



Middlebury

CSCI 201: Data Structures

Spring 2025

Lecture 8M: Trees

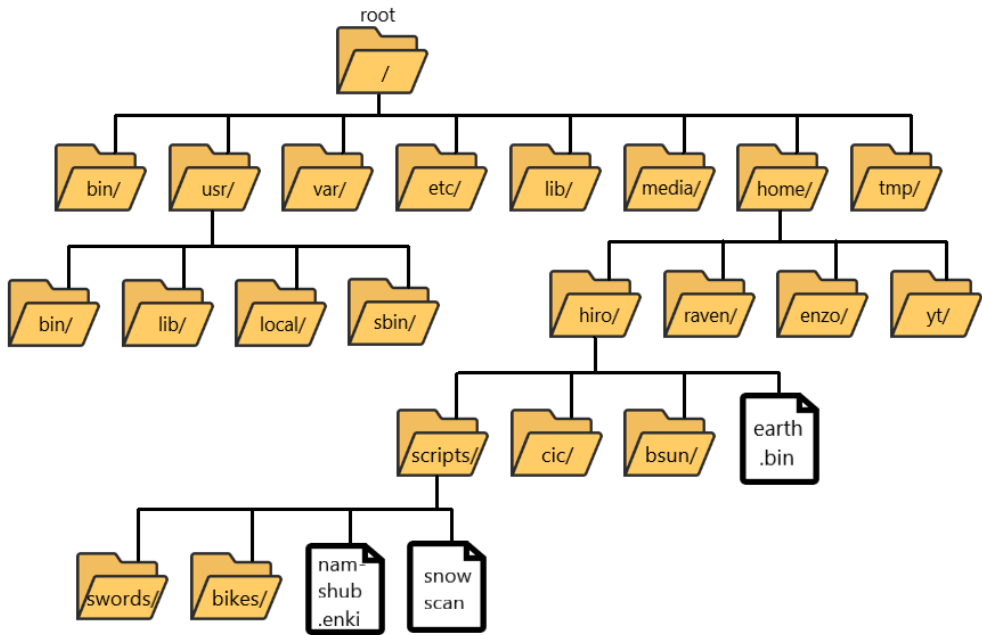
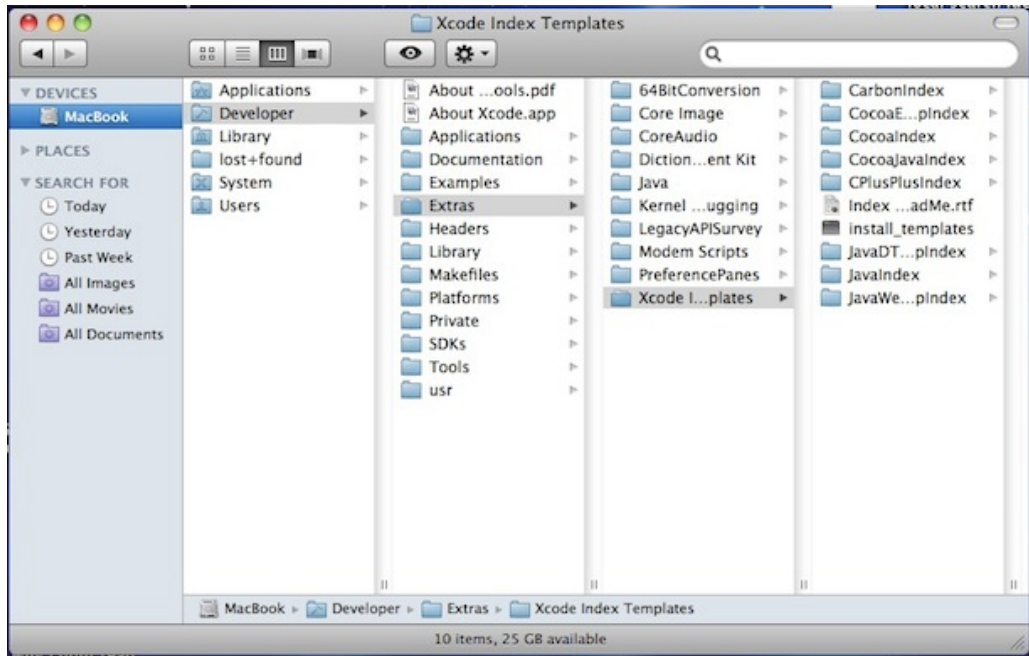
Goals for today:

- 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.



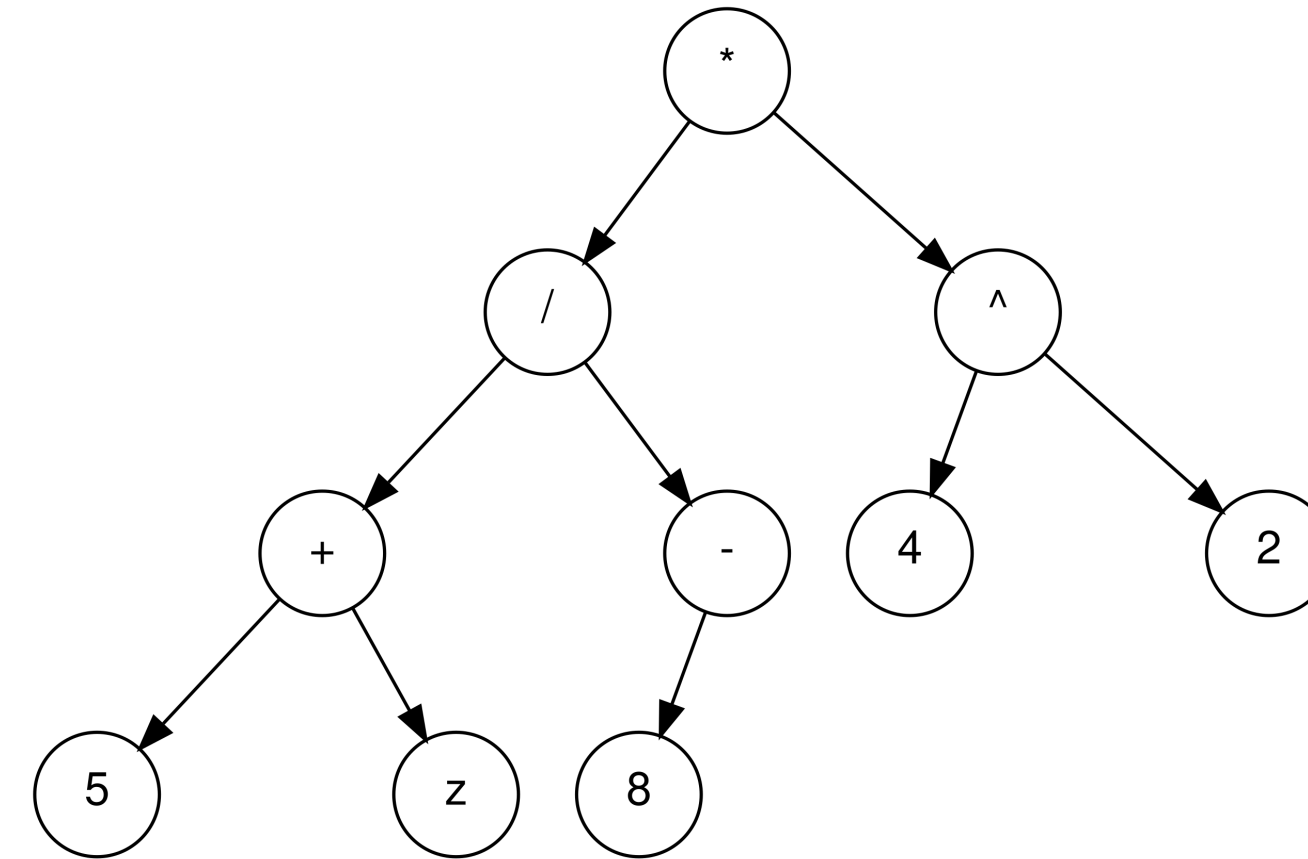
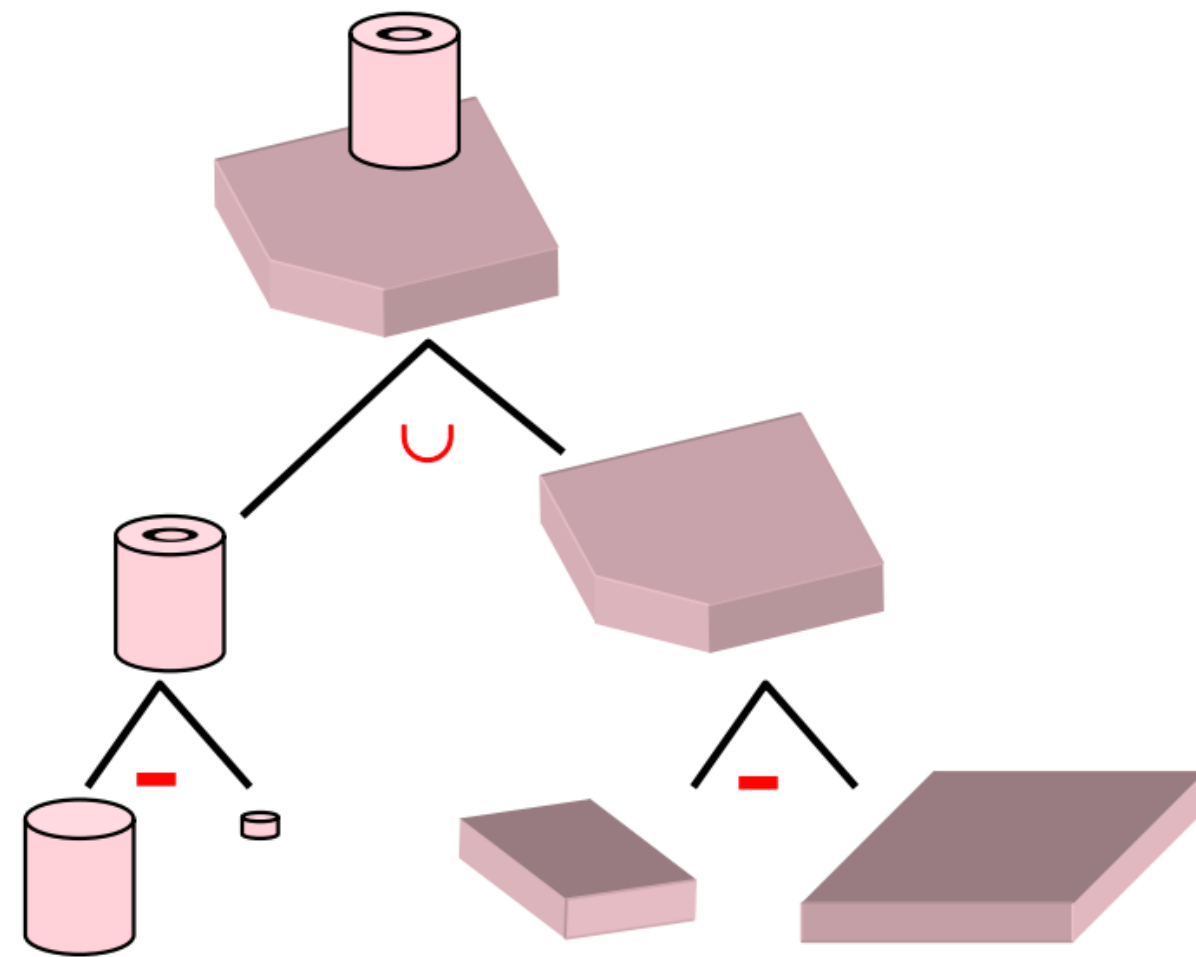
Trees are useful to represent hierarchy in data.

Contents		
1 INTRODUCTION		
Learning Objectives	1	
1.1 Some Characteristics of Fluids	3	
1.2 Dimensions, Dimensional Homogeneity, and Units	4	
1.2.1 Systems of Units	7	
1.3 Analysis of Fluid Behavior	11	
1.4 Measures of Fluid Mass and Weight	11	
1.4.1 Density	11	
1.4.2 Specific Weight	12	
1.4.3 Specific Gravity	12	
1.5 Ideal Gas Law	12	
1.6 Viscosity	14	
1.7 Compressibility of Fluids	20	
1.7.1 Bulk Modulus	20	
1.7.2 Compression and Expansion of Gases	21	
1.7.3 Speed of Sound	22	
1.8 Vapor Pressure	23	
1.9 Surface Tension	24	
1.10 A Brief Look Back in History	27	
1.11 Chapter Summary and Study Guide	29	
References	30	
Review Problems	30	
Problems	31	
2 FLUID STATICS		
Learning Objectives	38	
2.1 Pressure at a Point	38	
2.2 Basic Equation for Pressure Field	40	
2.3 Pressure Variation in a Fluid at Rest	41	
2.3.1 Incompressible Fluid	42	
2.3.2 Compressible Fluid	45	
2.4 Standard Atmosphere	47	
2.5 Measurement of Pressure	48	
2.6 Manometry	50	
2.6.1 Piezometer Tube	50	
2.6.2 U-Tube Manometer	51	
2.6.3 Inclined-Tube Manometer	54	
2.7 Mechanical and Electronic Pressure Measuring Devices	55	
2.8 Hydrostatic Force on a Plane Surface	57	
2.9 Pressure Prism	63	
2.10 Hydrostatic Force on a Curved Surface	66	
2.11 Buoyancy, Flotation, and Stability	68	
2.11.1 Archimedes' Principle	68	
2.11.2 Stability	71	
2.12 Pressure Variation in a Fluid with Rigid-Body Motion	72	
2.12.1 Linear Motion	73	
2.12.2 Rigid-Body Rotation	75	
2.13 Chapter Summary and Study Guide	77	
References	78	
Review Problems	78	
Problems	78	
3 ELEMENTARY FLUID DYNAMICS—THE BERNOULLI EQUATION		
Learning Objectives	93	
3.1 Newton's Second Law	94	
3.2 $\mathbf{F} = m\mathbf{a}$ along a Streamline	96	
3.3 $\mathbf{F} = m\mathbf{a}$ Normal to a Streamline	100	
3.4 Physical Interpretation	102	
3.5 Static, Stagnation, Dynamic, and Total Pressure	105	
3.6 Examples of Use of the Bernoulli Equation	110	
3.6.1 Free Jets	110	
3.6.2 Confined Flows	112	
3.6.3 Flowrate Measurement	118	
3.7 The Energy Line and the Hydraulic Grade Line	123	
3.8 Restrictions on Use of the Bernoulli Equation	126	
3.8.1 Compressibility Effects	126	
3.8.2 Unsteady Effects	128	
3.8.3 Rotational Effects	130	
3.8.4 Other Restrictions	131	
3.9 Chapter Summary and Study Guide	131	
References	133	
Review Problems	133	
Problems	133	



Trees are also useful for representing expressions.

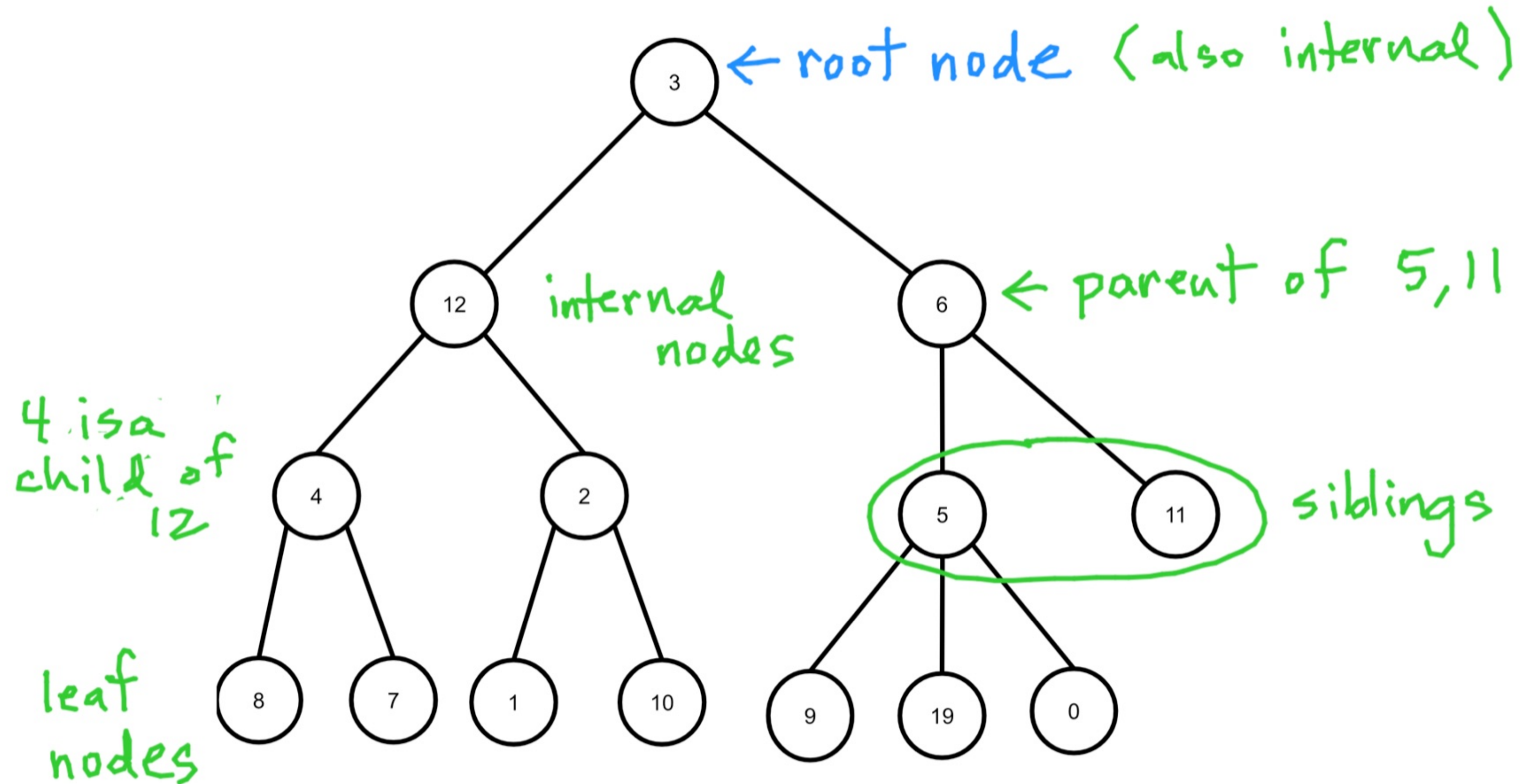
https://en.wikipedia.org/wiki/Binary_expression_tree



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

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

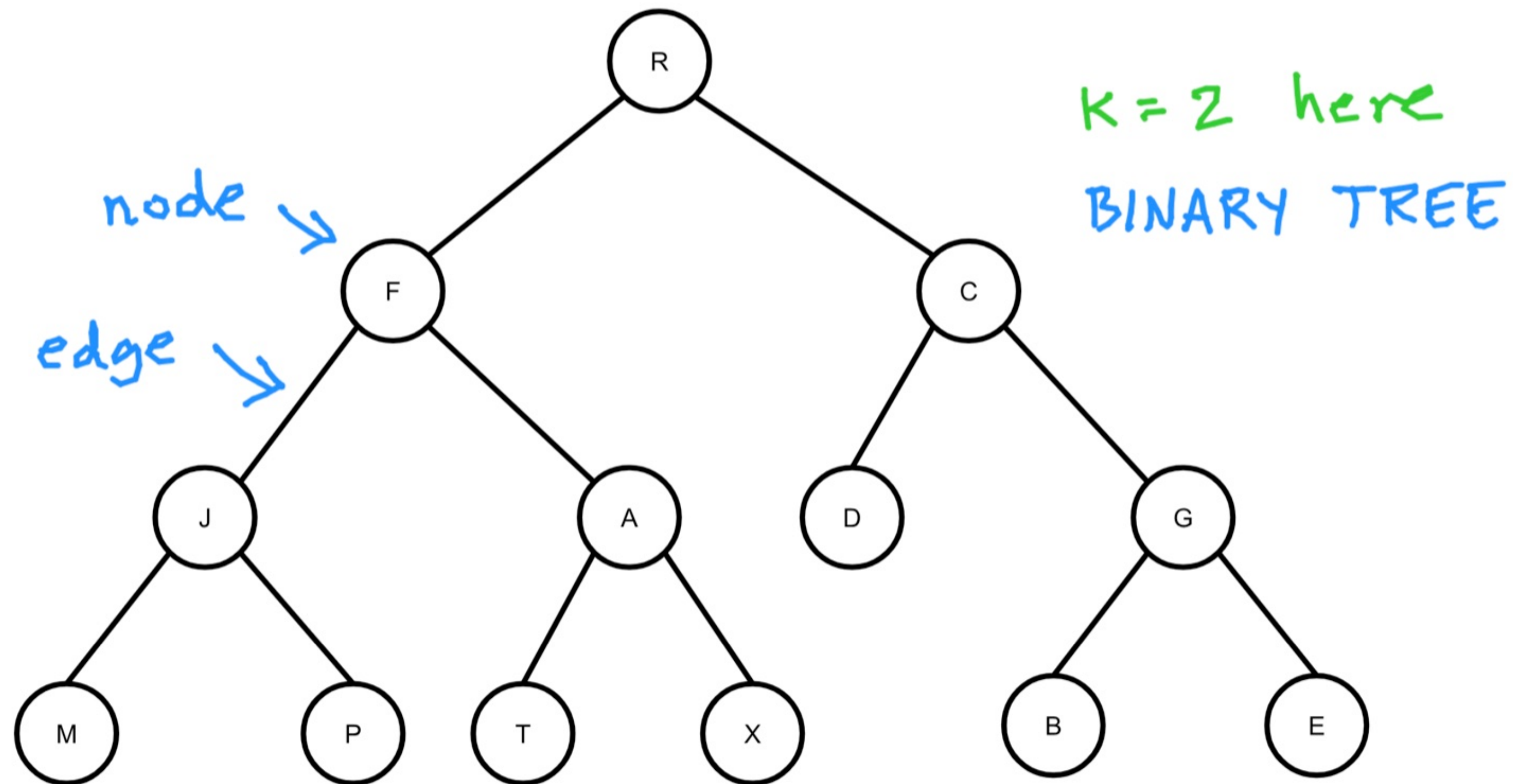
A **rooted tree** is a set of nodes based on a parent-child relationship.



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

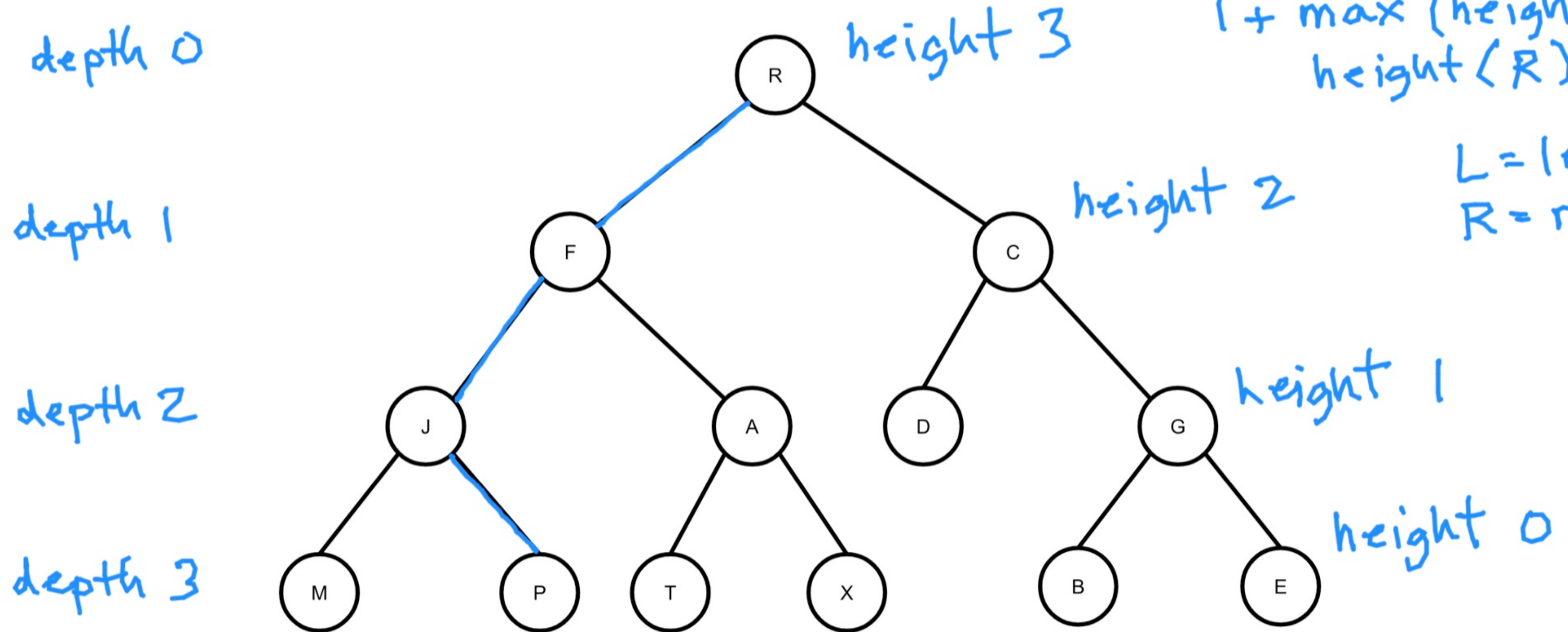


Paths, level, depth and height of a node.

- 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.

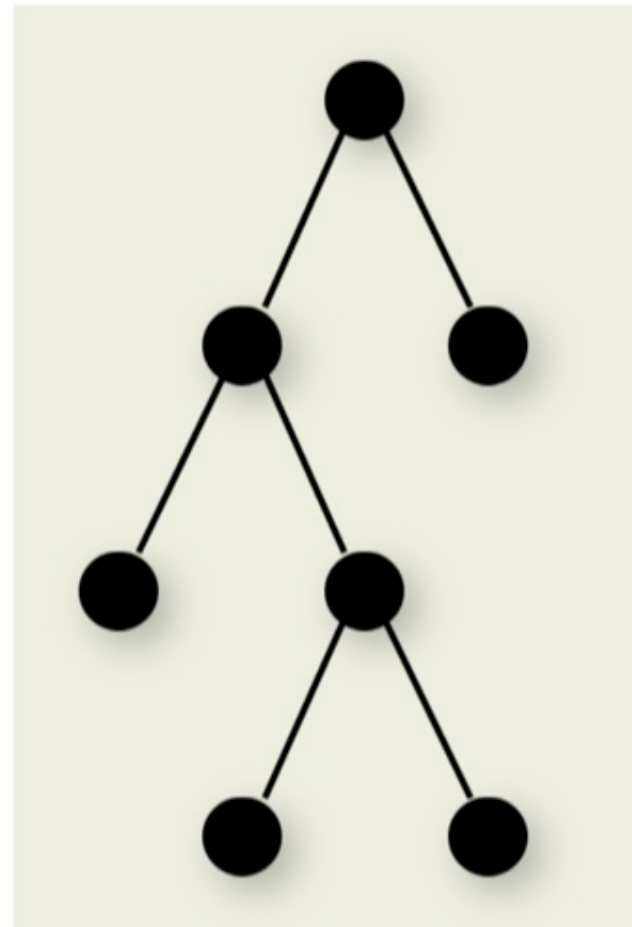
$\text{height}(\text{null}) = -1$
 $\text{height}(\text{leaf}) = 0$
 $\text{height}(\text{node}) :$
 $1 + \max(\text{height}(L), \text{height}(R))$

$L = \text{left}$
 $R = \text{right}$

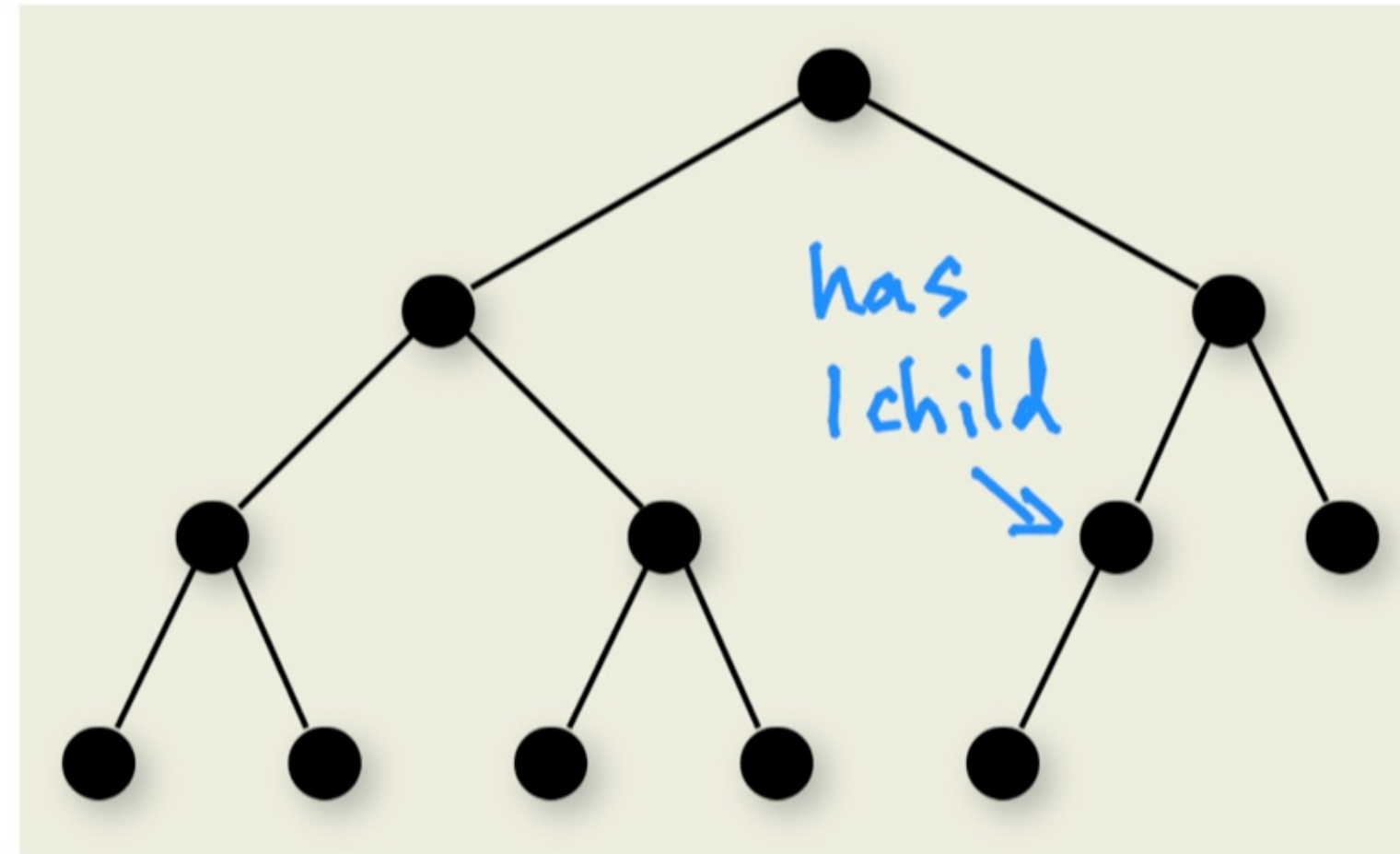


Properties of binary trees.

- **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



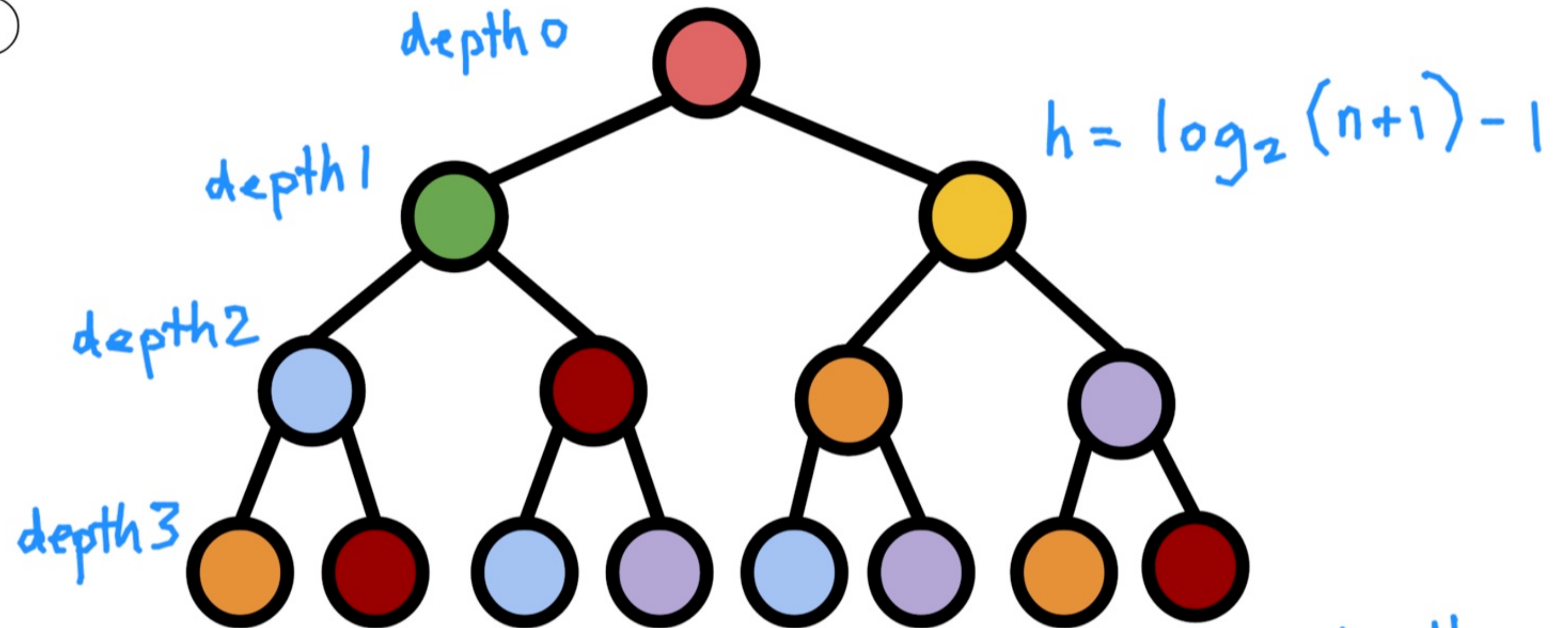
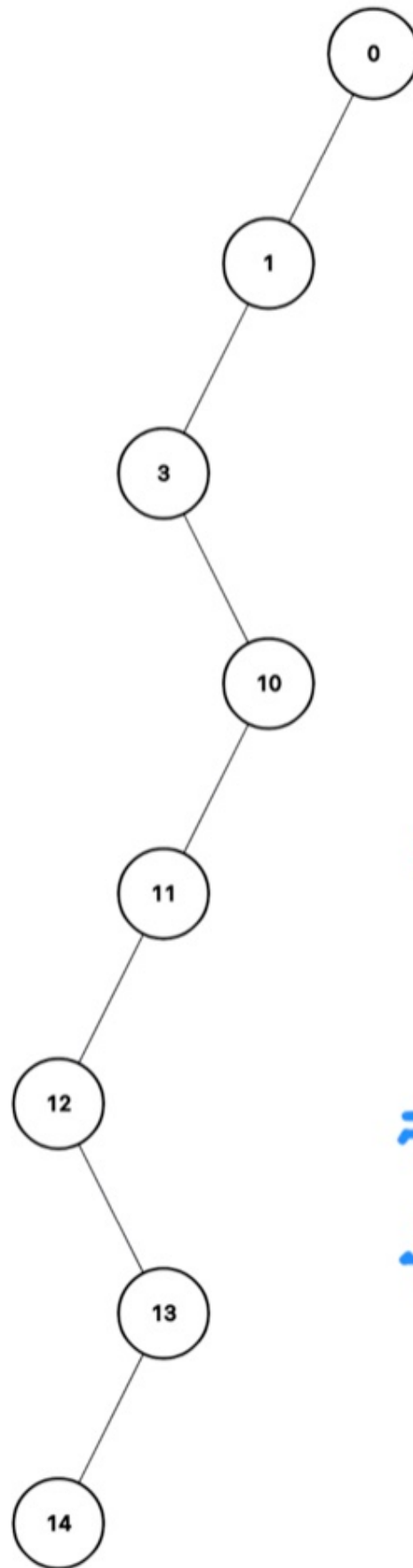
not full
complete

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

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 =
 $n-1$



nodes at a particular level/depth: 2^{depth}
total # nodes: $2^0 + 2^1 + 2^2 + \dots + 2^h$ (height h)

geometric series $r=2$

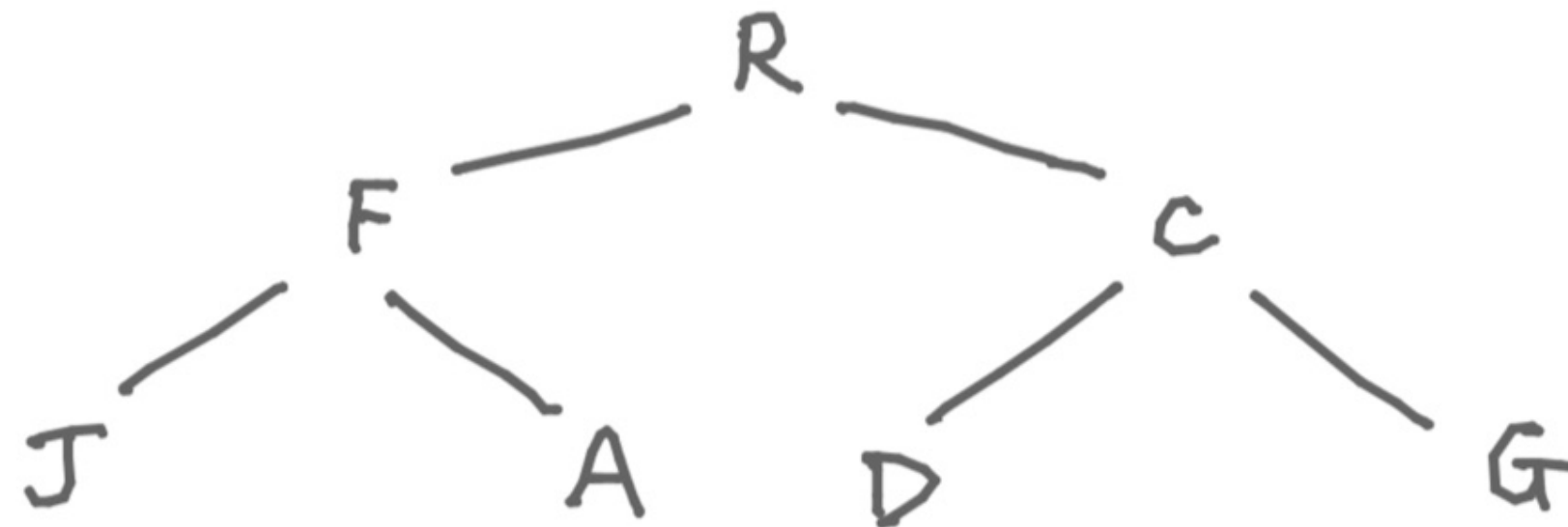
$$\frac{2^{h+1} - 1}{2 - 1} = 2^{h+1} - 1 = n \rightarrow h+1 = \log_2(n+1)$$

Representing trees with a computer.

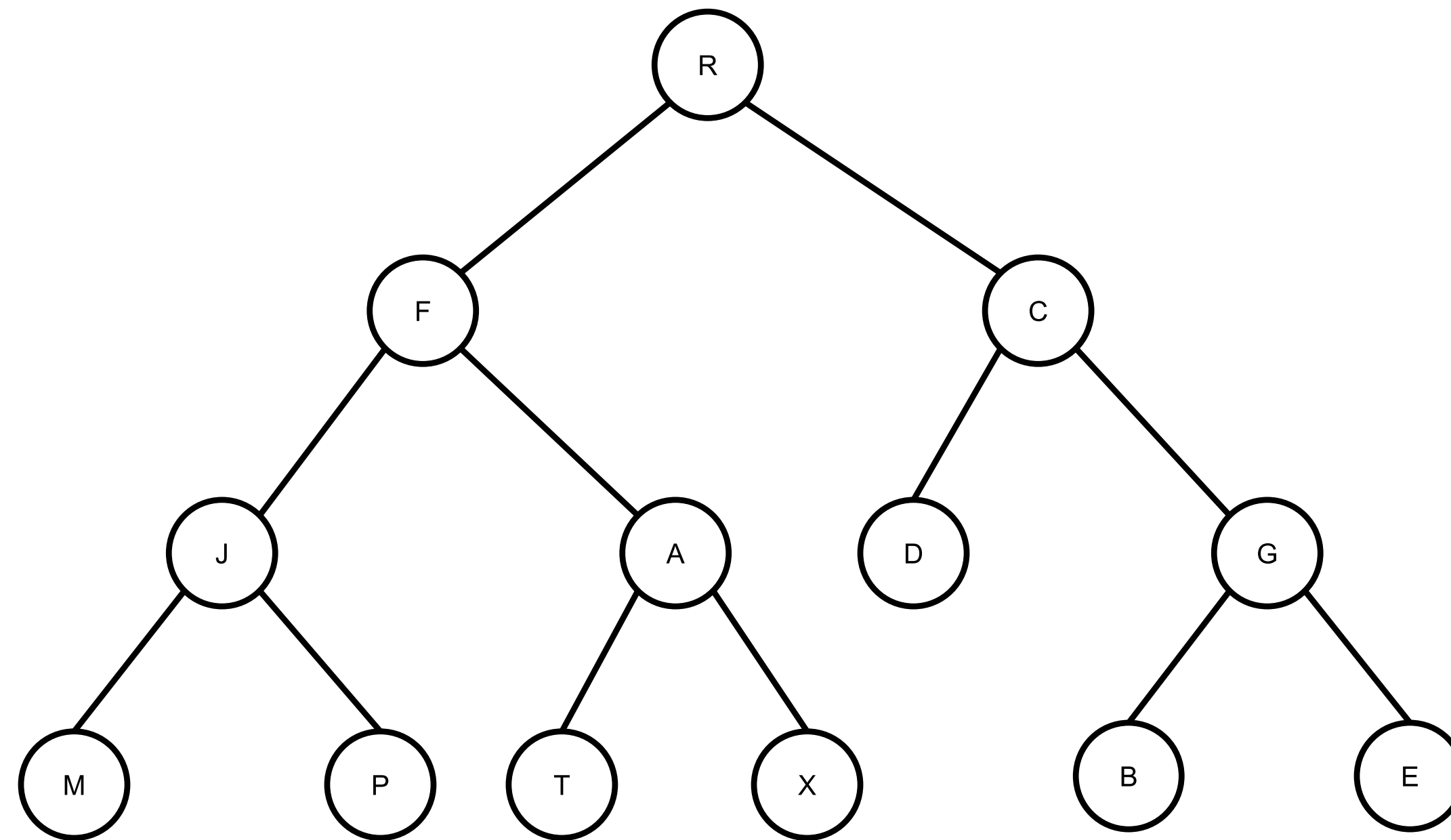
```
1 class TreeNode<E> {  
2     private E data;  
3     public TreeNode<E> left;  
4     public TreeNode<E> right;  
5  
6     public TreeNode(E data) {  
7         this.data = data;  
8         left = null;  
9         right = null;  
10    }  
11  
12    public E get() {  
13        return data;  
14    }  
15 }
```

```
1 public class BinaryTree<E> {  
2     public TreeNode root;  
3  
4     public BinaryTree() {  
5         root = null;  
6     }  
7  
8     public static void main(String[] args) {  
9         // For now, we'll build the tree explicitly.  
10        BinaryTree<String> tree = new BinaryTree<>();  
11  
12        tree.root = new TreeNode<String>("R");  
13        TreeNode<String> f = new TreeNode<>("F");  
14        TreeNode<String> c = new TreeNode<>("C");  
15        tree.root.left = f;  
16        tree.root.right = c;  
17        // ...  
18    }  
19 }
```

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

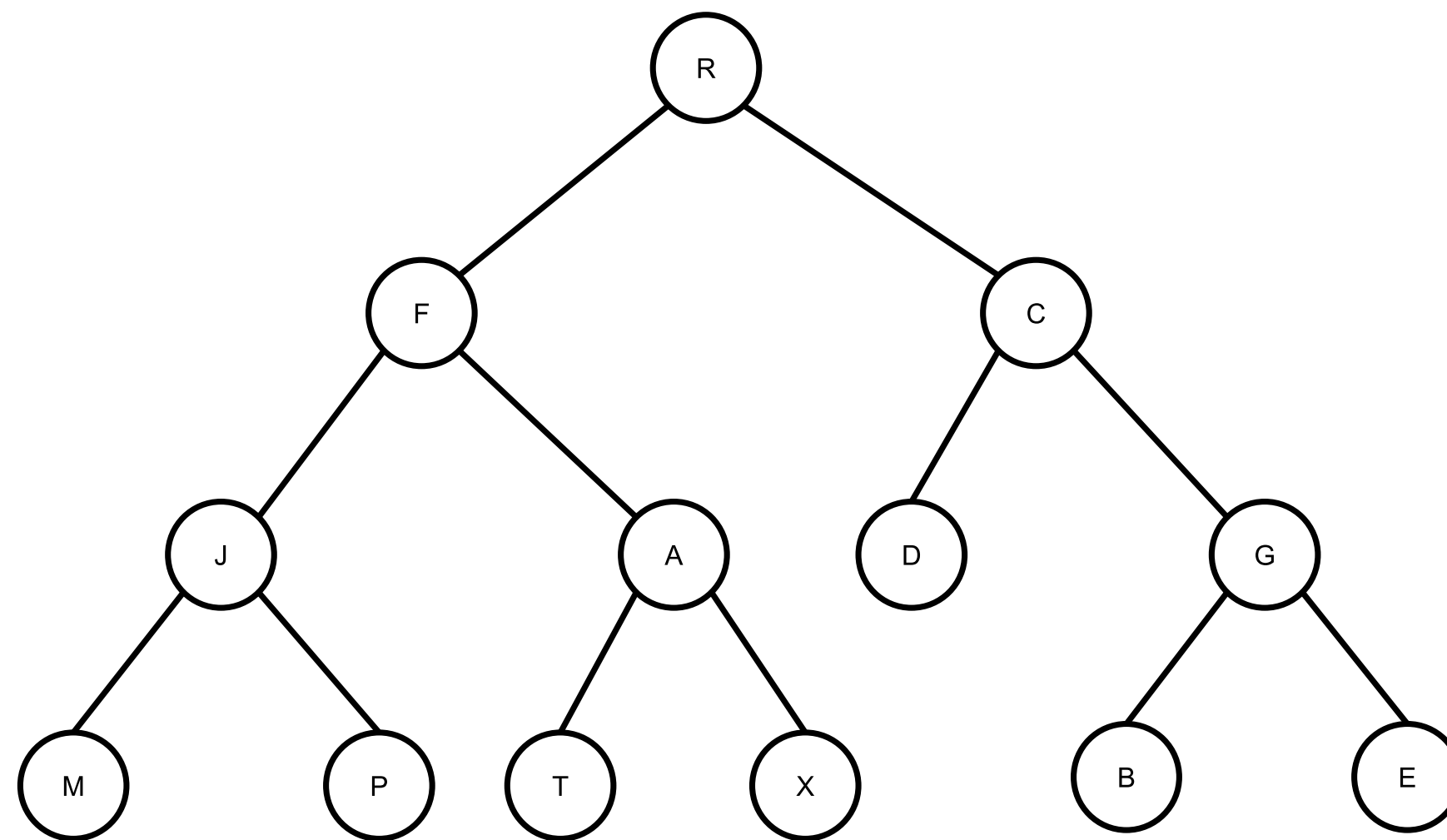


Exercise: finish manually building up this tree.



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

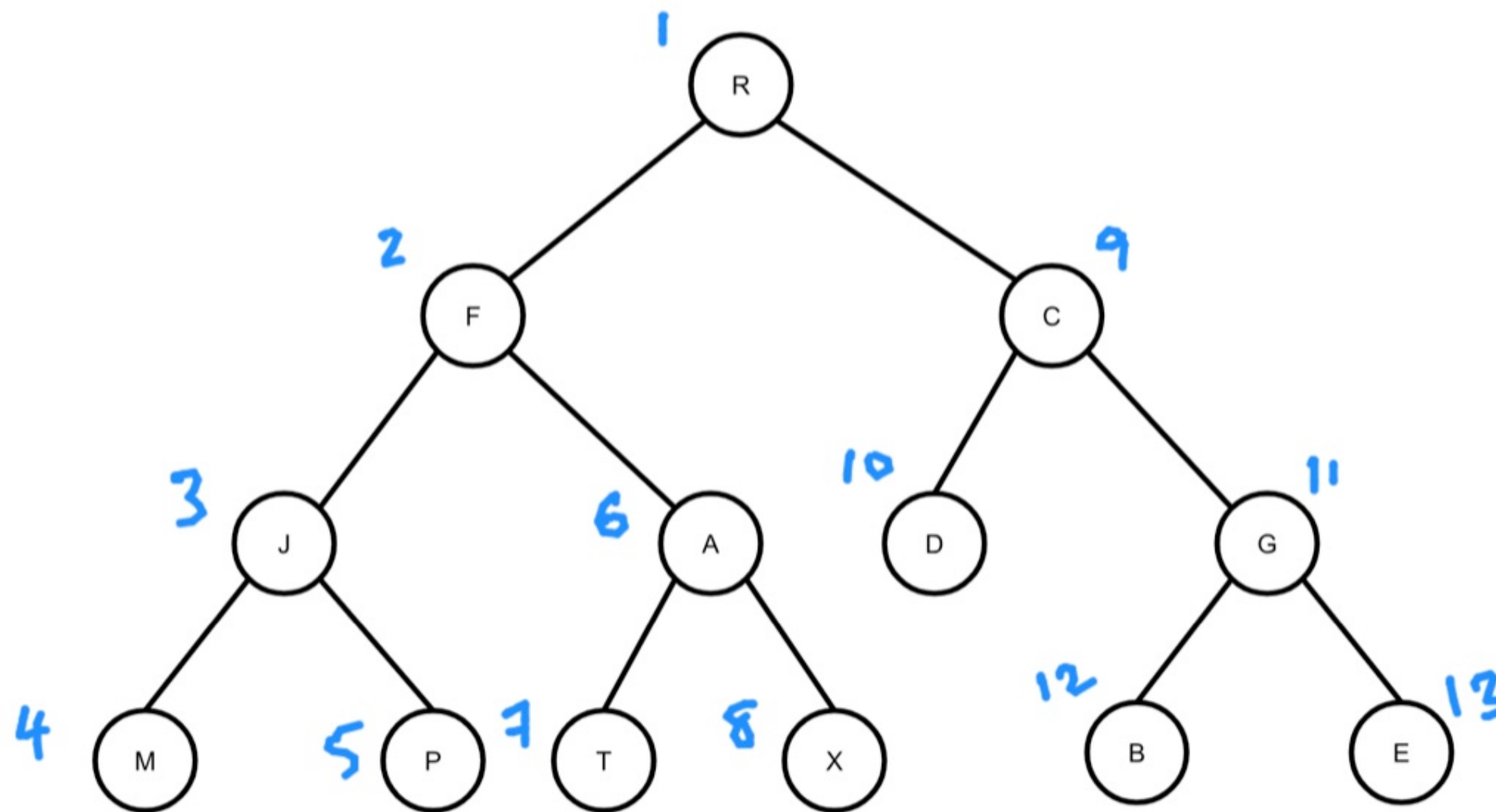
- **Pre-order:** process current node, then left node (and children), then right node (and children).
- **In-order:** process left node (and children), then current node, then right node (and children).
- **Post-order:** process left node (and children), then right node (and children), then current node.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

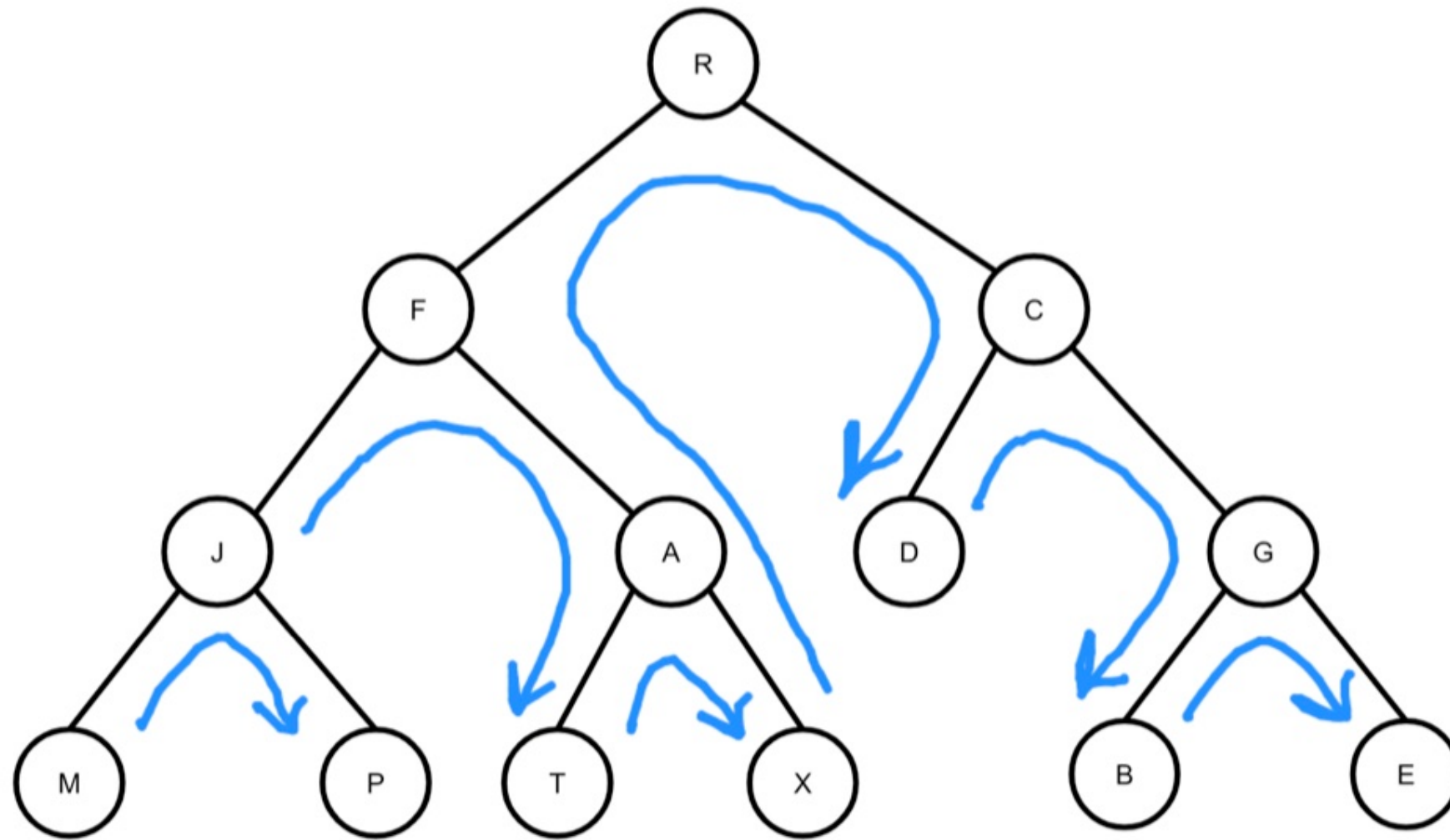
pre-order: R, F, J, M, P, A, T, X, C, D, G, B, E



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).

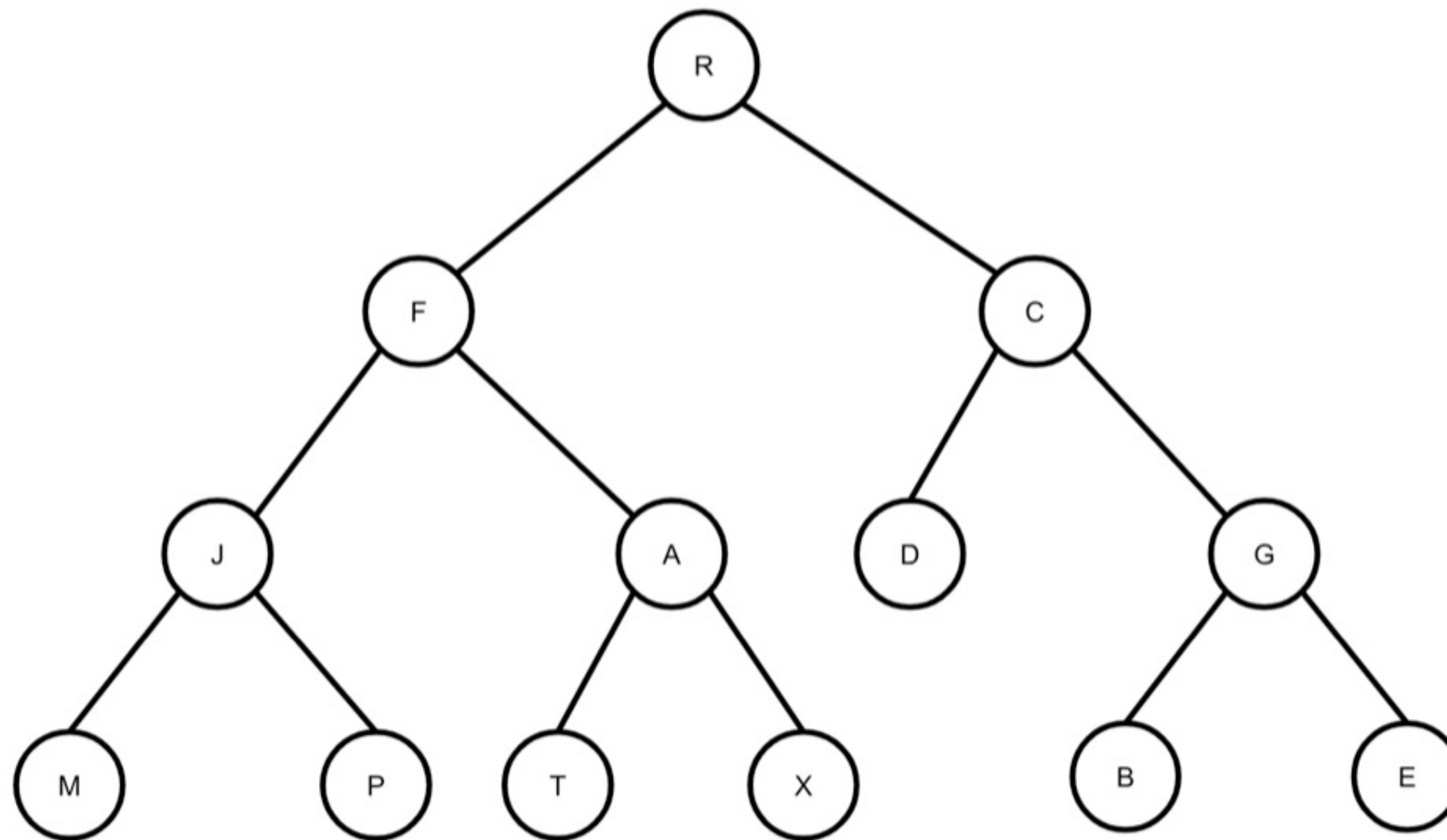
inorder: M, J, P, F, T, A, X, R, D, C, B, G, E



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.

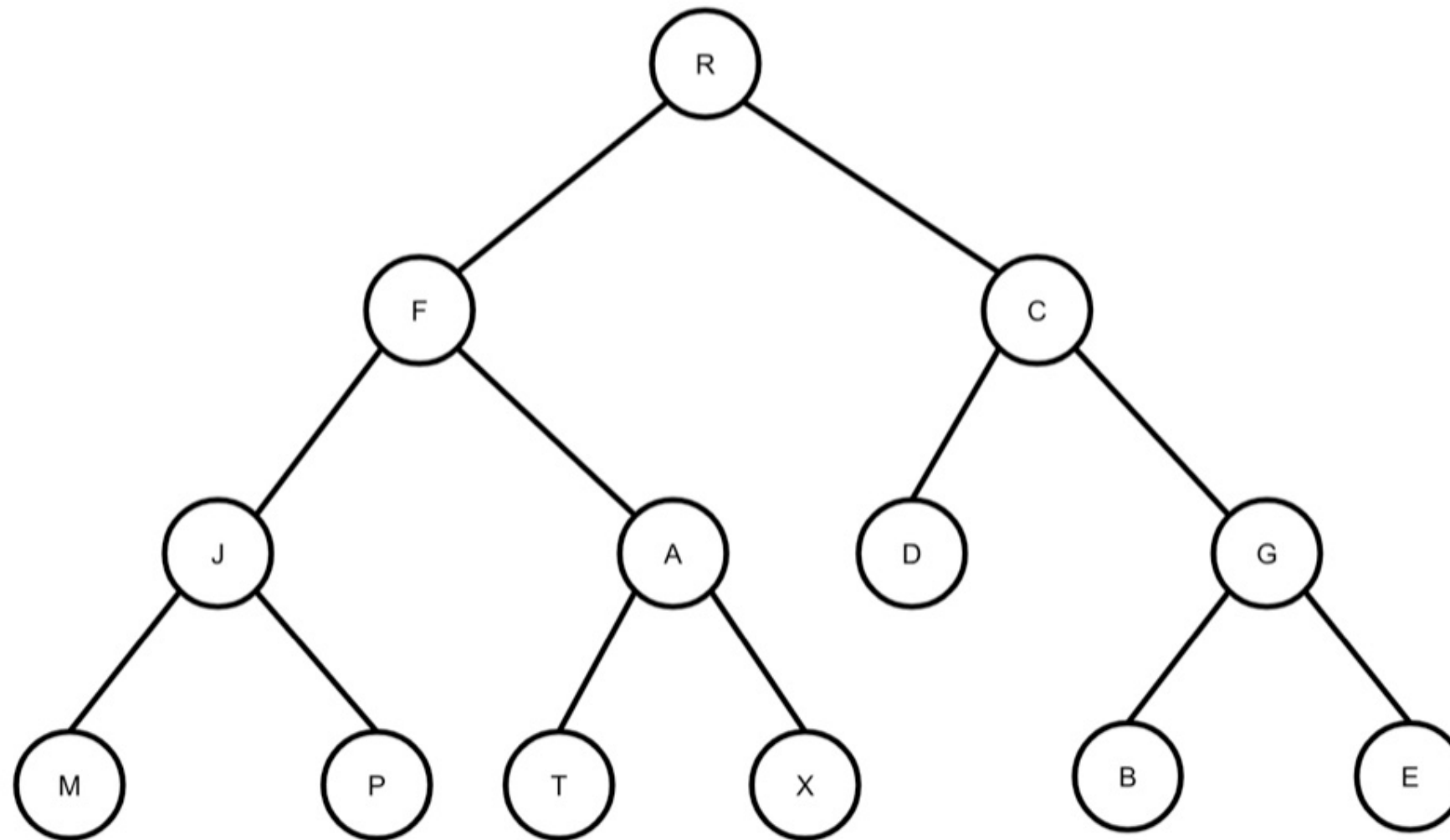
Post order : M, P, J, T, X, A, F, D, B, E, G, C, R



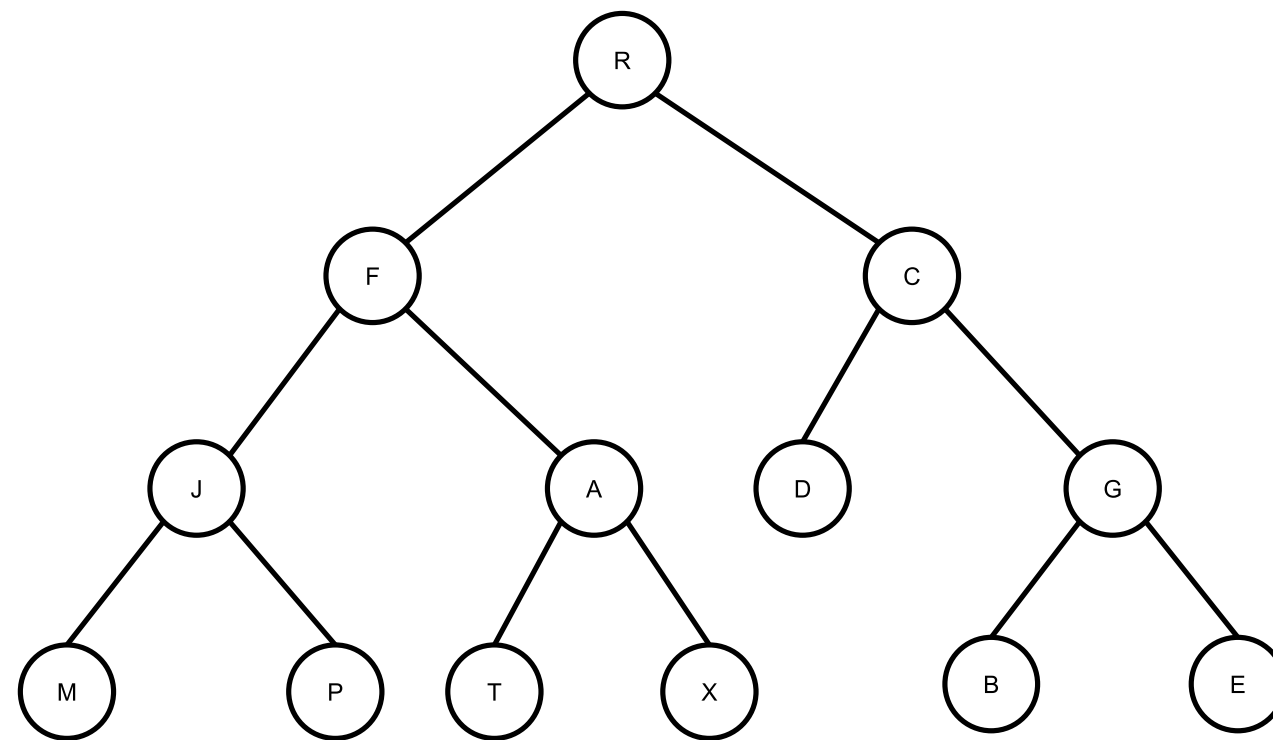
Traversing trees: pre-order, in-order, post-order, level-order.

- **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

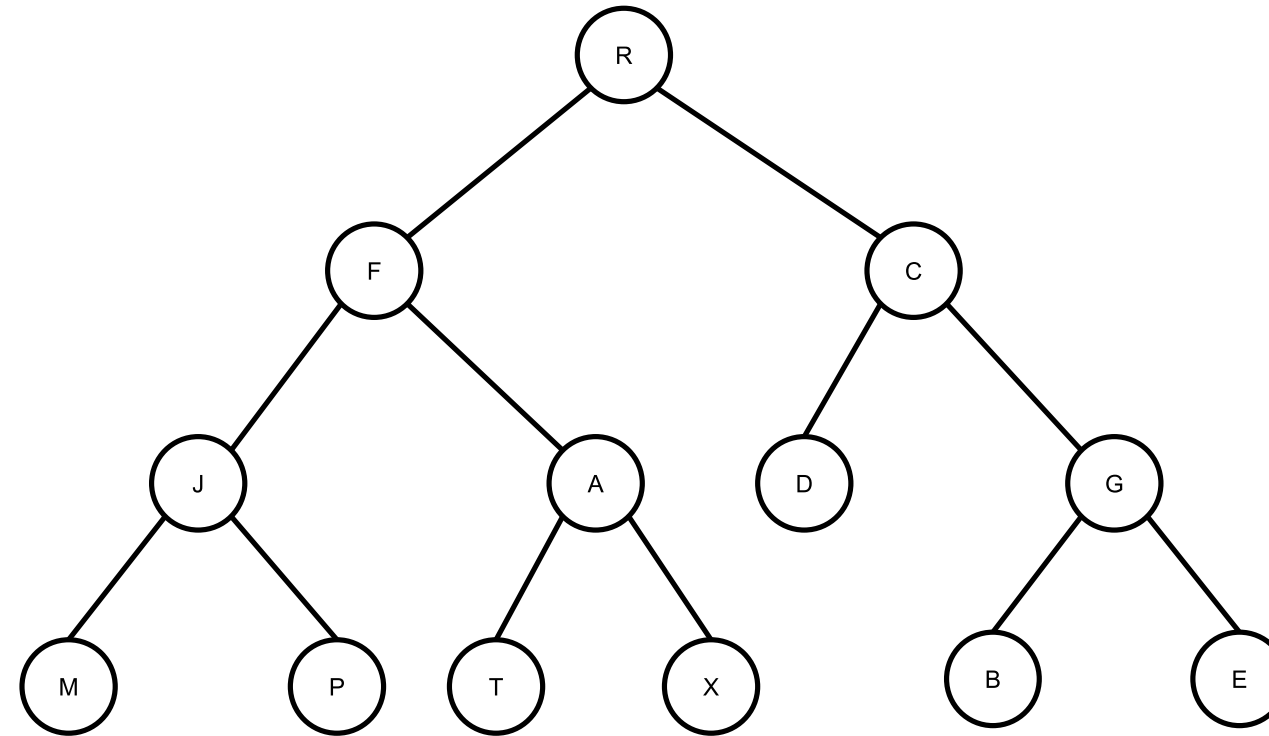


Exercise: works in pairs and complete the **height** method to compute the height of a tree.



```
1  public int height() {  
2      return height(root);  
3  }  
4  
5  private int height(TreeNode<E> node) {  
6      if (node == null) return -1;  
7      int leftHeight = height(node.left);  
8      int rightHeight = height(node.right);  
9      return 1 + Math.max(leftHeight, rightHeight);  
10 }  
11
```

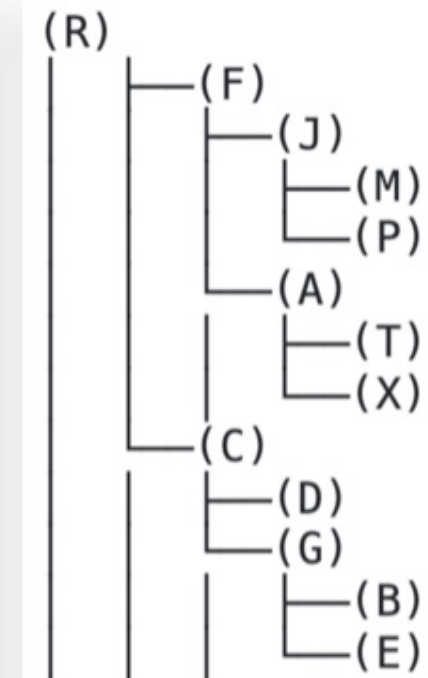
Exercise: works in pairs and write code to verify the pre-order, in-order and post-order traversal results we had earlier.



```
1  public String visitPreOrder() {
2      return visitPreOrder(root);
3  }
4
5  private String visitPreOrder(TreeNode<E> node) {
6      String result = "";
7      if (node == null) return result;
8      result += node.get() + " ";
9      result += visitPreOrder(node.left);
10     result += visitPreOrder(node.right);
11     return result;
12 }
13
```

Printing the tree using pre-order traversal.

```
1 private String toStringHelper(String padding, TreeNode<E> node) {  
2     if (node == null) {  
3         return "";  
4     }  
5  
6     String result = padding + "└─(" + node.toString() + ")" + "\n";  
7  
8     padding += "|   ";  
9     result += toStringHelper(padding, node.left);  
10    result += toStringHelper(padding, node.right);  
11  
12    return result;  
13 }
```



inspired 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](#).