School of
Information Systems

SMU
SINGAPORE MANAGEMENT UNIVERSITY

COR-IS1702:
COMPUTATIONAL THINKING
WEEK 7: BINARY TREES

2020/21 Term 1

(07) Trees Part 1: Binary Trees

Video (16mins): https://youtu.be/iCDCV_CM9zU

Road Map

Algorithm Design and Analysis

(Weeks 1 - 5)

Fundamental Data Structures

- Week 6: Linear data structures (stack, queue)
- This week → Week 7: Hierarchical data structure (binary tree)
- Week 9: Networked data structure (graph)

Computational Intractability and Heuristic Reasoning

(Weeks 10 - 13)

Learning Outcomes

- Understand the operations of hierarchical data structures, primarily binary trees and binary search trees
- Able to apply the hierarchical data structure in different application contexts

Good Things Come in Pairs

Modeling hierarchical relationships using binary trees

- Tree
- Binary Tree
- Binary Search Tree

Reference

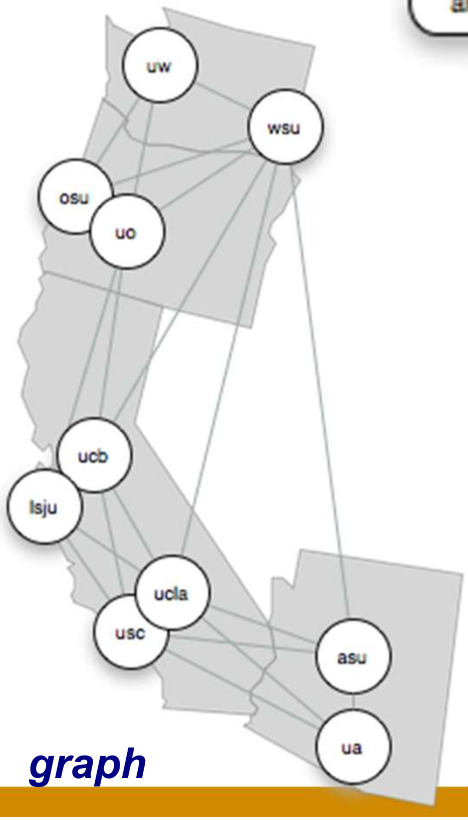
- Handout on Fundamental Data Structures Chapter 2

Recap: Linear Data Structures

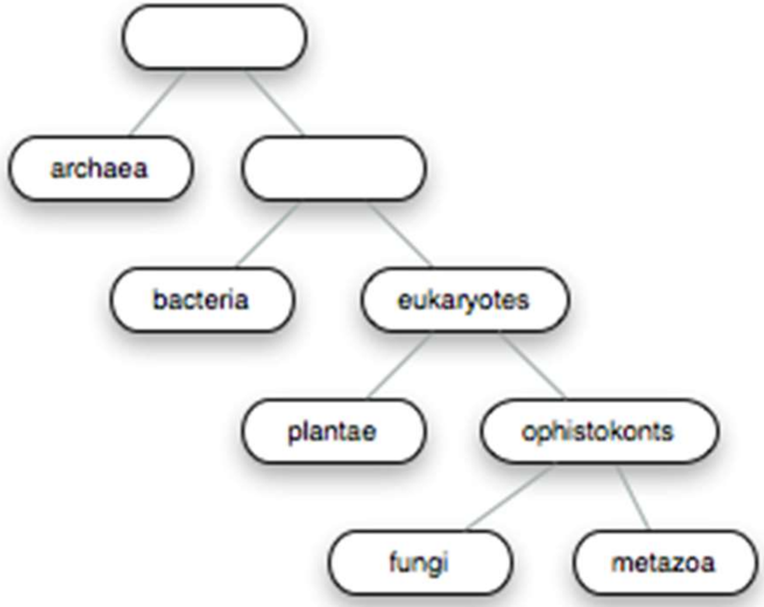
- Only one data element is accessible at any point of time
- Simplifies the client's programming logic
  - no need to keep track of indices
- The key question is **which** data element is accessible
- Three types:
  - Stack
  - Queue
  - Priority Queue

Non-linear Data Structures

- Representing not just the data elements, but also their relationships
- Complex relationships
  - not just “before and after”
- Two types:
  - Trees
    - hierarchical
  - Graphs
    - non-hierarchical

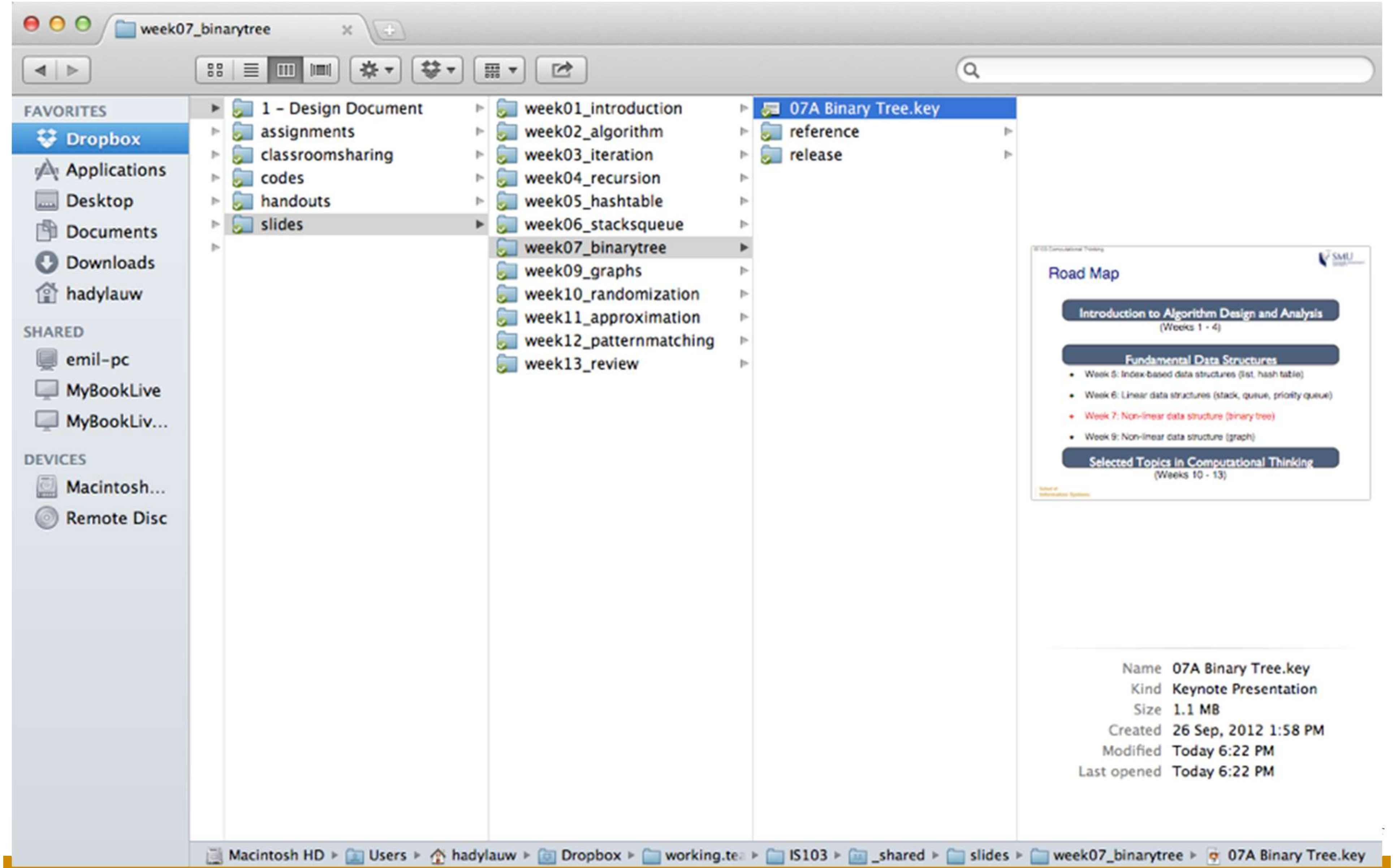


graph



tree

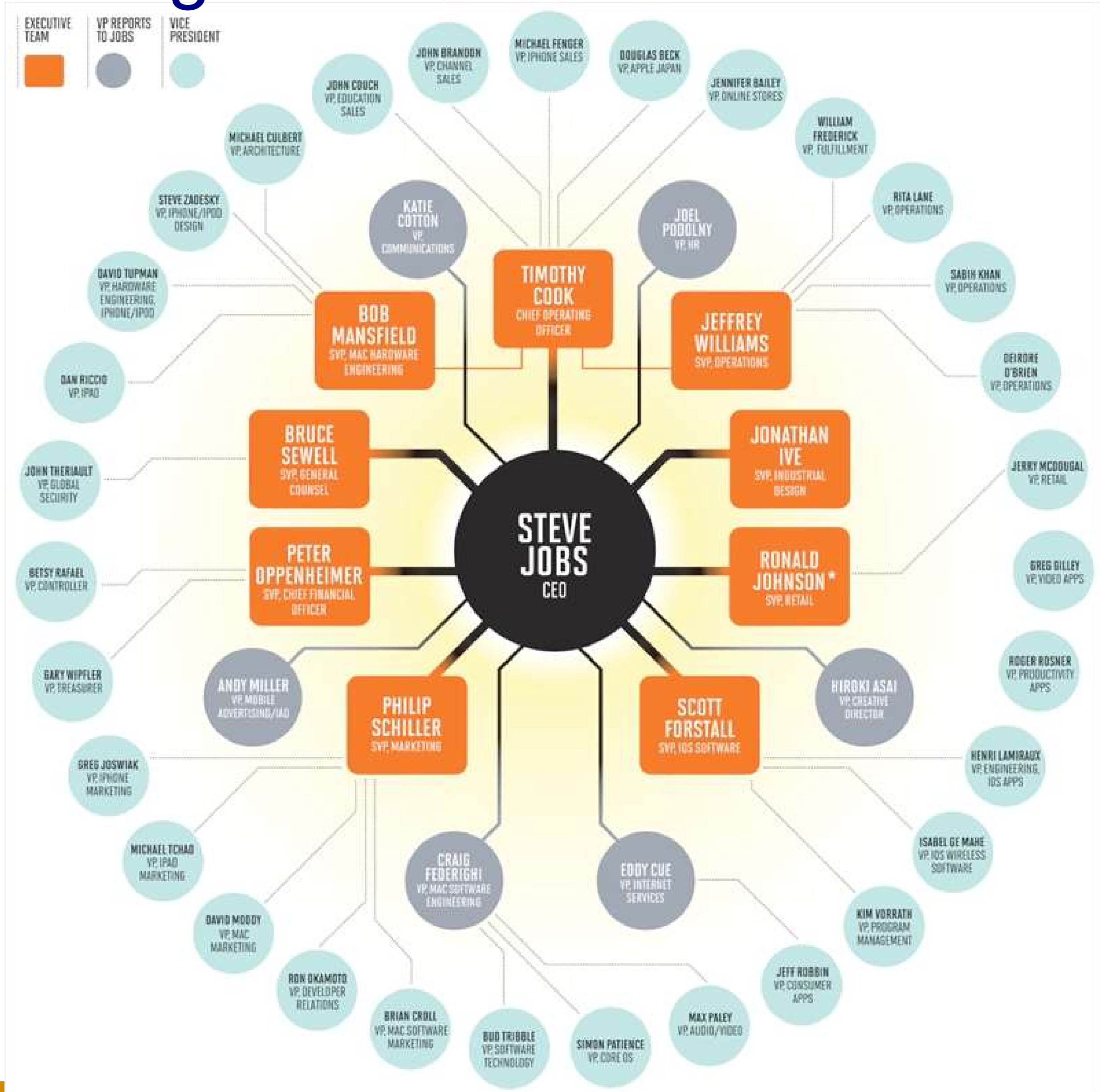
Example: File Directory Structure



Example: Family Tree

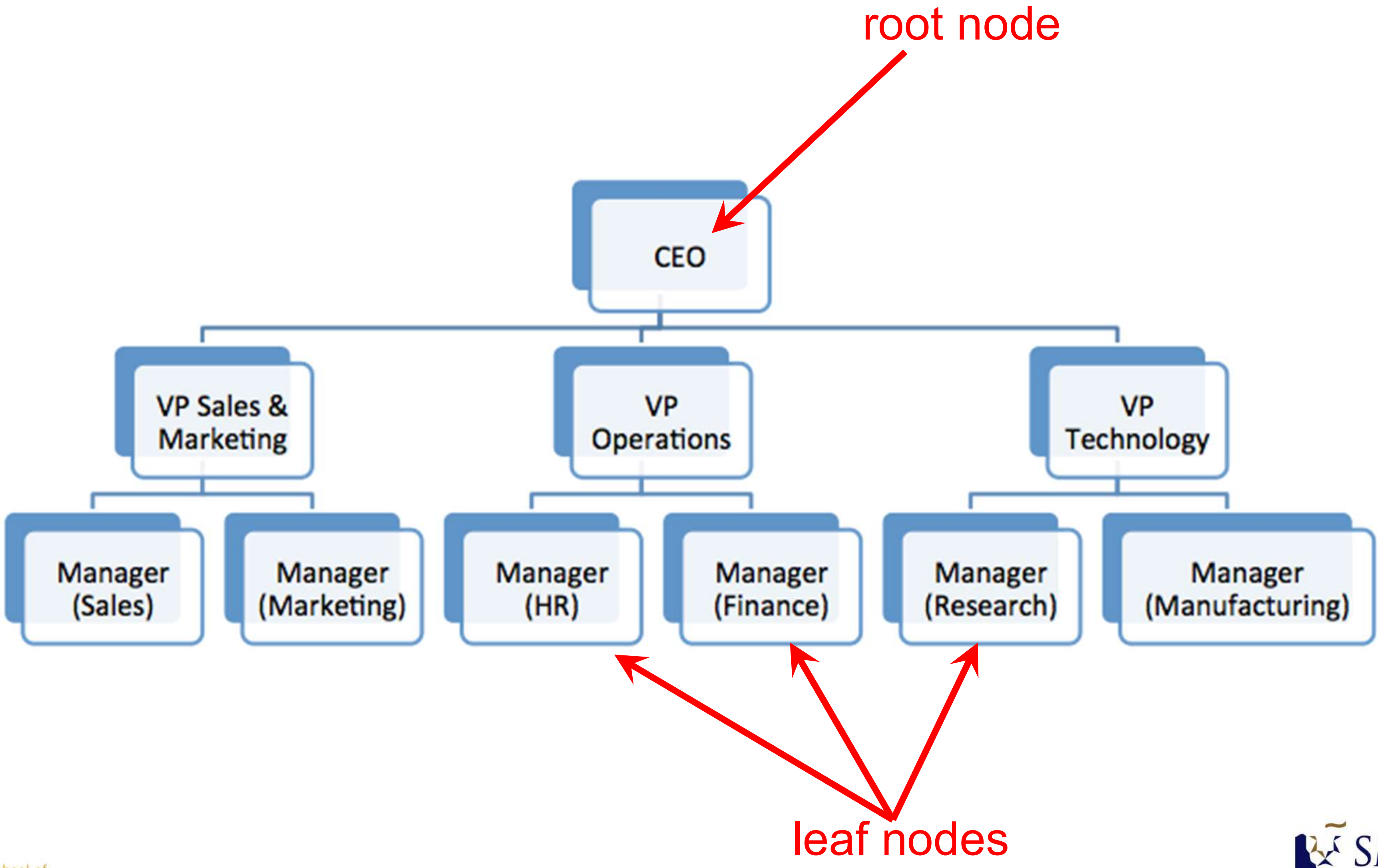
http://en.wikipedia.org/wiki/File:Family_Tree_Sample.jpg

Example: Organizational Chart



http://tech.fortune.cnn.com/2011/08/29/rethinking-apples-org-chart/

Tree


root node
CEO
VP Sales & Marketing
VP Operations
VP Technology
Manager (Sales)
Manager (Marketing)
Manager (HR)
Manager (Finance)
Manager (Research)
Manager (Manufacturing)
leaf nodes

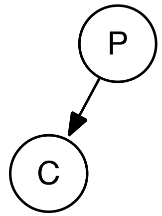
Level and Height

- The root node is at level 1 of the tree
  - Children of the root node are at level 2, and so on

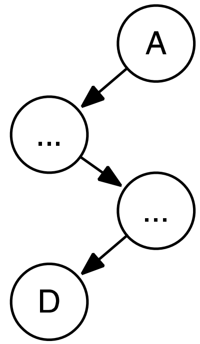
- Height of the tree is the maximum level among any of its nodes
  - An empty tree has a height of 0

Relationships

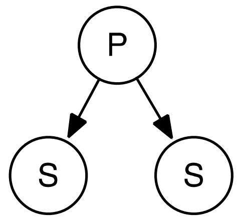
- Parent and child
  - a parent node is directly related to a number of children nodes
  - e.g., CEO is the parent of the three VP-level nodes



- Ancestor and descendant
  - an ancestor is indirectly or directly related to a number of descendant nodes through transitive parent-child relationships
  - e.g., CEO is the ancestor of all the Manager-level nodes

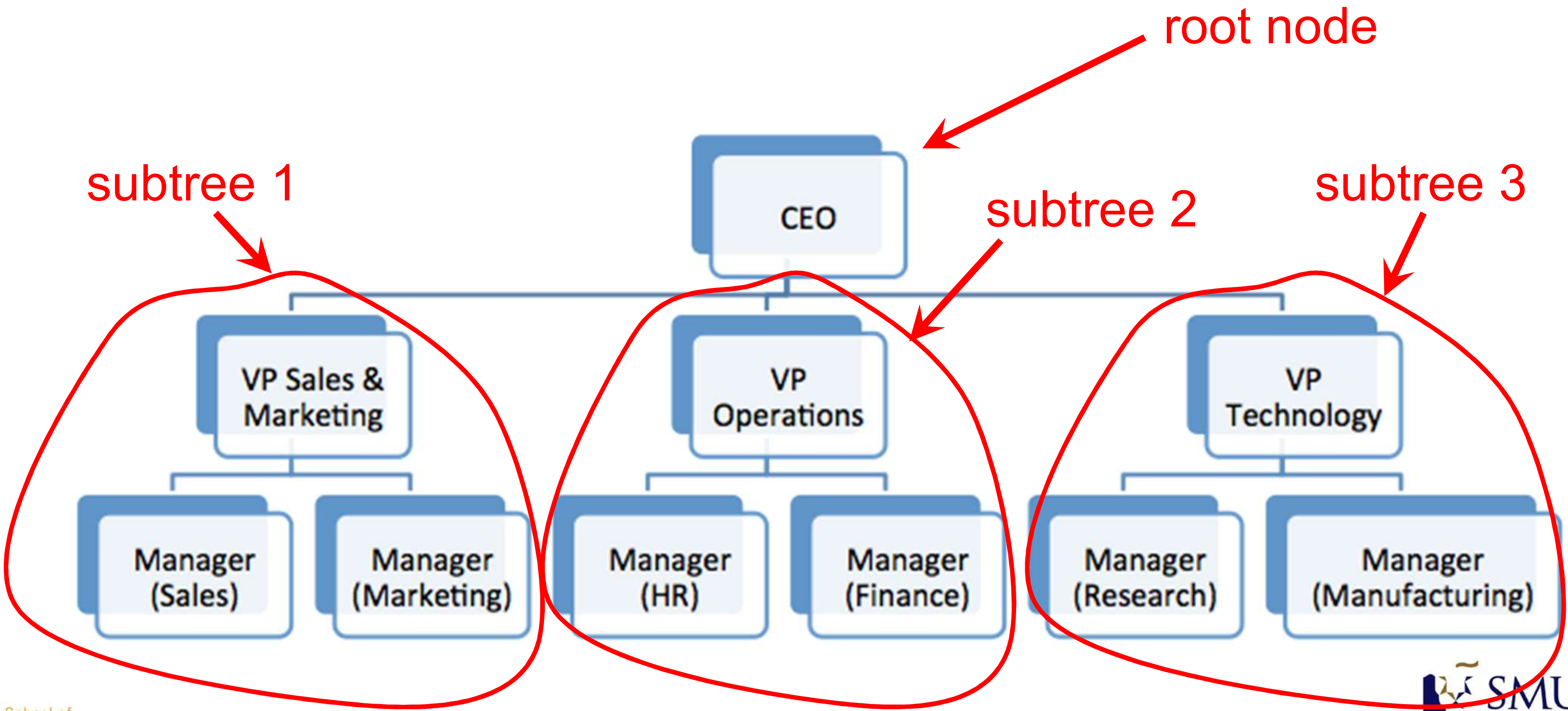


- Siblings
  - children nodes of the same parent node
  - e.g., Manager (Sales) and Manager (Marketing) are siblings



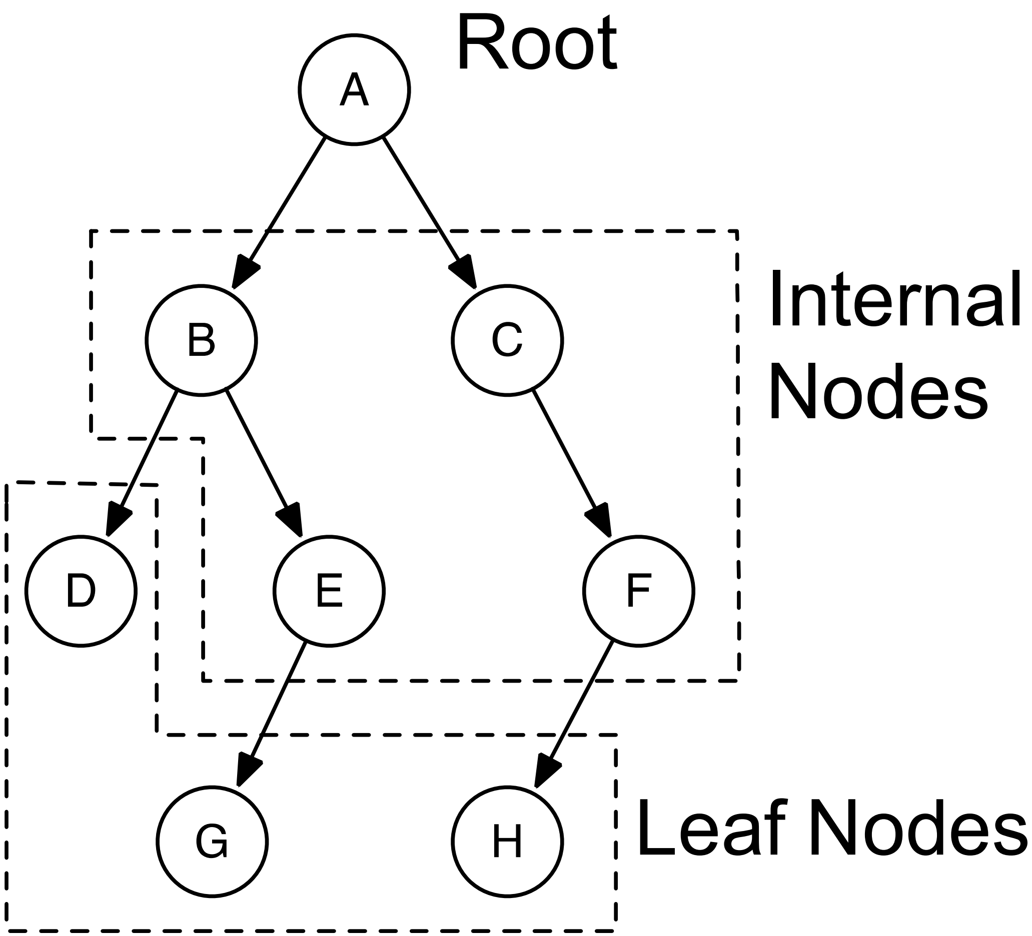
Recursive Definition of a Tree

- A tree consists of:
  - a root node `n`
  - zero or more sub-trees, each rooted at a child of `n`



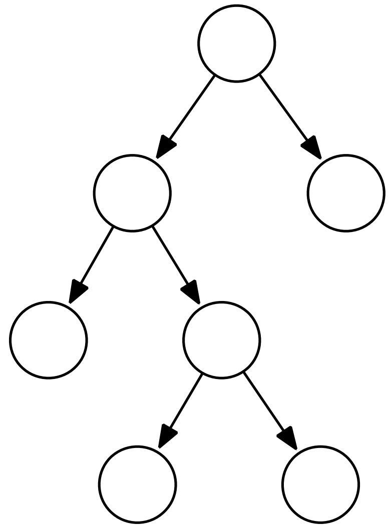
Binary Tree

A tree where each parent can have at most two children.

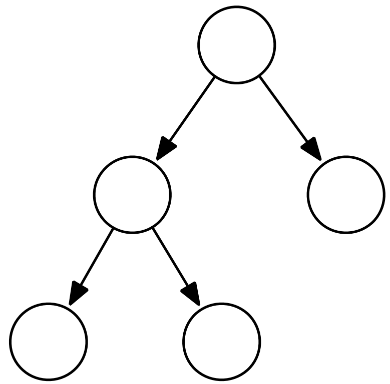


Types of Binary Tree

- Full Binary Tree:
  - Each node has exactly zero or two children.

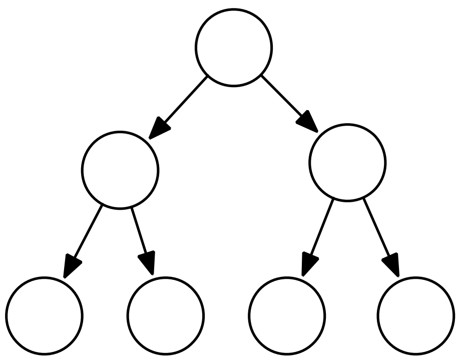


- Complete Binary Tree:
  - Every level, except possibly the deepest, is completely filled. At depth *h*, the height of the tree, all nodes must be as far left as possible.



Types of Binary Tree

- Perfect Binary Tree:
  - All leaf nodes are at the same depth; all internal nodes have degree 2.

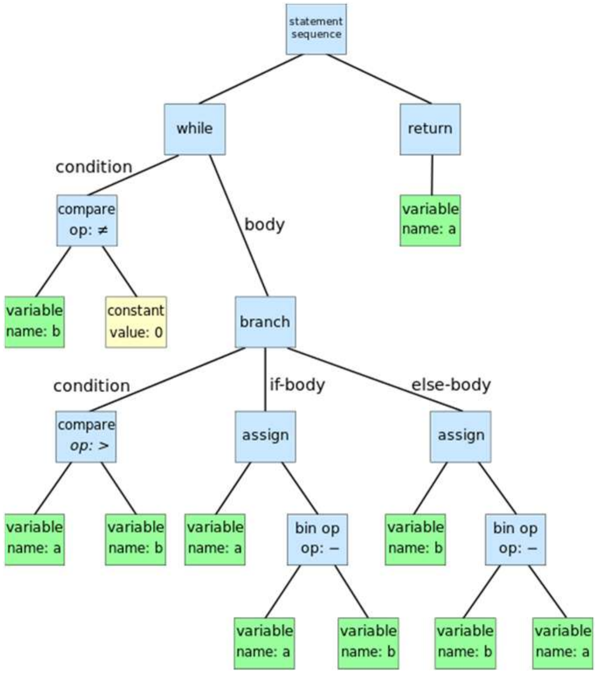


How many nodes does a perfect binary tree with height *h* have?

Why is Binary Tree Important?

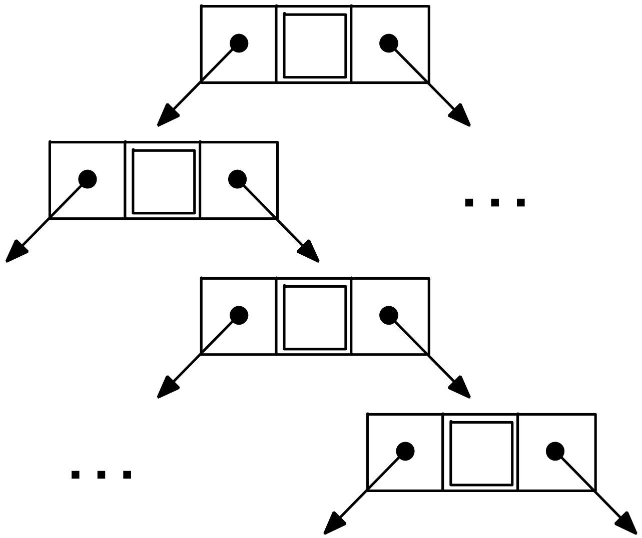
- **Heaps**: Used in implementing efficient priority-queues.
- **Huffman Coding Tree**: Commonly used in data compression methods, such as ZIP, JPEG, and MP3.
- **Syntax Tree**: How computer interprets your programs, and how calculators interprets your arithmetic expressions.

```
while b != 0:
  if a > b:
    a = a - b
  else:
    b = b - a
return a
```



Implementation of Binary Tree

- Binary tree is composed of independent “nodes” linked together.
- Each node contains data it stores and 2 references to the left and right children (**None** if no child follows).



`Node` Object: Operations

- `left / right`
  - returns the current left / right child of this node
- `value`
  - value currently stored in this node; can be any object

- `Node(value)`
  - creates a new node object with the specified value
- `setLeft(node) / setRight(node)`
  - set the input node to be the left / right child of this node
- `delLeft() / delRight()`
  - removes the current left / right child of this node

`BinaryTree` Object: Operations

- `root`
  - returns the root node
- `noOfNodes()`
  - returns the number of nodes in the tree
- `heightOfTree()`
  - returns the height of the tree

- `BinaryTree()`
  - creates a new binary tree object
- `setRoot(node)`
  - sets the input node to be the root node of this tree
- `isEmpty()`

Traversals of a Binary Tree

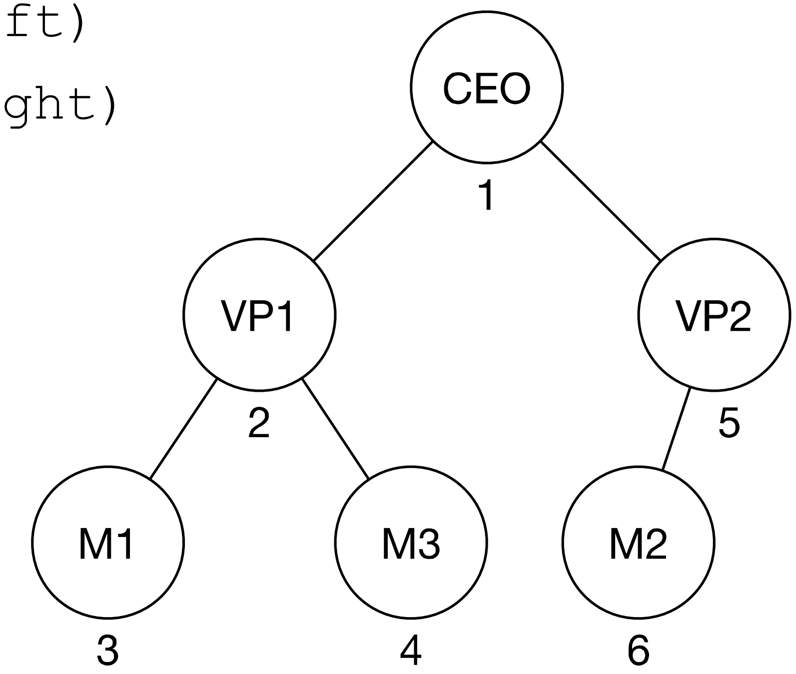
- “Walk through” the tree by visiting each node once
  - e.g., searching for a particular node

- Three common methods of traversal:
  - **preorder**
    - visiting a parent first, before visiting the left child and the right child
  - **inorder**
    - visiting the parent after the left child, but before the right child
  - **postorder**
    - visiting the parent after visiting the left and right children

- All three can be defined recursively

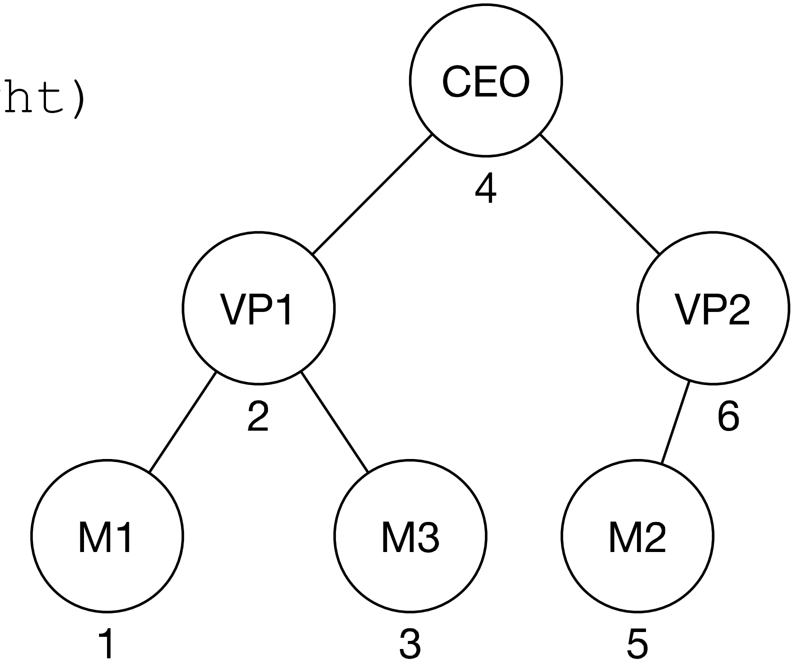
Traversal: `preorder`

```
def preorder_traverse(node):
  if node != None:
    visit(node)
    preorder_traverse(node.left)
    preorder_traverse(node.right)
```



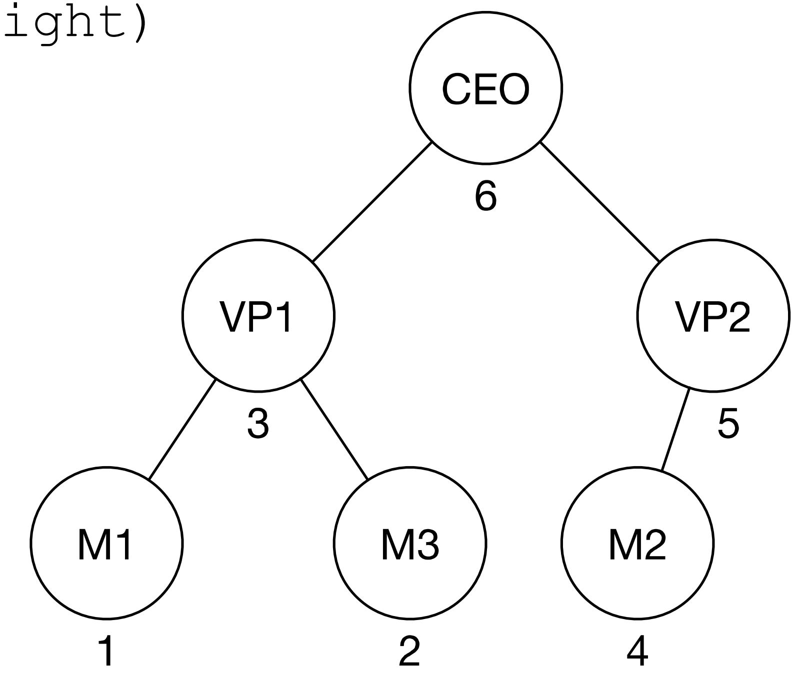
Traversal: `inorder`

```
def inorder_traverse(node):
  if node != None:
    inorder_traverse(node.left)
    visit(node)
    inorder_traverse(node.right)
```

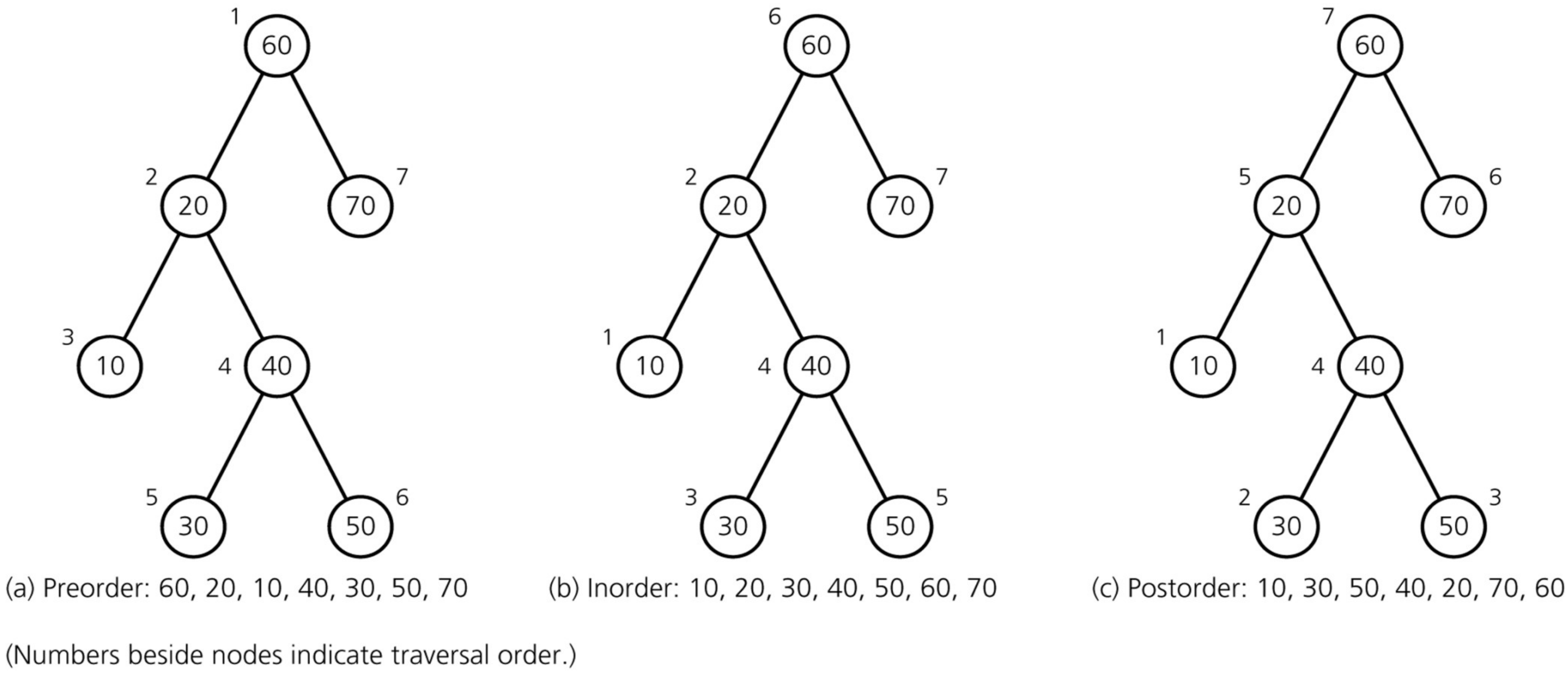


Traversal: `postorder`

```
def postorder_traverse(node):
  if node != None:
    postorder_traverse(node.left)
    postorder_traverse(node.right)
    visit(node)
```



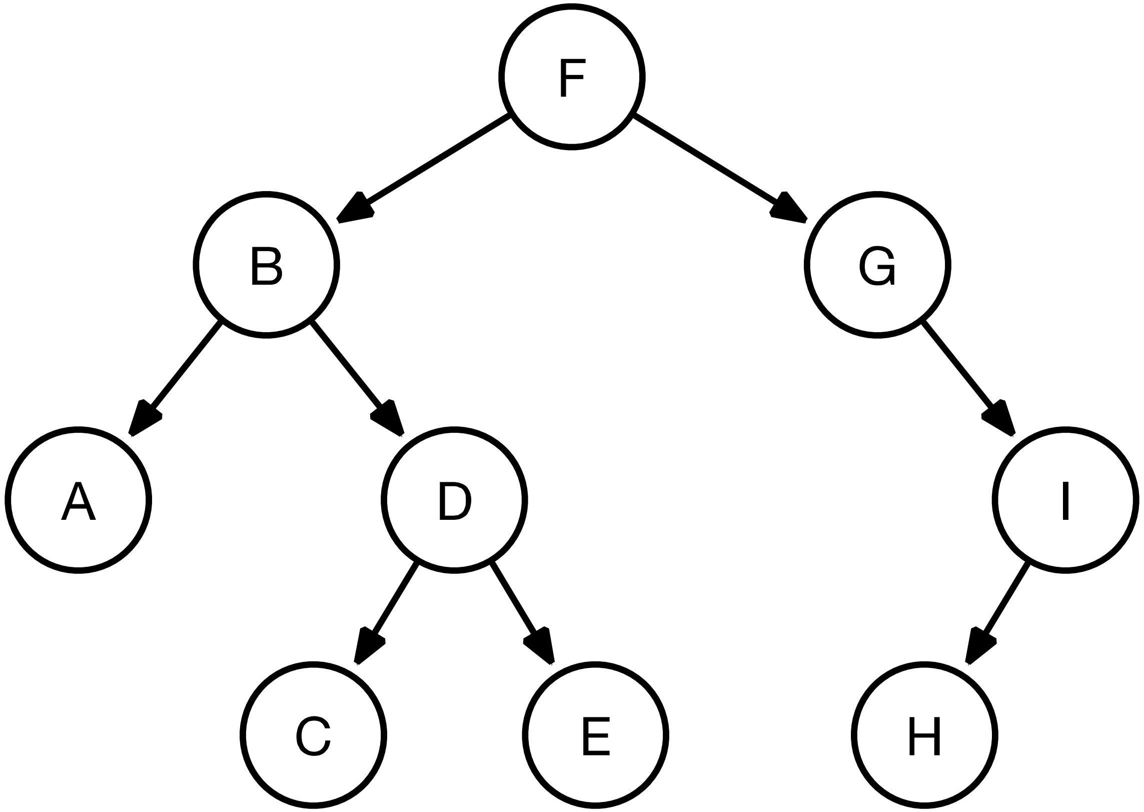
Traversal of a Binary Tree



a) preorder; b) inorder; c) postorder

Animation: http://btv.melezinek.cz/binary-search-tree.html

Practice Traversal #1



Preorder?

Inorder?

Postorder?

http://en.wikipedia.org/wiki/Tree_traversal

Practice Traversal #2