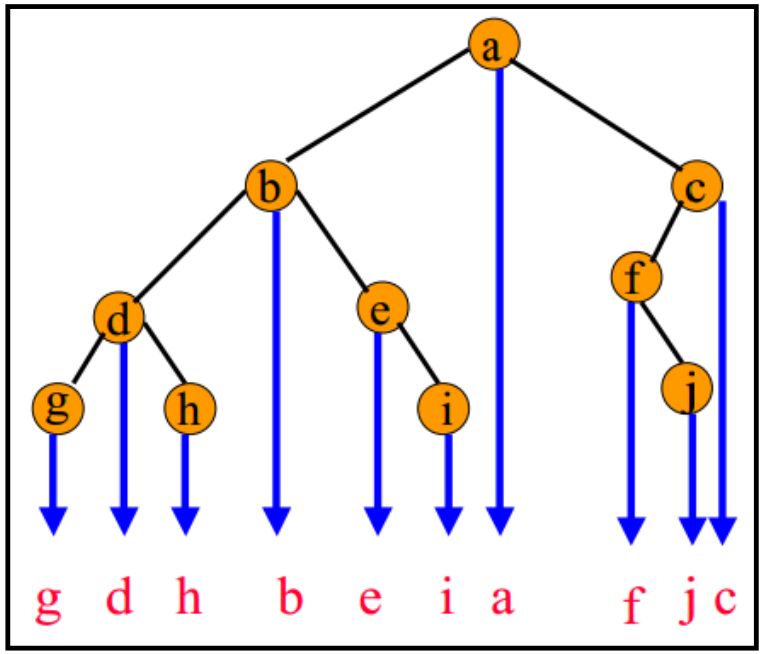


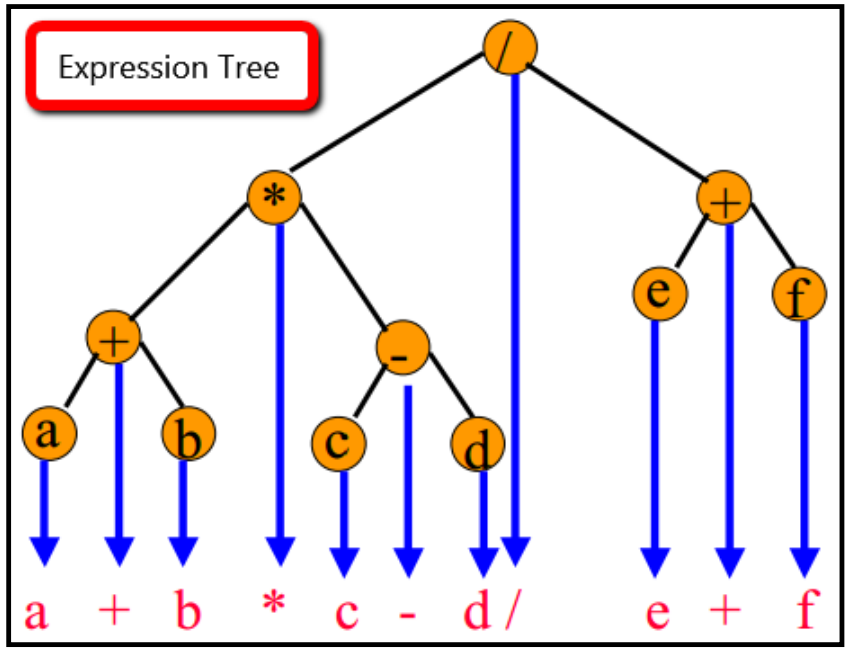

g d h b e i a f j c

- The in-order traversal **sorts the values from smallest to largest** ***for a Binary Search Tree (BST) – more on this soon!***
- (*See "**3 methods**" slide*)
- Prints in order: left, **root**, right

# In-order Traversal (Projection)

- Gives **infix** form of expression (sans parenthesis)

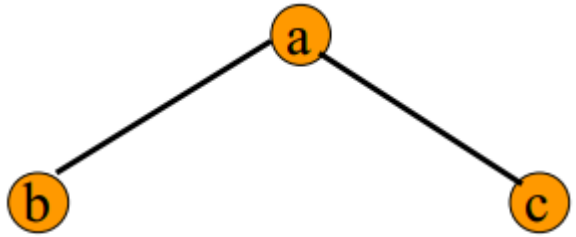


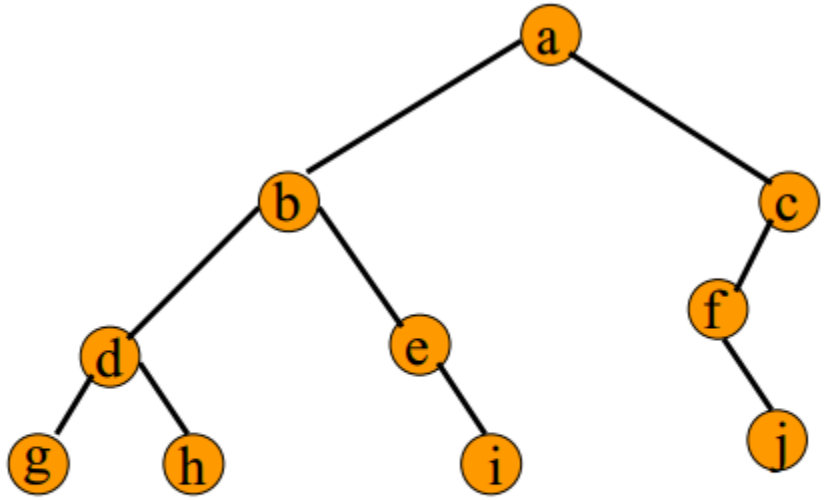

(will revisit this later)

# Post-order Traversal

- Prints in order: left, right, **root**



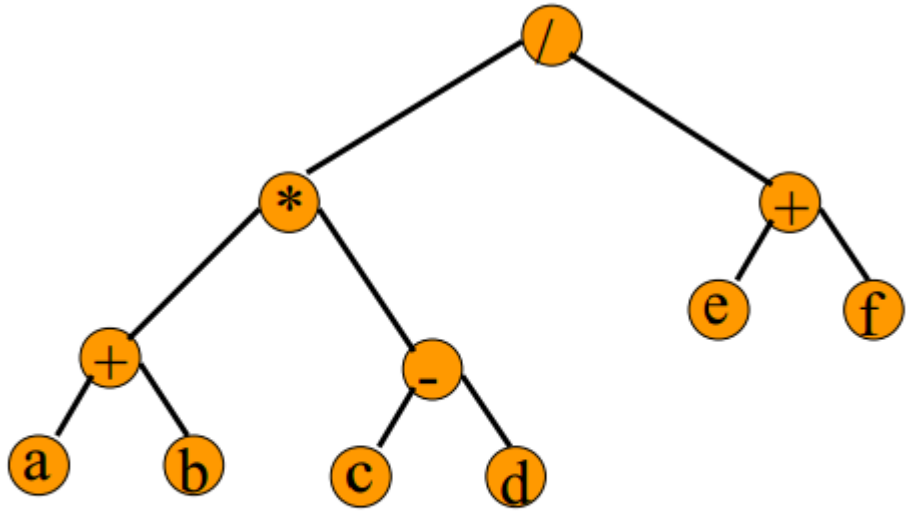

b c a




g h d i e b j f c a

# Post-order Traversal

- Gives **postfix** form of expression

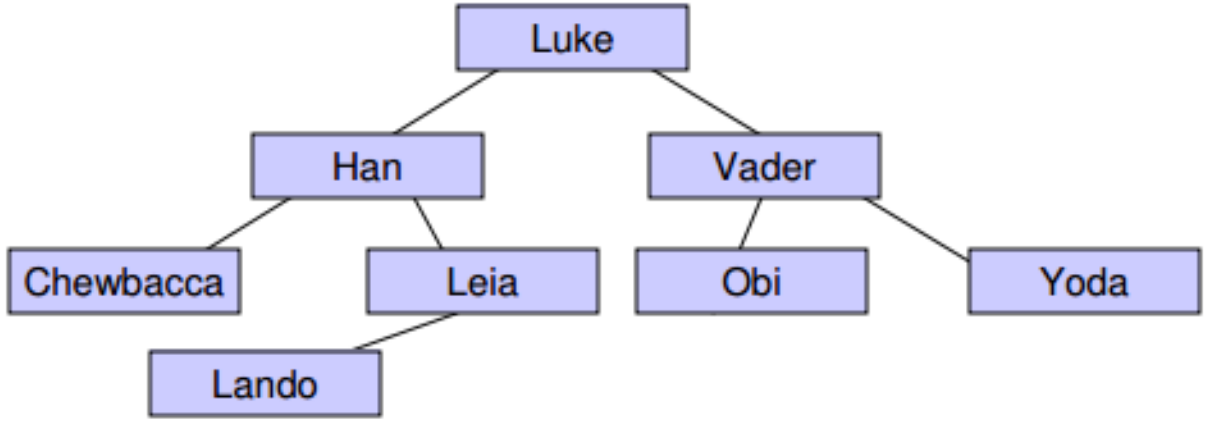



a b + c d - * e f + /

(will revisit this later)

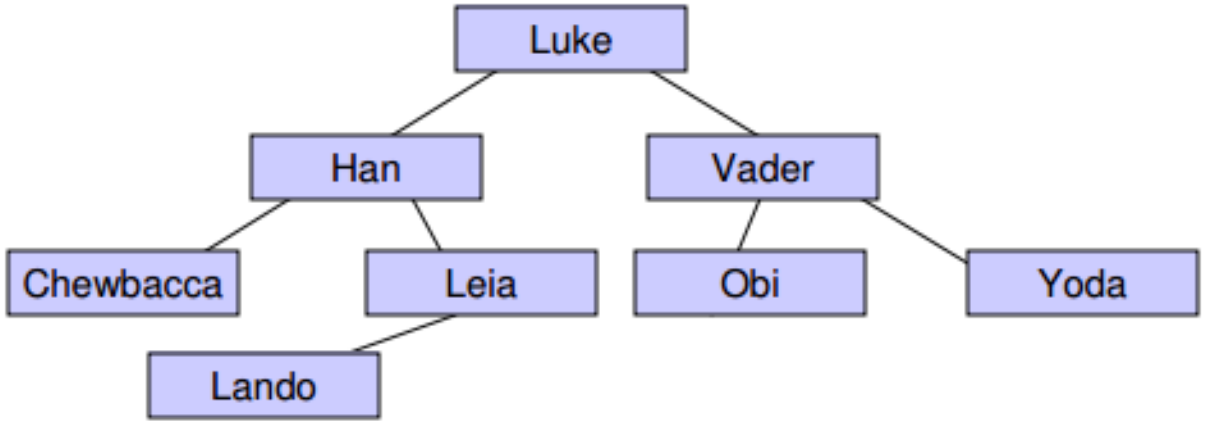
# Tree Traversal Practice

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



# Tree Traversal Practice

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




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

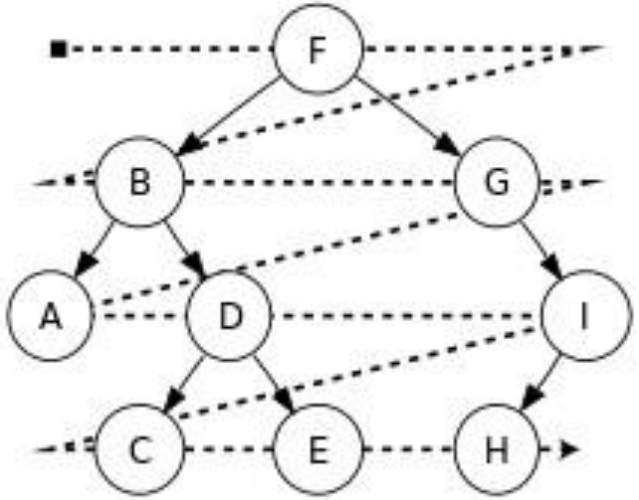
# Traversal Applications

When would we want to traverse a tree? What are some applications?

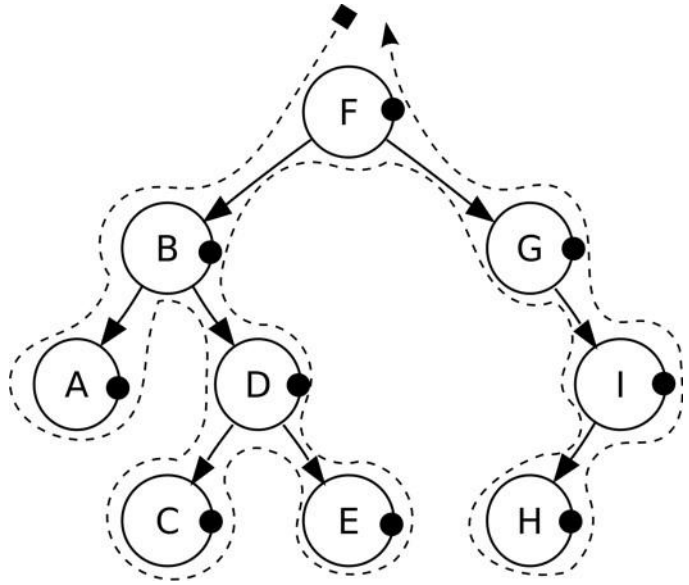
- Processing tree elements
- Make a clone (deep copy) of a tree
- Determine tree height
- Determine tree size (number of nodes)
- Searching
- …

# Depth First vs. Breadth First

Breadth First




Depth First



# Iterative Depth-First Search

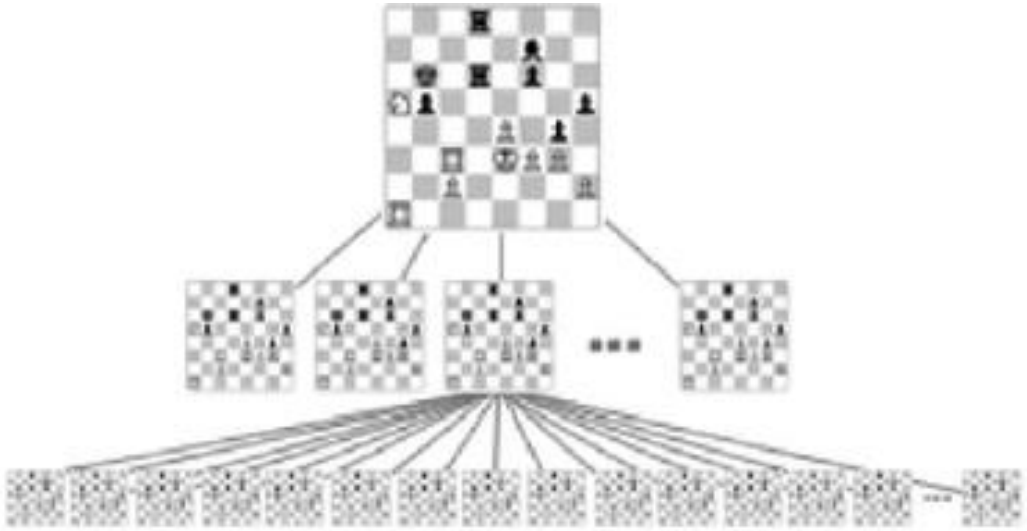
- **Depth-first search** (**DFS**) goes deeply into the tree and then backtracks when it reaches the leaves.
- DFS pseudocode algorithm using a ***Stack***!

```
stack.push(root)
while (stack is not empty):
        n = stack.pop()
        process(n) // “visit”
        for (each child of n, starting with the last one):
                stack.push(child)
```

This algorithm accomplishes a **pre-order** traversal

# When would you use Depth-First?

- Often used when simulating games
- Populate a tree with all possible chess moves
- Perform a depth-first search to find a leaf node that ends in a **win**
- Follow the moves that lead to that leaf!



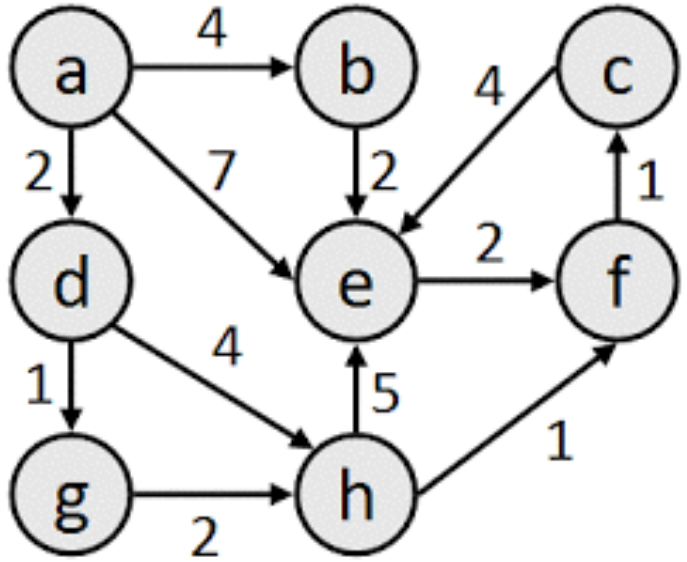
# Iterative Breadth-First Search

- **Breadth-first search** (**BFS**) visits all notes on the same level before going to the next.
- Harder to do recursively than DFS (harder to simulate a queue)
- BFS pseudocode algorithm using a ***Queue***!

```
queue.add(root)
while (queue is not empty):
        n = queue.remove()
        process(n) // “visit”
        for (each child of n):
                queue.add(child)
```

# When would you use Breadth-First?

- Breadth-First Search has an interesting property in that it can be used to find the **shortest path** between two nodes
- See **Dijkstra's algorithm**




(not a tree)

# Additional Tree Traversal Practice!

- Challenge yourself with this giant, wavy, binary tree!

For the following tree, determine what would be the result of:

**In-order traversal,**
**Pre-order traversal, and**
**Post-order traversal**

On this binary tree, show:

- **in-order**,
- **pre-order**, and
- **post-order** traversal