Middlebury

CSCI 201: Data Structures

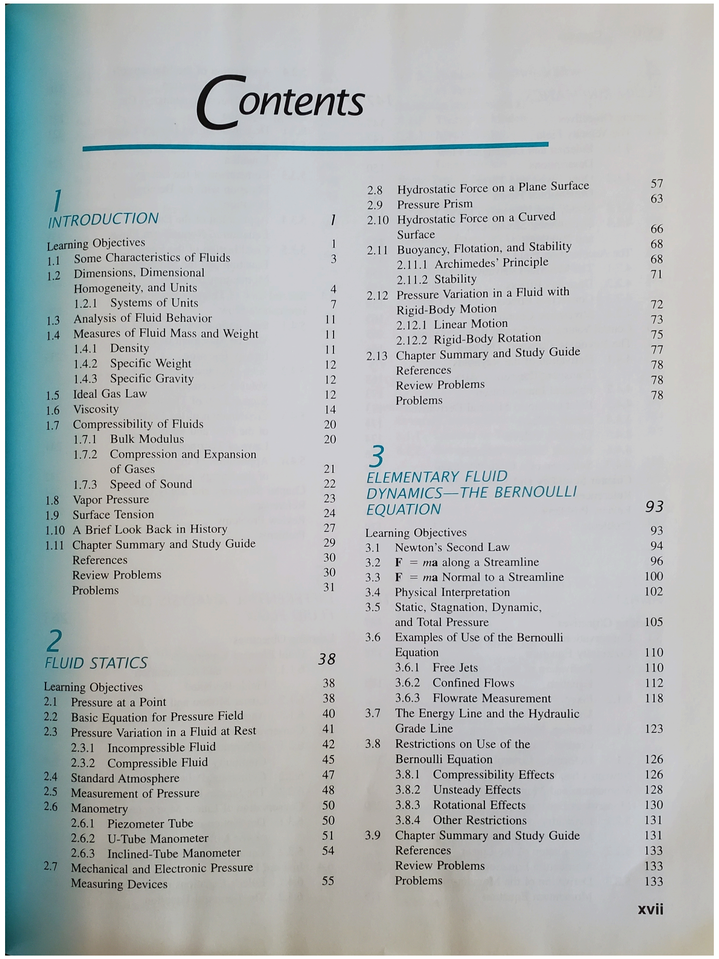
Spring 2025

Lecture 8M: Trees

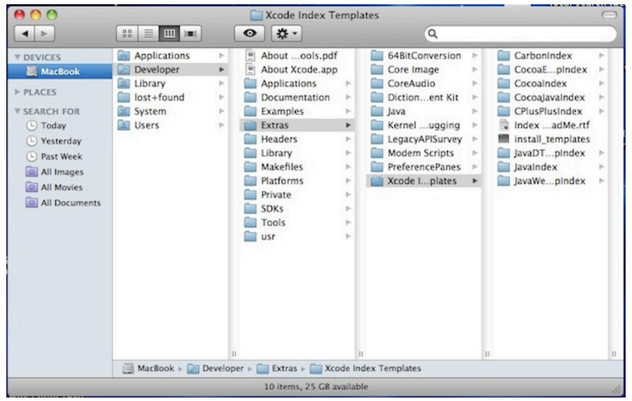
Goals for today:

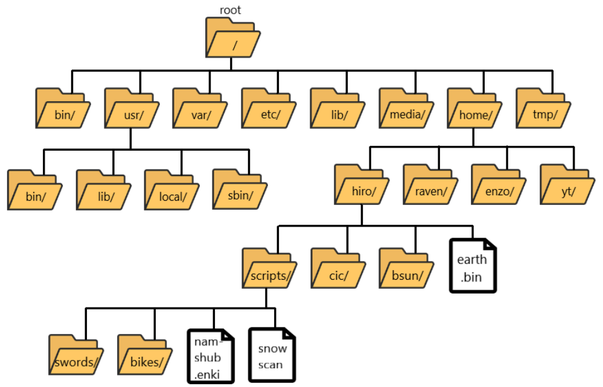
- What are **trees**? Why are they important in computer science?
- Use appropriate tree terminology: **nodes, edges, internal, leaf, parent, sibling nodes, path**.
- Identify properties of trees and nodes: k**-ary, height, degree, level, depth, full, complete**.
- **Represent** binary trees in a computer.
- Visit/process nodes in a binary tree using **pre-order, in-order, post-order** traversal.

Trees are useful to represent hierarchy in data.



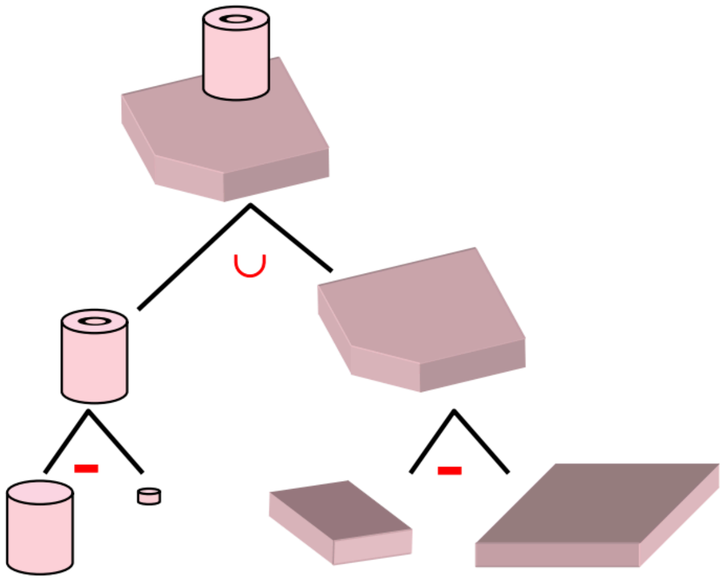
Contents

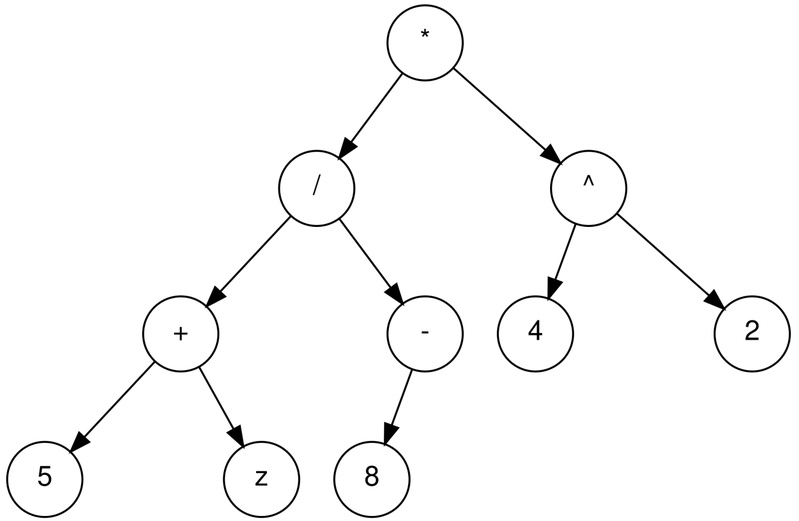




Trees are also useful for representing expressions.

https://en.wikipedia.org/wiki/Binary_expression_tree

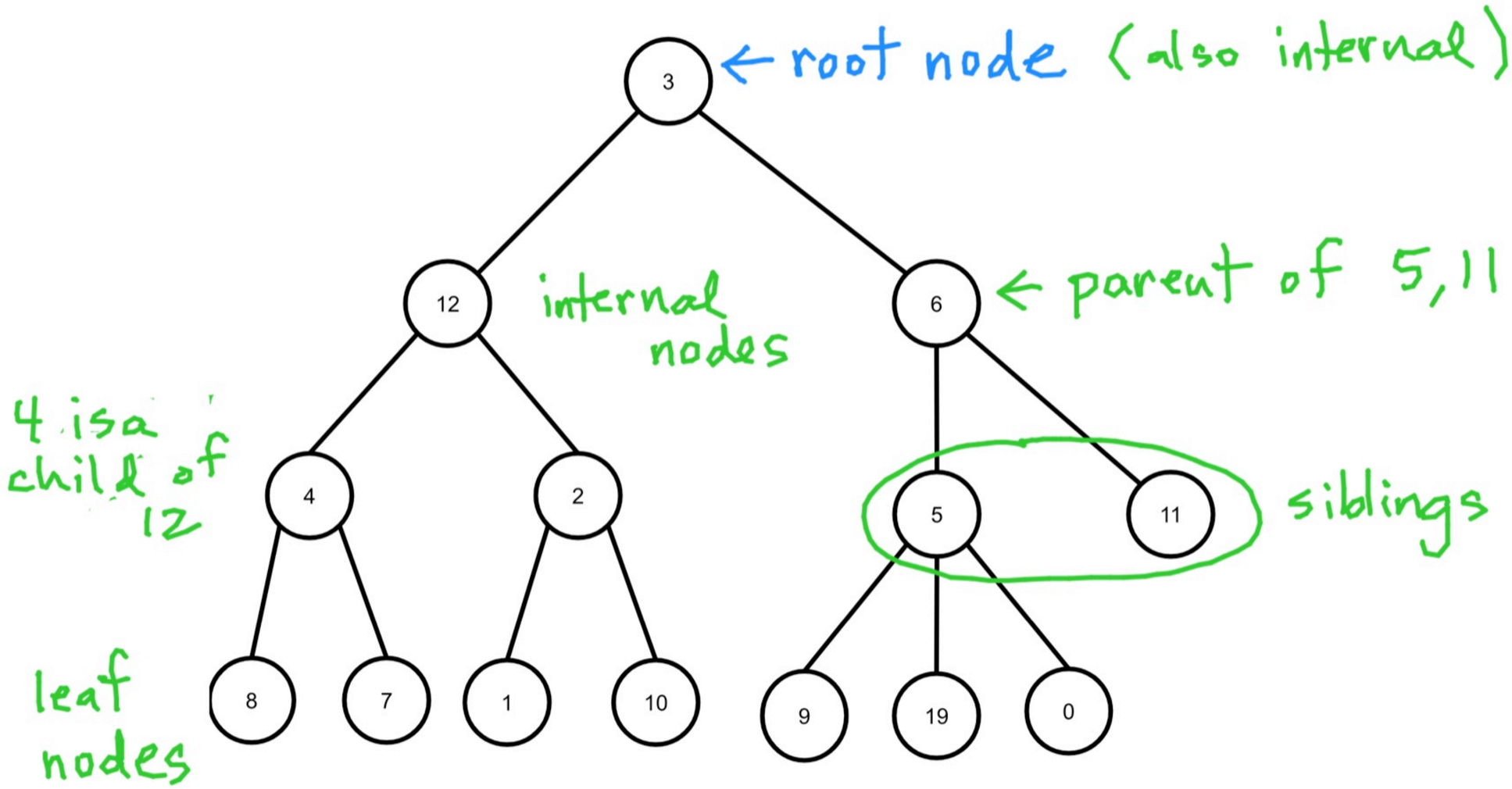




$$\frac{5+z}{-8} \cdot 4^2$$

Terminology: nodes (root, internal, leaf), parent, siblings.

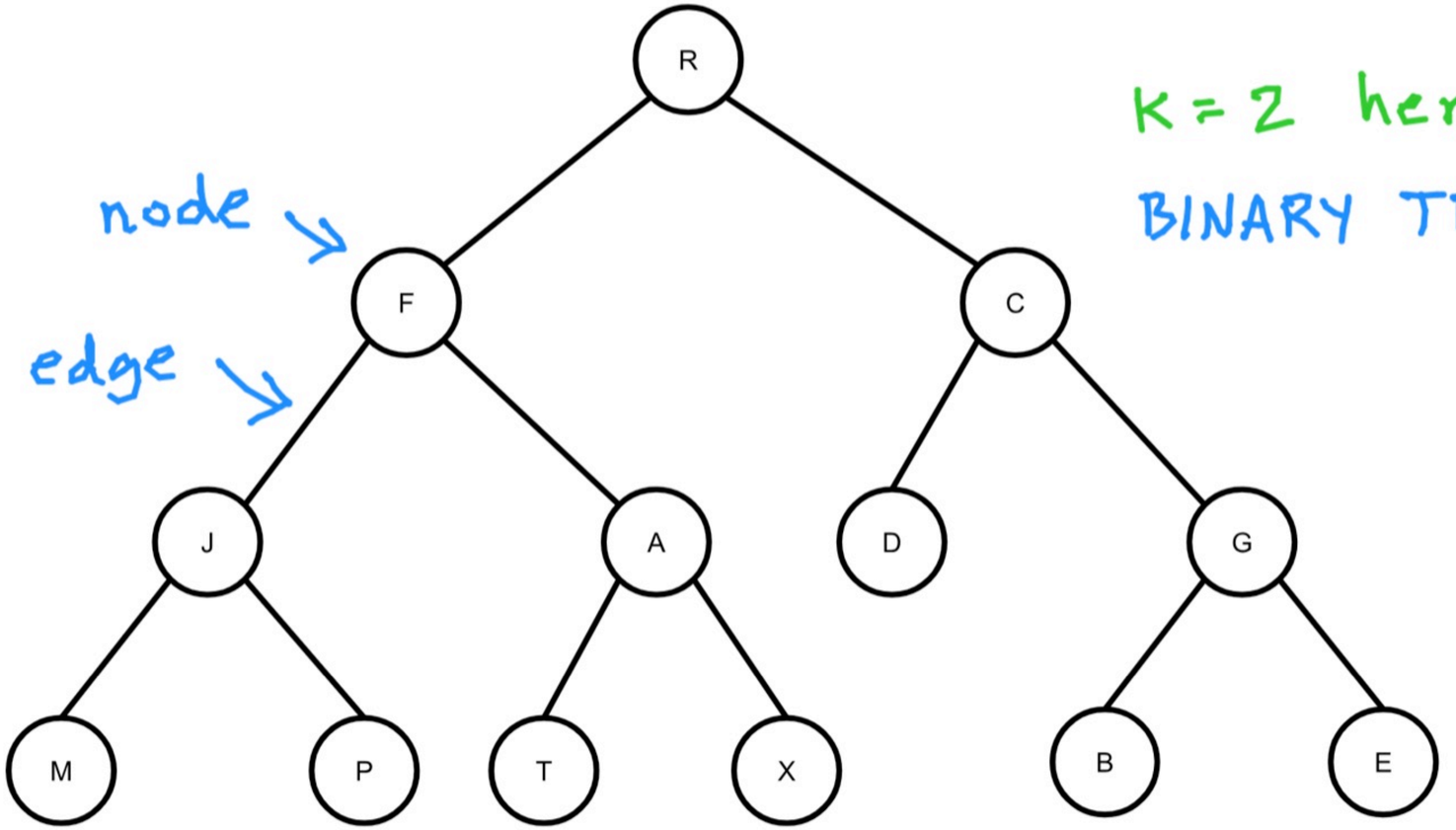
A **rooted tree** is a set of nodes based on a parent-child relationship.



k-ary trees: maximum number of children of any node is k.

The **degree** of a node is the number of children.

degree of tree is the maximum degree of any node



node

edge

k = 2 here

BINARY TREE

Paths, level, depth and height of a node.

- A **path** is the unique, shortest sequence of edges from a node to an ancestor.
- The **depth** (also level) of a node is the length of the path from the root to the node.
- The **height** of a node is the length of the longest path from that node to a leaf.

height(null): -1

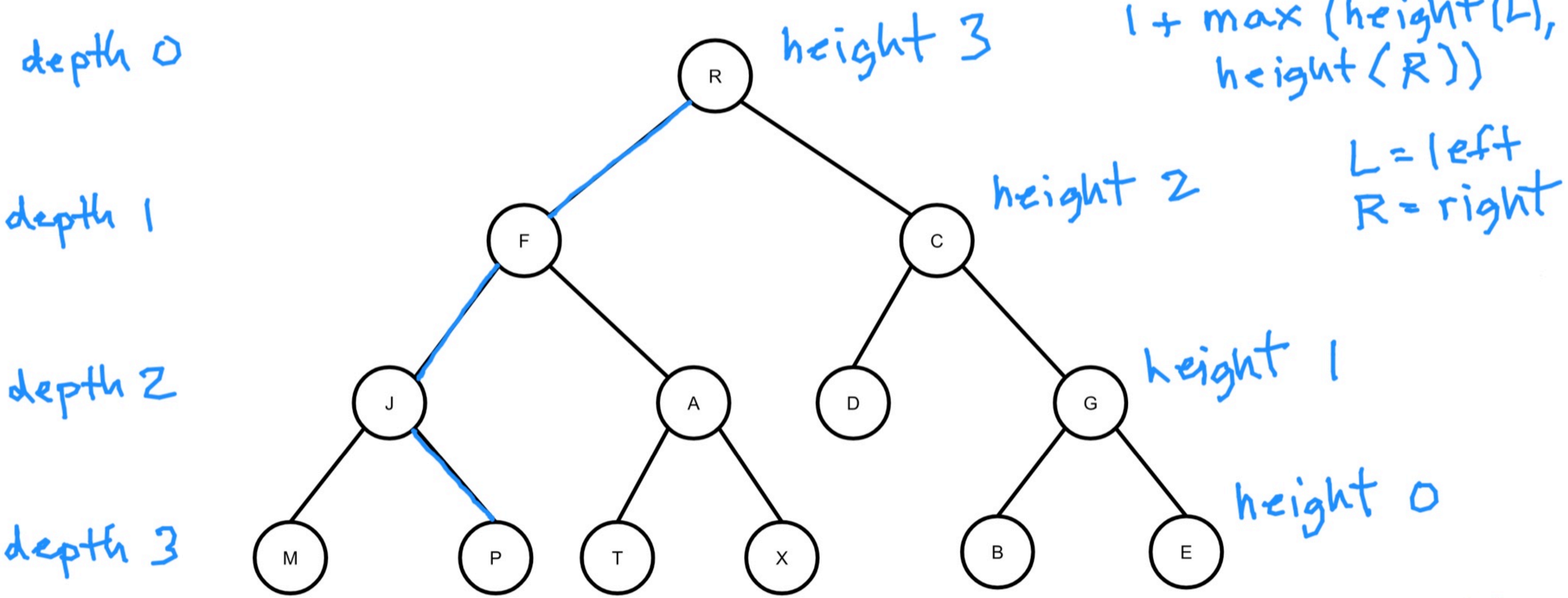
height(leaf): 0

height(node):

1 + max(height(L), height(R))

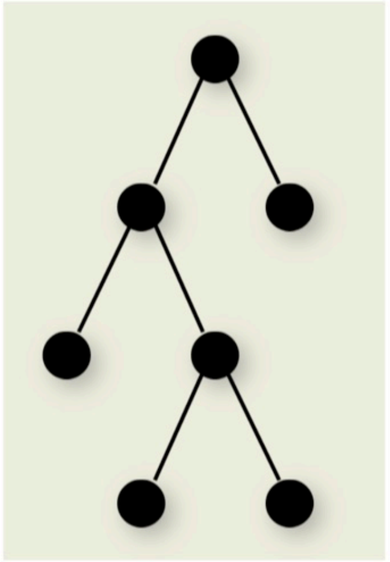
L = left

R = right

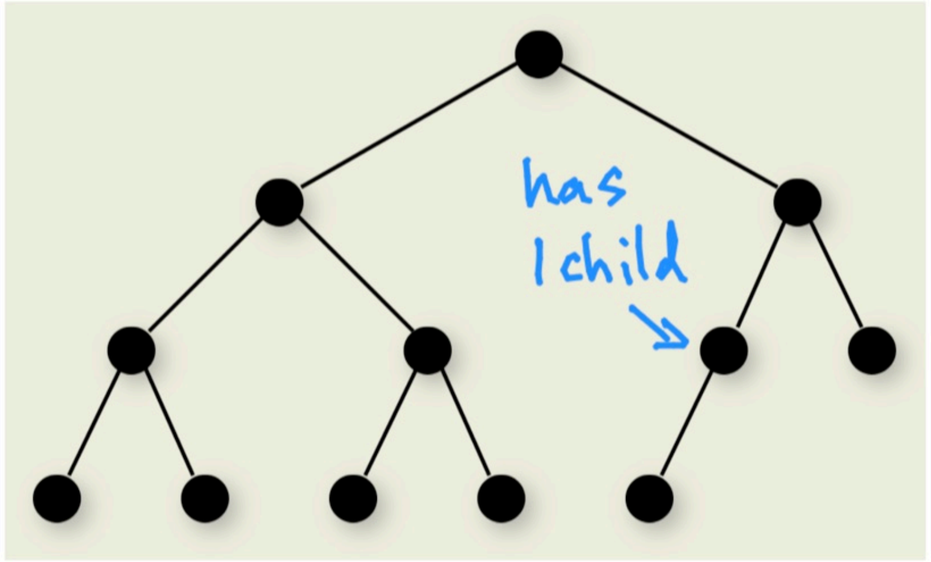


Properties of binary trees.

- **Full**: every node is either an internal node with 2 children or a leaf node (0 children).
- **Balanced:** height of left and right subtrees of any node differ by at most 1.
- **Complete**: all levels are filled except the last level, and nodes on the last level are as far left as possible. For a tree of height h, all levels except possibly level h are full. Nodes at level h filled from left to right.



full
not complete



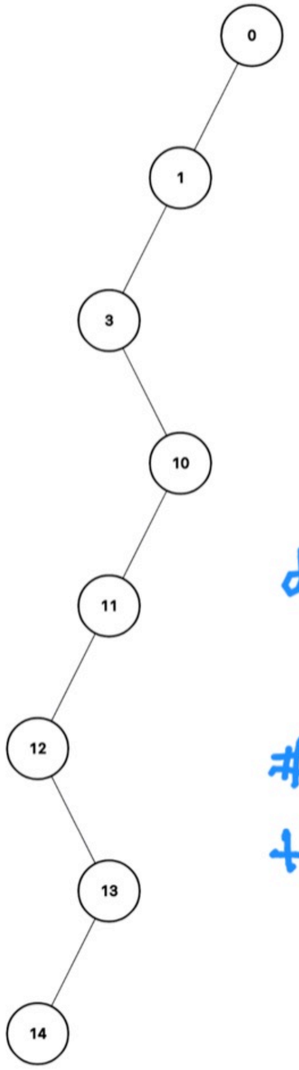
(https://opendsa-server.cs.vt.edu/ODSA/Books/sbu/cse214/fall-2022/CSE214/html/BinaryTree.html)

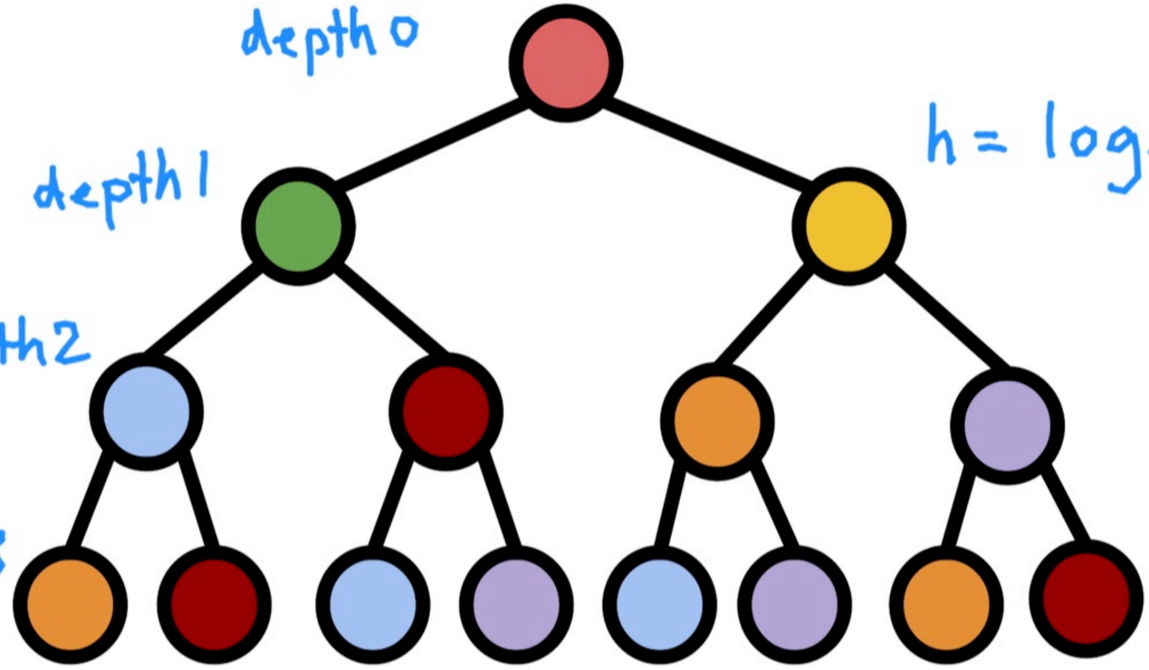
not full
complete

Limits on the height of a binary tree.

$\log_2(n+1) - 1 \leq \text{height} \leq n-1$

n nodes
n-1 edges
height = n-1





$h = \log_2(n+1) - 1$

nodes at a particular level/depth : 2^{depth}

total # nodes : $2^0 + 2^1 + 2^2 + \cdots + 2^h$ (height h)

geometric series $r = 2$

$$\frac{2^{h+1} - 1}{2-1} = 2^{h+1} - 1 = n \rightarrow h+1 = \log_2(n+1)$$

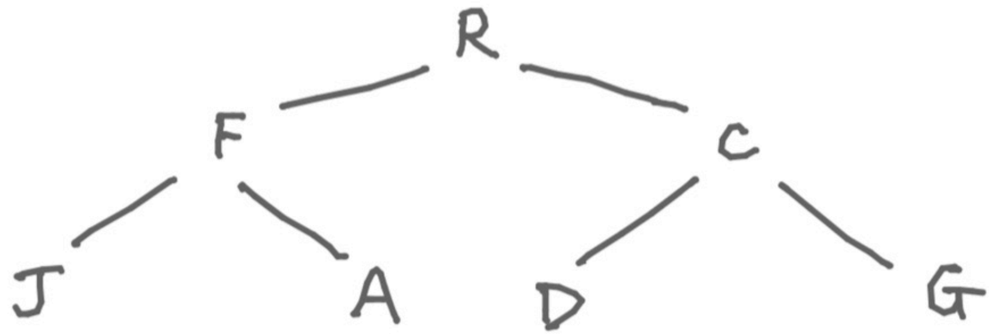
$$2^{h+1} = n+1$$

Representing trees with a computer.

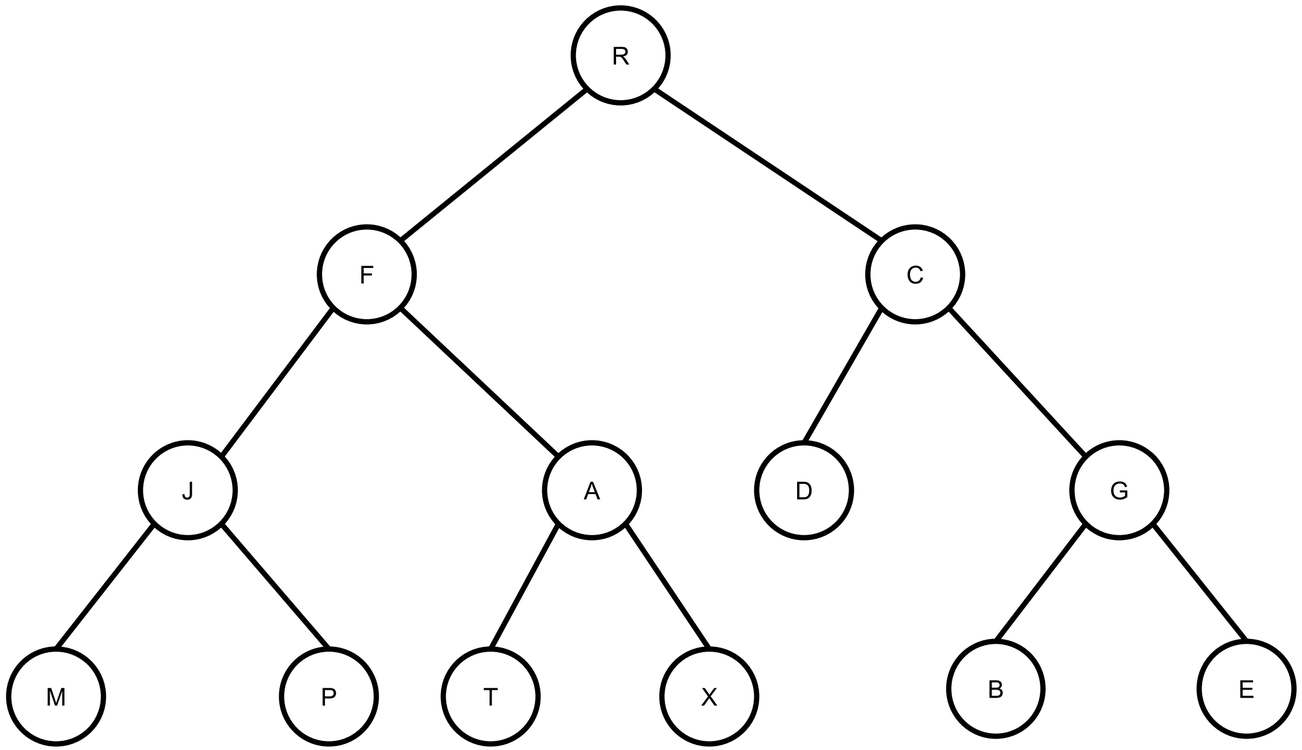
```java
class TreeNode<E> {
  private E data;
  public TreeNode<E> left;
  public TreeNode<E> right;

  public TreeNode(E data) {
    this.data = data;
    left = null;
    right = null;
  }

  public E get() {
    return data;
  }
}
```

```java
public class BinaryTree<E> {
  public TreeNode root;

  public BinaryTree() {
    root = null;
  }

  public static void main(String[] args) {
    // For now, we'll build the tree explicitly.
    BinaryTree<String> tree = new BinaryTree<>();

    tree.root = new TreeNode<String>("R");
    TreeNode<String> f = new TreeNode<>("F");
    TreeNode<String> c = new TreeNode<>("C");
    tree.root.left = f;
    tree.root.right = c;
    // ...
  }
}
```

Two things to consider: (1) building the tree, (2) traversing the tree.

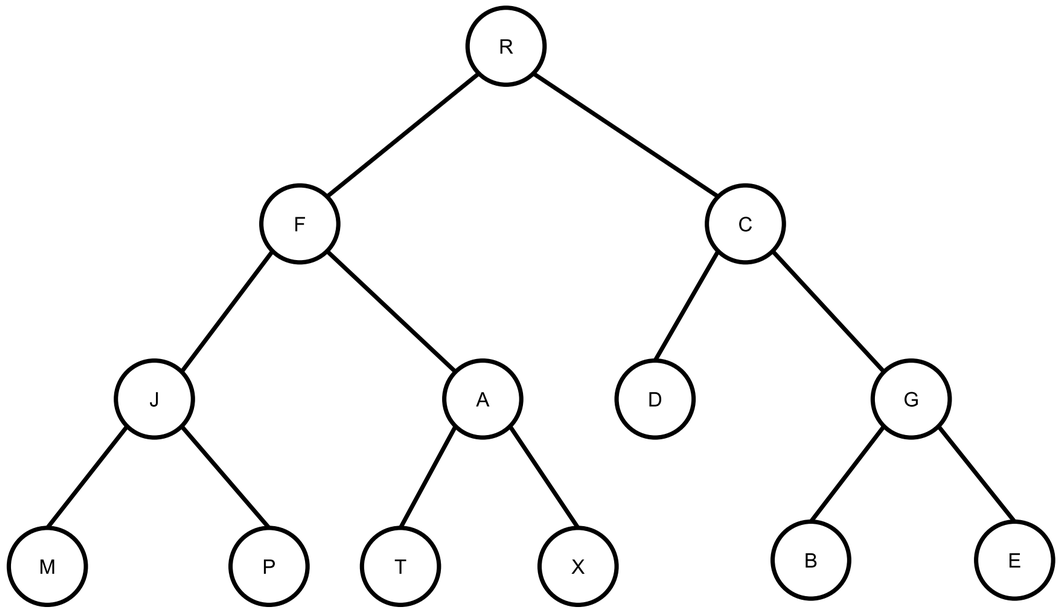


Exercise: finish manually building up this tree.



Traversing trees: pre-order, in-order, post-order, level-order.

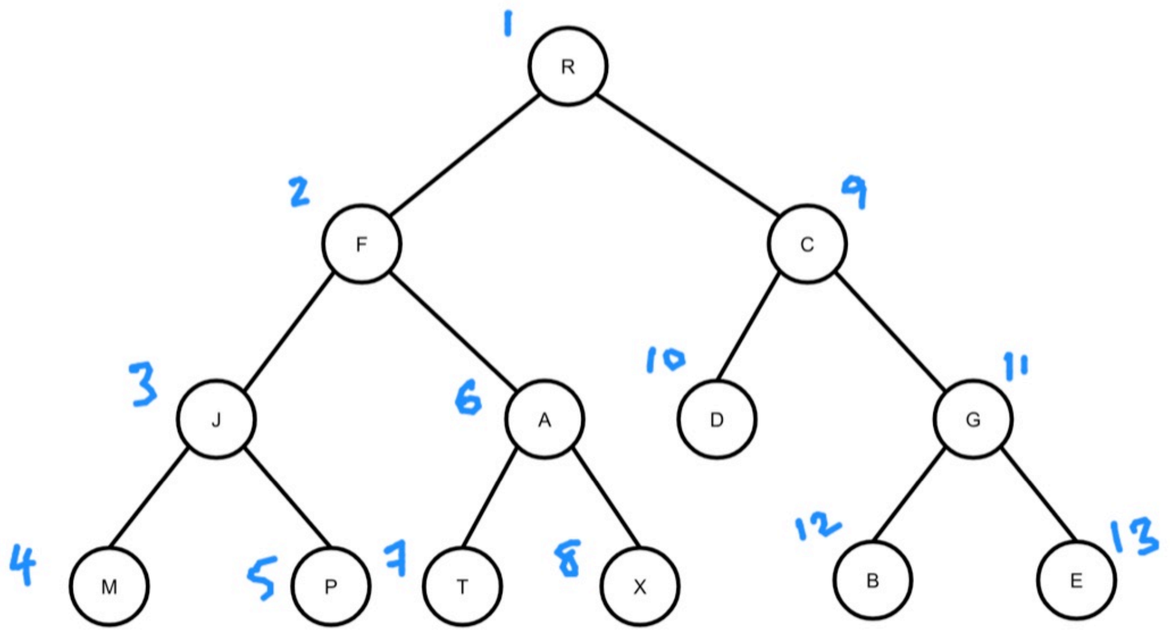
- **Pre-order:** process current node, then left node (and children), then right node (and children).
- **In-order:** process left node (and children), then current node, then right node (and children).
- **Post-order:** process left node (and children), then right node (and children), then current node.
- **Level-order:** all nodes at level i are processed before moving to nodes at level $i + 1$.



Traversing trees: pre-order, in-order, post-order, level-order.

- **Pre-order:** process current node, then left node (and children), then right node (and children).

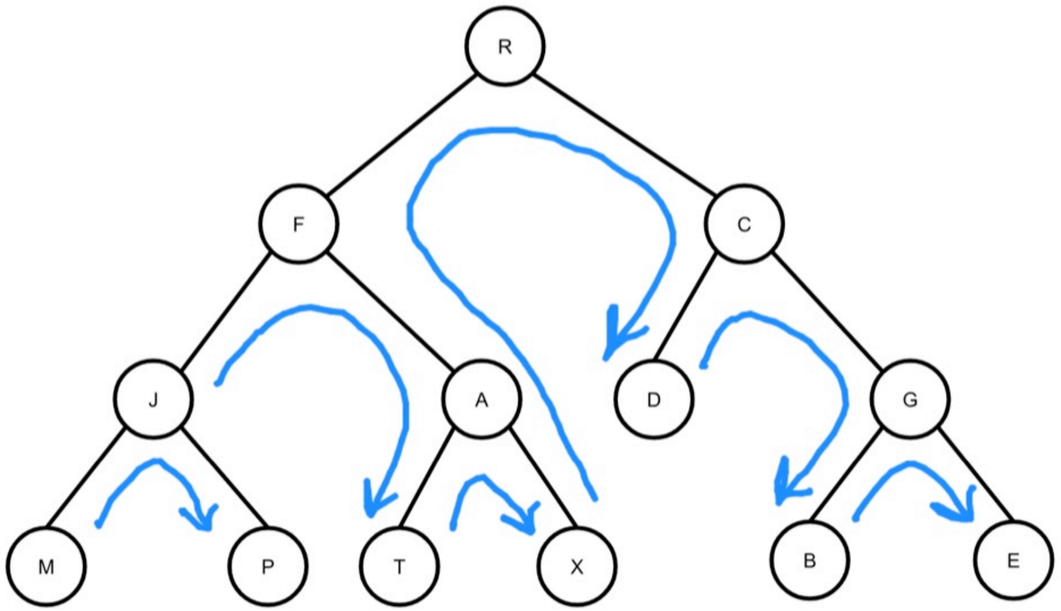
pre-order: R, F, J, M, P, A, T, X, C, D, G, B, E



Traversing trees: pre-order, in-order, post-order, level-order.

- **Pre-order:** process current node, then left node (and children), then right node (and children).
- **In-order:** process left node (and children), then current node, then right node (and children).

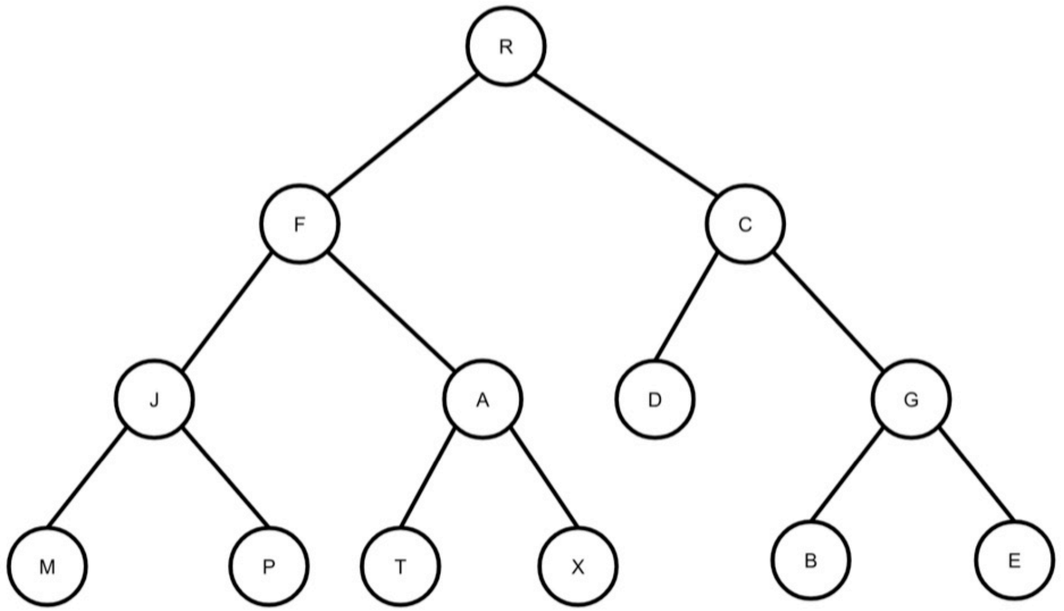
inorder: M, J, P, F, T, A, X, R, D, C, B, G, E



Traversing trees: pre-order, in-order, post-order, level-order.

- **Pre-order:** process current node, then left node (and children), then right node (and children).
- **In-order:** process left node (and children), then current node, then right node (and children).
- **Post-order:** process left node (and children), then right node (and children), then current node.

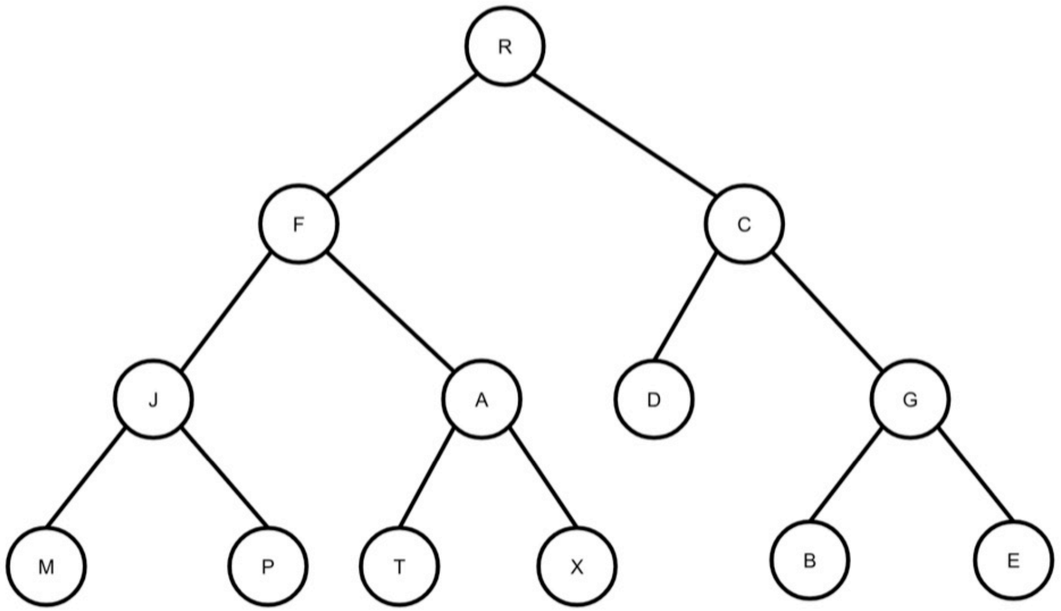
Post order : M, P, J, T, X, A, F, D, B, E, G, C, R



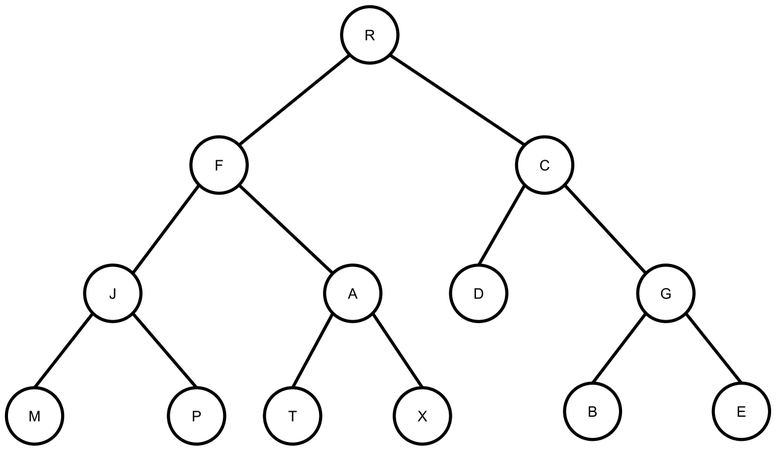
Traversing trees: pre-order, in-order, post-order, level-order.

- **Pre-order:** process current node, then left node (and children), then right node (and children).
- **In-order:** process left node (and children), then current node, then right node (and children).
- **Post-order:** process left node (and children), then right node (and children), then current node.
- **Level-order:** all nodes at level i are processed before moving to nodes at level $i + 1$.

Level order : R, F, C, J, A, D, G, M, P, T, X, B, E

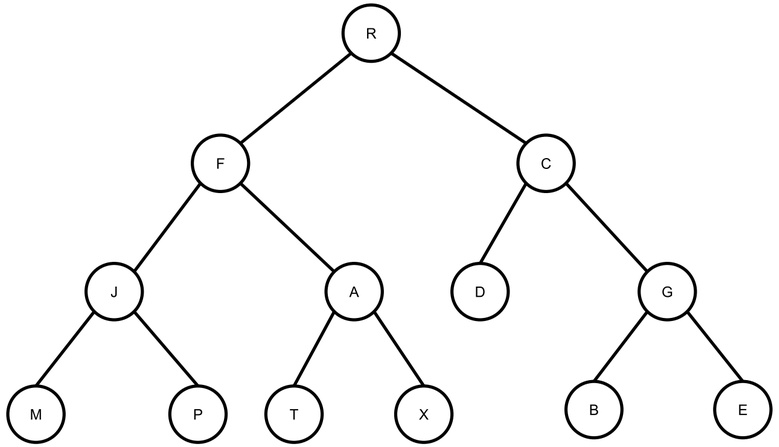


Exercise: works in pairs and complete the `height` method to compute the height of a tree.




```java
public int height() {
  return height(root);
}

private int height(TreeNode<E> node) {
  if (node == null) return -1;
  int leftHeight = height(node.left);
  int rightHeight = height(node.right);
  return 1 + Math.max(leftHeight, rightHeight);
}

```

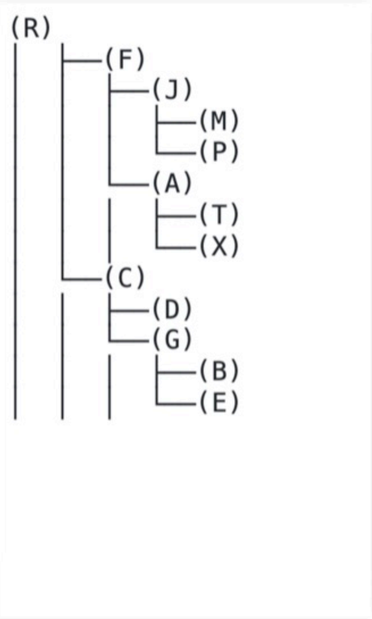
Exercise: works in pairs and write code to verify the pre-order, in-order and post-order traversal results we had earlier.




```java
public String visitPreOrder() {
  return visitPreOrder(root);
}

private String visitPreOrder(TreeNode<E> node) {
  String result = "";
  if (node == null) return result;
  result += node.get() + " ";
  result += visitPreOrder(node.left);
  result += visitPreOrder(node.right);
  return result;
}

```

Printing the tree using pre-order traversal.

```java
private String toStringHelper(String padding, TreeNode<E> node) {
  if (node == null) {
    return "";
  }

  String result = padding + "└──(" + node.toString() + ")"  + "\n";

  padding += "│    ";
  result += toStringHelper(padding, node.left);
  result += toStringHelper(padding, node.right);

  return result;
}
```



insipired by: https://www.baeldung.com/java-print-binary-tree-diagram

Reminders:

- Homework 6 due Thursday 4/10: implement a calculator (using a stack) & mid-semester check-in.
- **Next class:** variant of queues where elements have a *priority*.
- No lab meetings on Friday 4/11 -- attend the Spring Student Symposium.