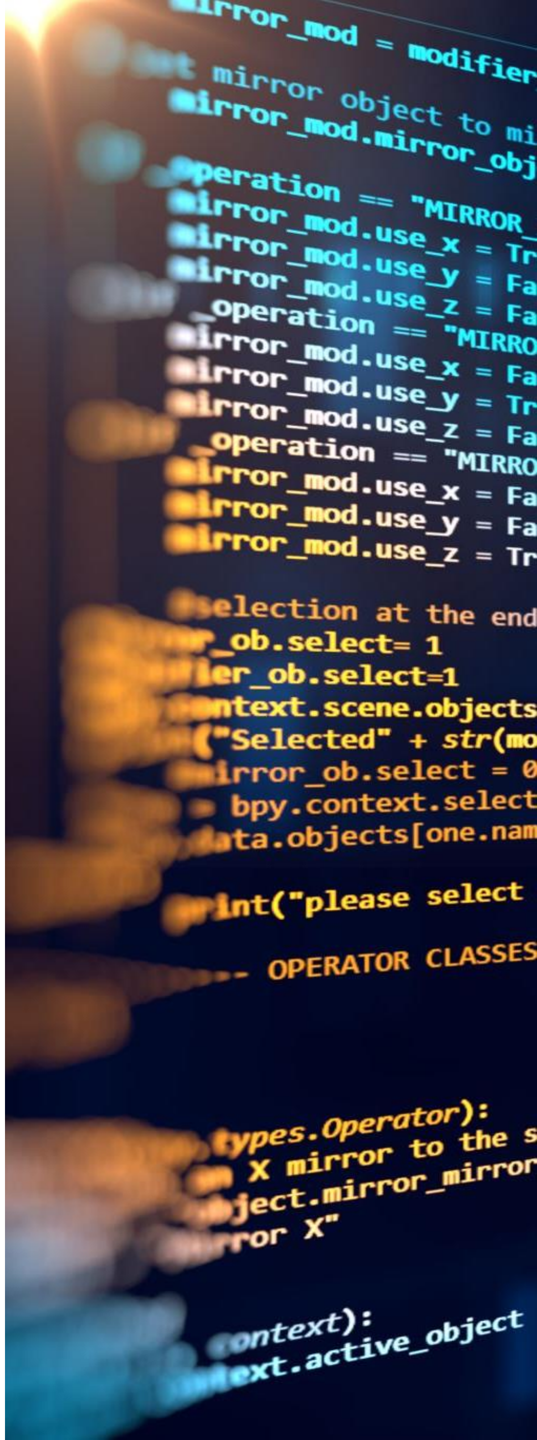
Please use the following QR code to check in and record your attendance.

CS 1027
Fundamentals of Computer Science II

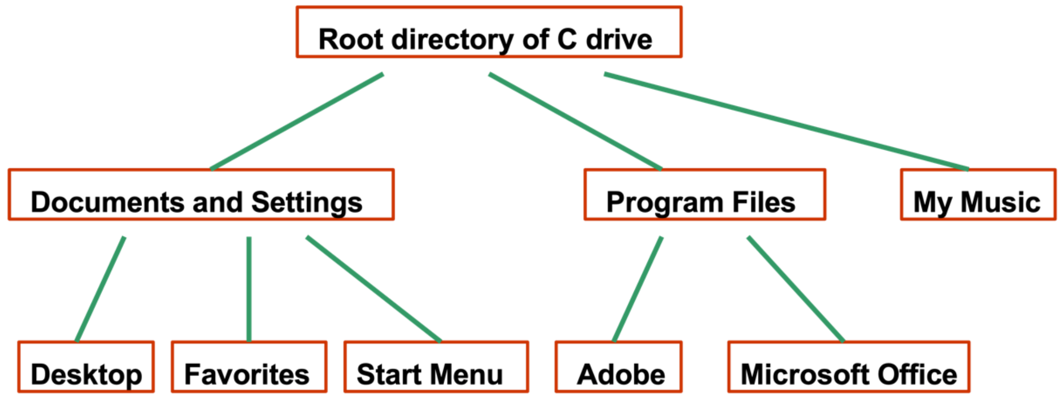
Trees ADT

Ahmed Ibrahim

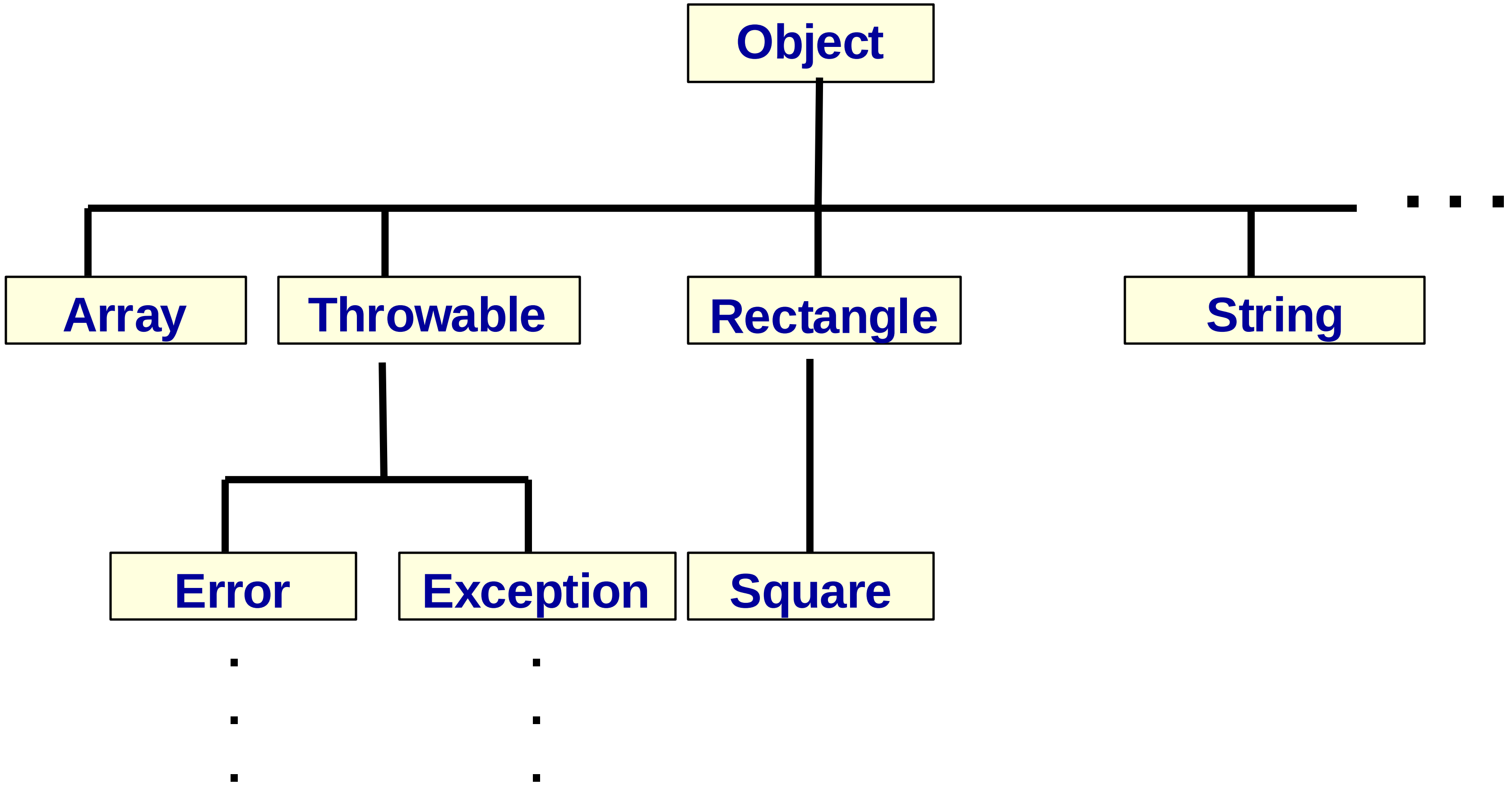


Trees

- A tree is a **non-linear abstract data type** that stores information in a hierarchy.
- Examples in real life:
  - Family tree
  - Table of contents of a book
  - Class Inheritance Hierarchy in Java
  - Computer file system (folders and subfolders)
  - Decision trees



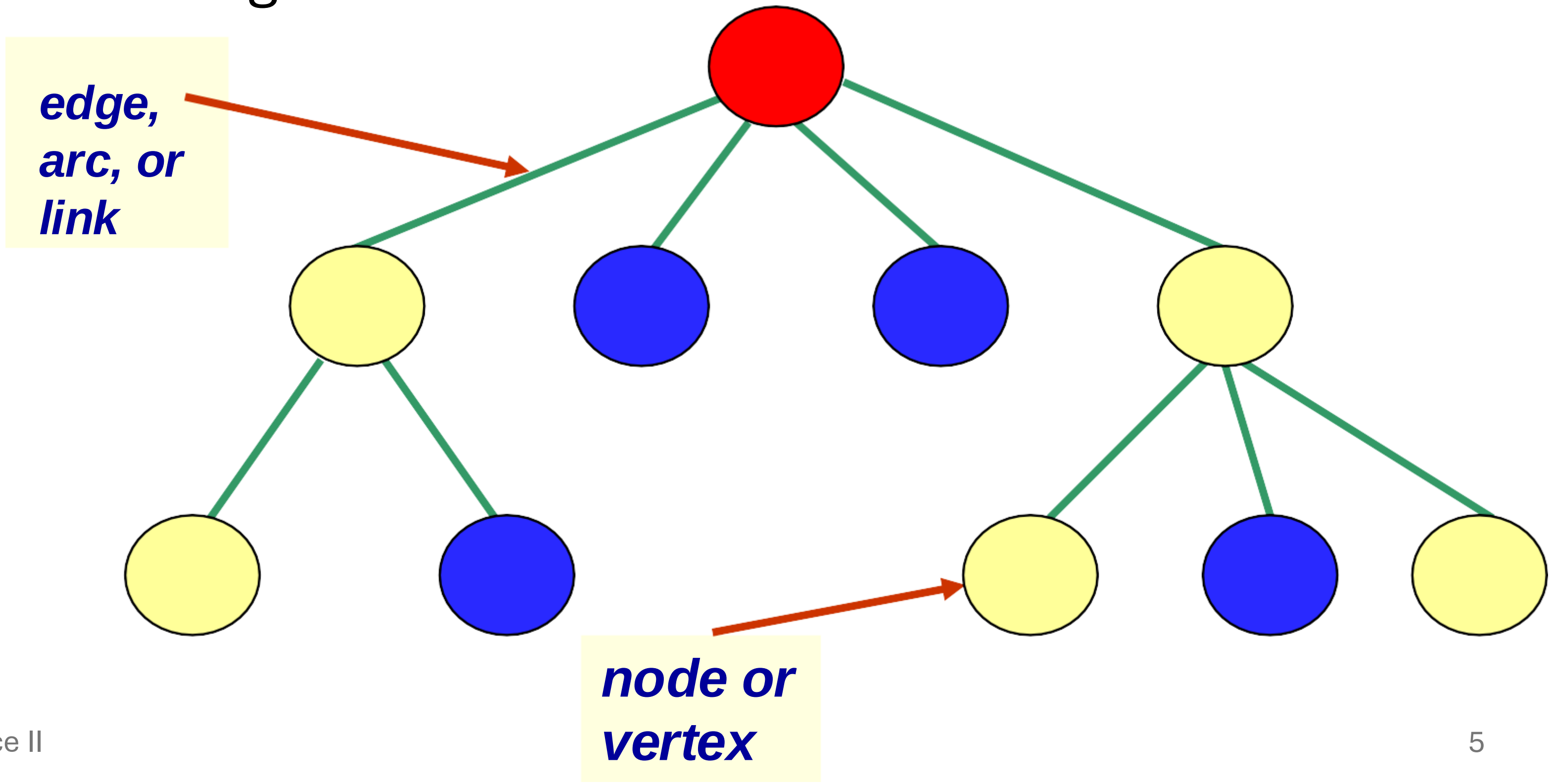
Example: Java's Class Hierarchy



Example: Java's Class Hierarchy

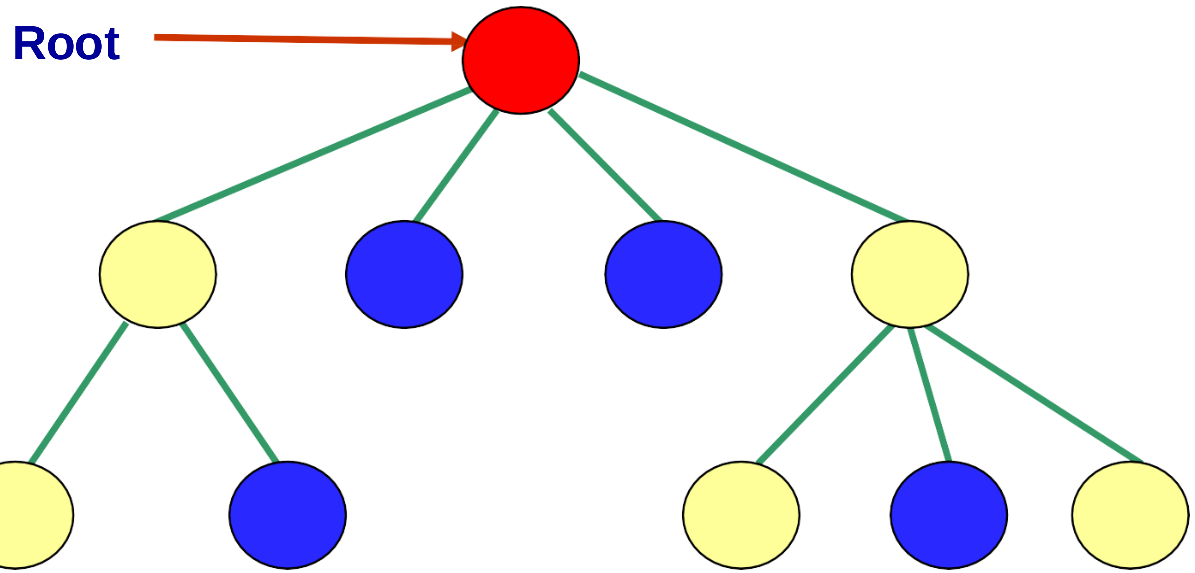
A tree consists of a set of

- **nodes** or vertices storing data and
- **edges**, links, or arcs connecting the nodes



Tree Definition

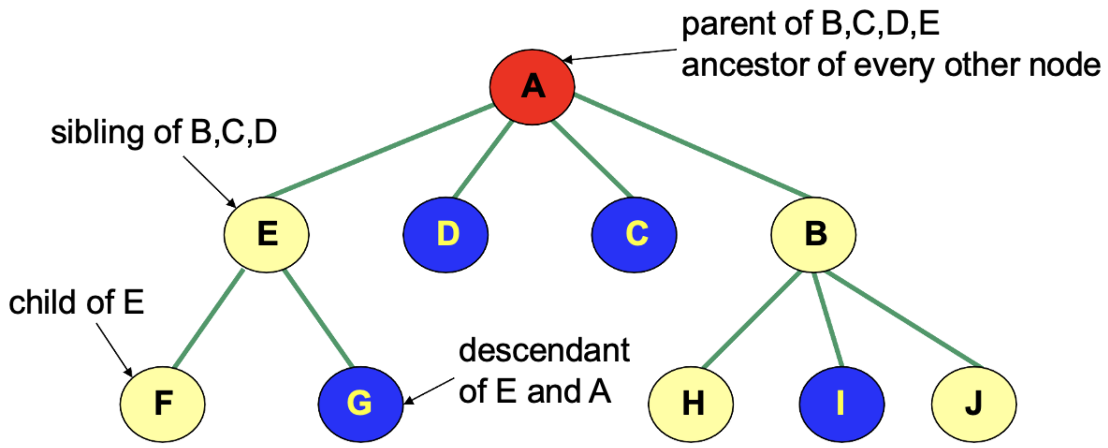
- There is a distinguished node called the root (usually drawn as the topmost node in the tree).



- An empty tree has no nodes or edges.

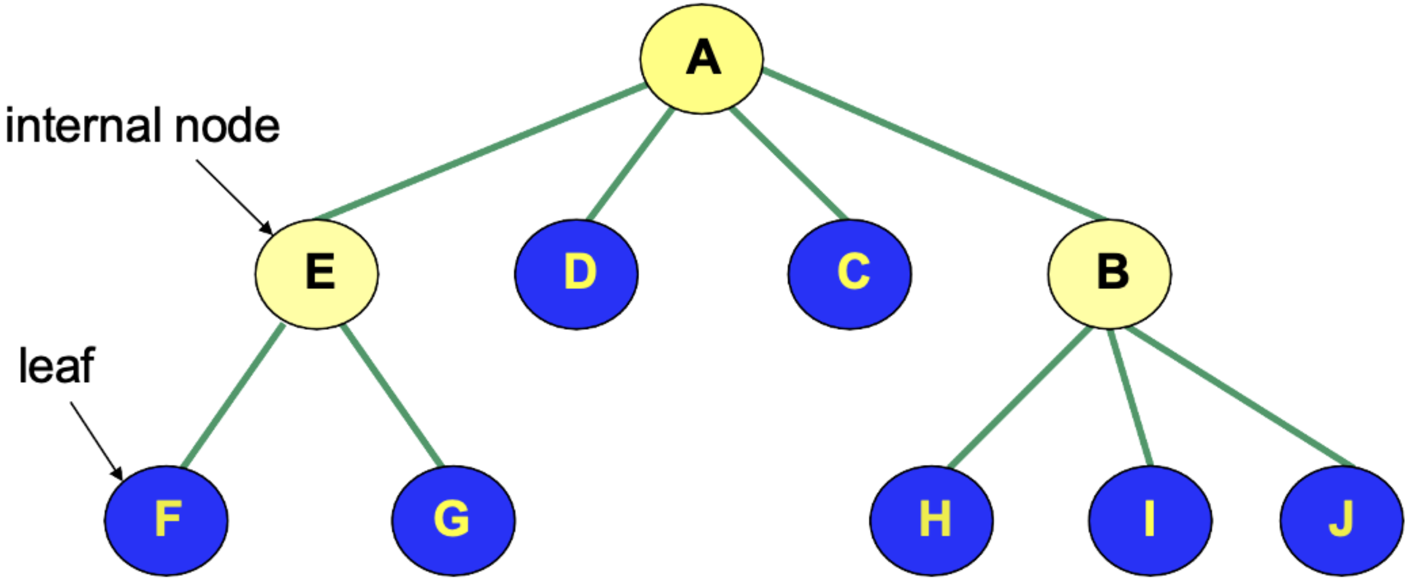
Tree Terminology

- **Parent**: the node directly above another node in the tree
- **Child**: a node directly below another node in the tree
- **Siblings**: nodes that have the same parent
- **Ancestors** of a node: its parent, the parent of its parent, etc.
- **Descendants** of a node: its children, the children of its children, etc.



Tree Terminology

- **Leaf node**: a node without children
- **Internal node**: a node that is not a leaf node

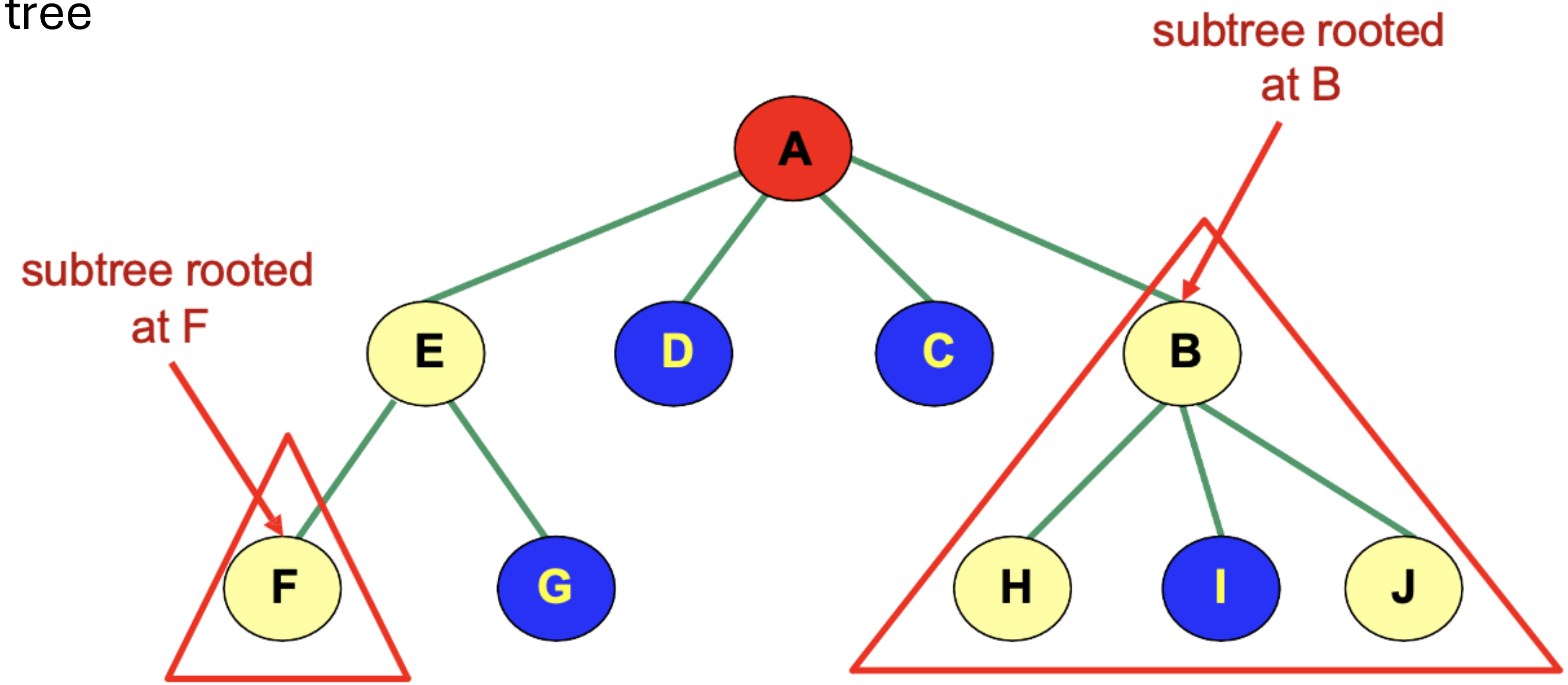


Stop & Think

- Does a leaf node have any children?
- Does the root node have a parent?
- How many children can a node have?

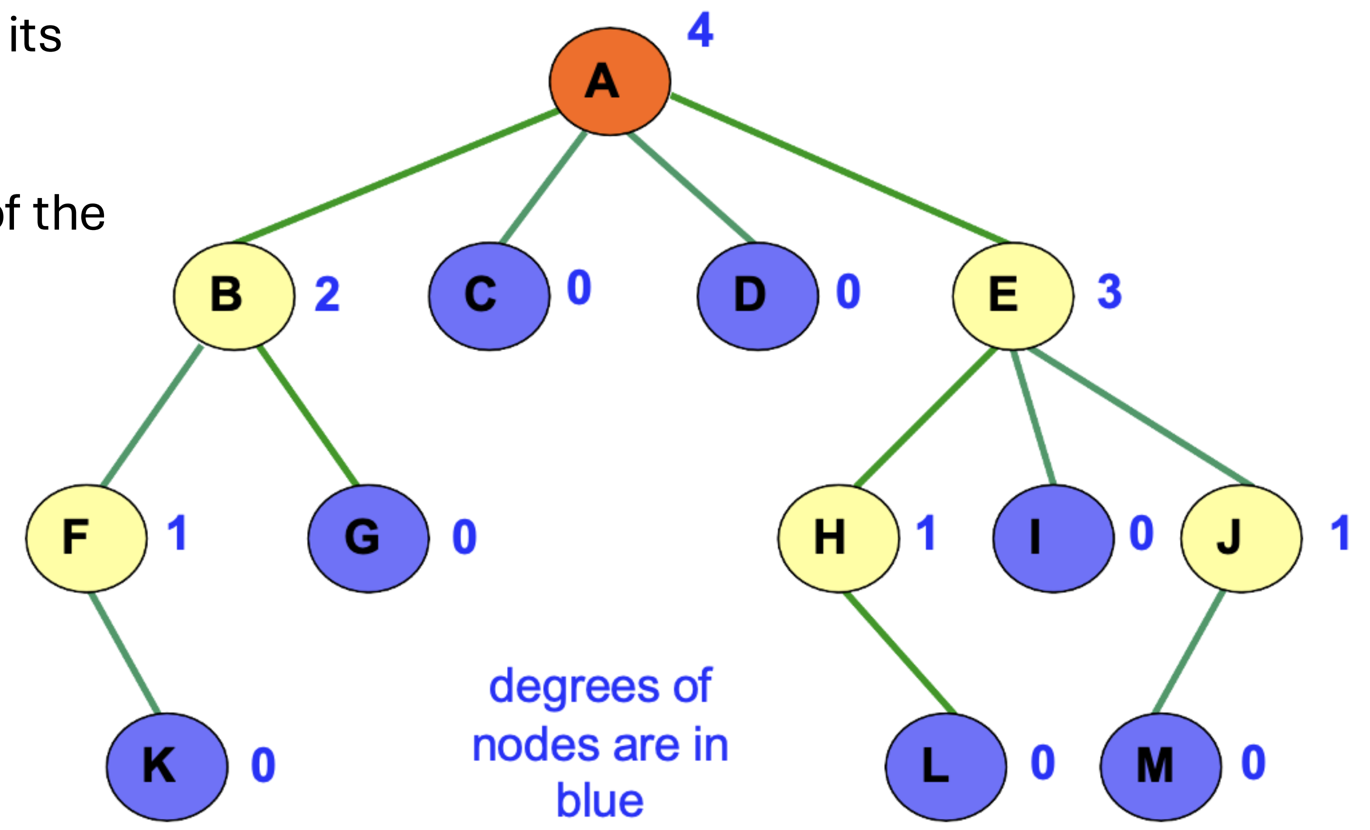
Subtrees

- The subtree rooted at a node consists of the node and all its descendants
- A subtree is itself a tree



Tree Terminology

- **Degree of a node:** the number of its children
- **Degree of a tree:** the maximum of the degrees of the tree's nodes

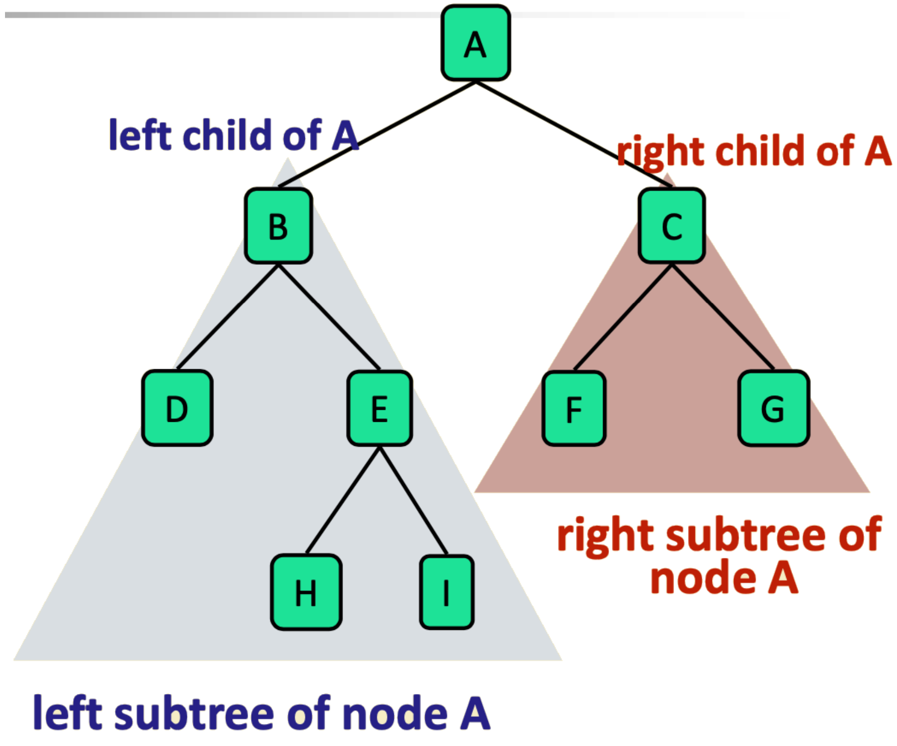


Classification of Trees

- Trees can be classified into many categories by their **properties** and **applications**. We will look into the following categories.
  - General trees – no restriction
  - Binary trees – each node has at most two children.
  - Binary search trees – binary trees for efficient searching
    - Ex. **AVL trees:** height-balanced binary search trees
  - Multi-way search trees – a generalization to binary search trees
    - Ex. **B-trees** – balanced multi-way search trees

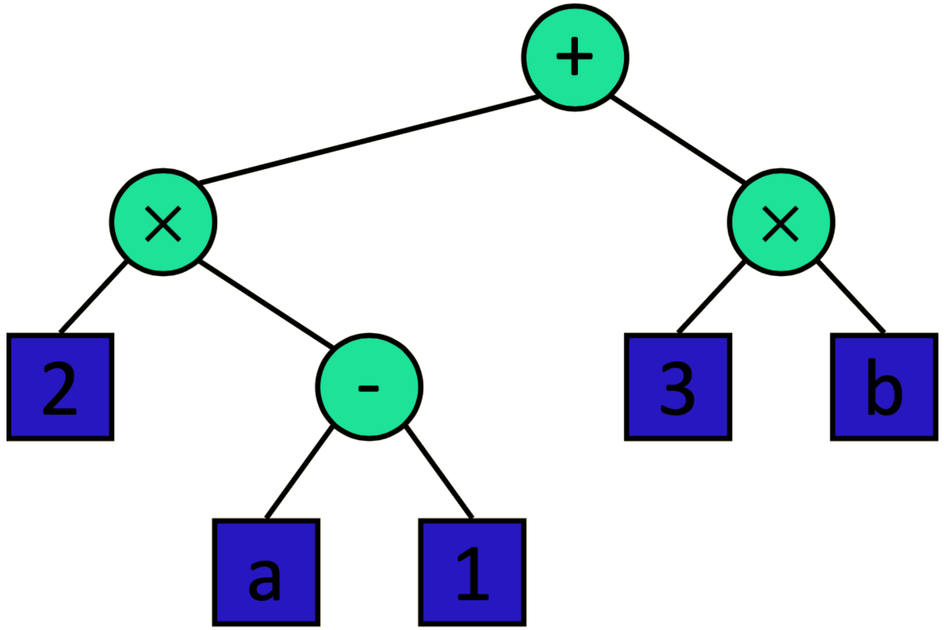
Binary Trees (BTs)

- In a **Binary tree,** a node has at most two children
- Children are an ordered pair
  - **left child** and **right child**
  - corresponding subtrees are the **left subtree** and **right subtree**
- In a binary tree, each internal node has exactly two children
- Applications
  - Arithmetic expressions
  - Decision processes



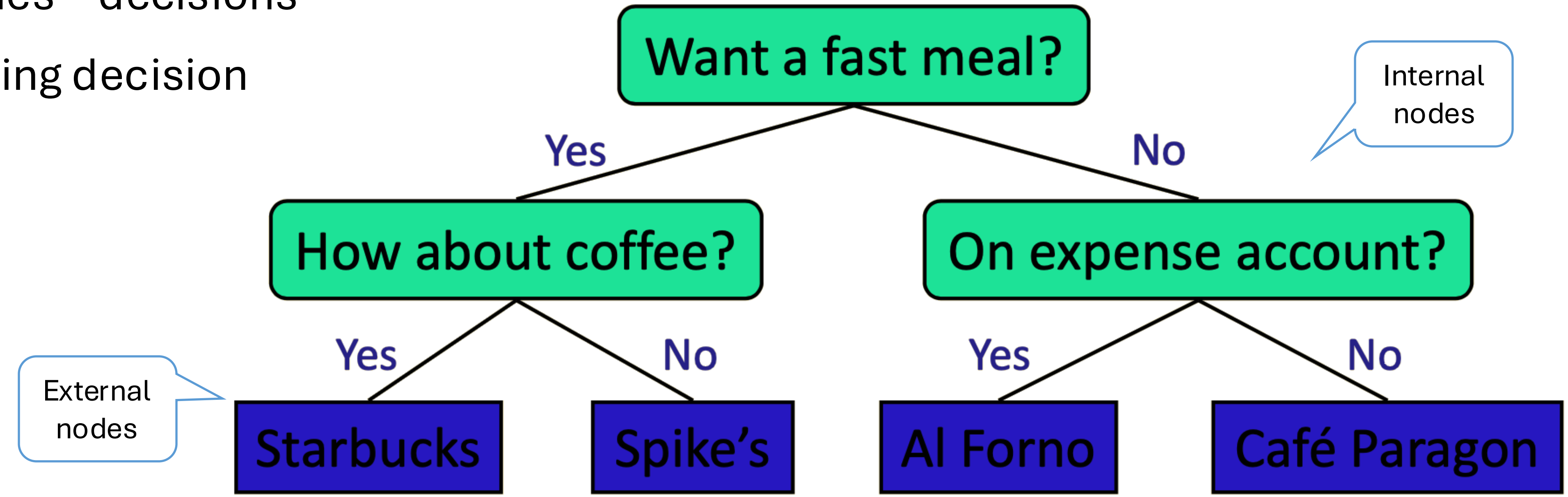
Arithmetic Expression Tree

- The binary tree associated with an arithmetic expression
  - internal nodes store operators
  - external nodes store operands
- Example: arithmetic expression tree for the expression: (2 × (a - 1) + (3 × b))



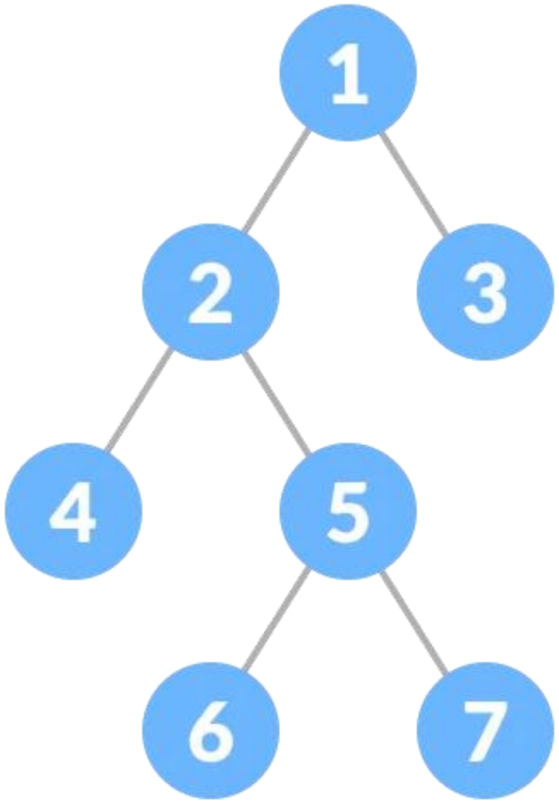
Decision Tree

- Binary tree associated with a decision process
- Internal nodes – questions with yes/no answer
- External nodes – decisions
- Example dining decision



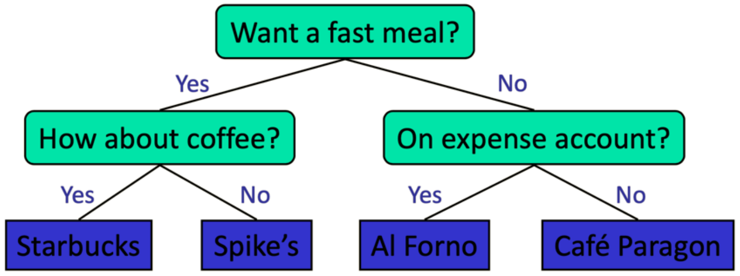
Types of BTs: Full Binary Tree

- A **full** Binary tree is a special type of binary tree in which every parent node/internal node has either **two** or **no** children.
- It is also known as a **proper** binary tree.



Properties of Proper BT

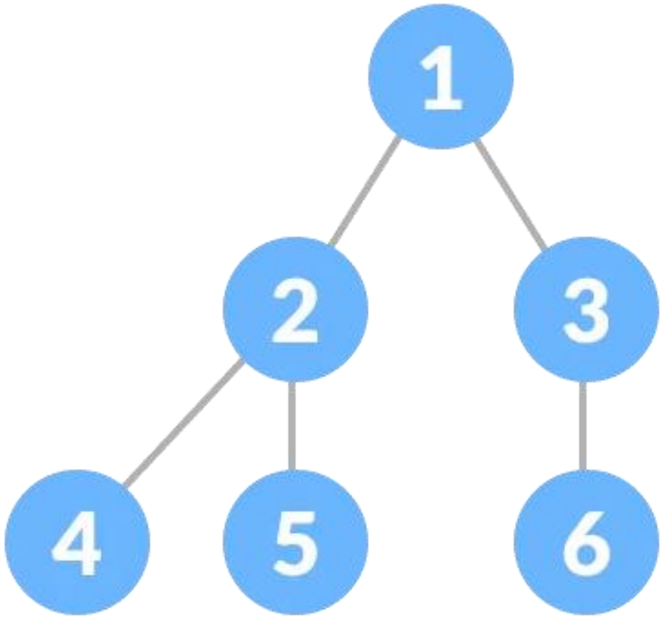
- Notations
  - **n** number of all nodes
  - **e** number of external nodes
  - *i* number of internal nodes
  - ***d*** depth
  - ***h*** height



1. The number of leaves is $i + 1$.
2. The total number of nodes is $2i + 1$.
3. The number of internal nodes is $(n – 1) / 2$.
4. The number of leaves is $(n + 1) / 2$.
5. The total number of nodes is $2e – 1$.
6. The number of internal nodes is $e – 1$.
7. The number of leaves is at most $2^{h - 1}$

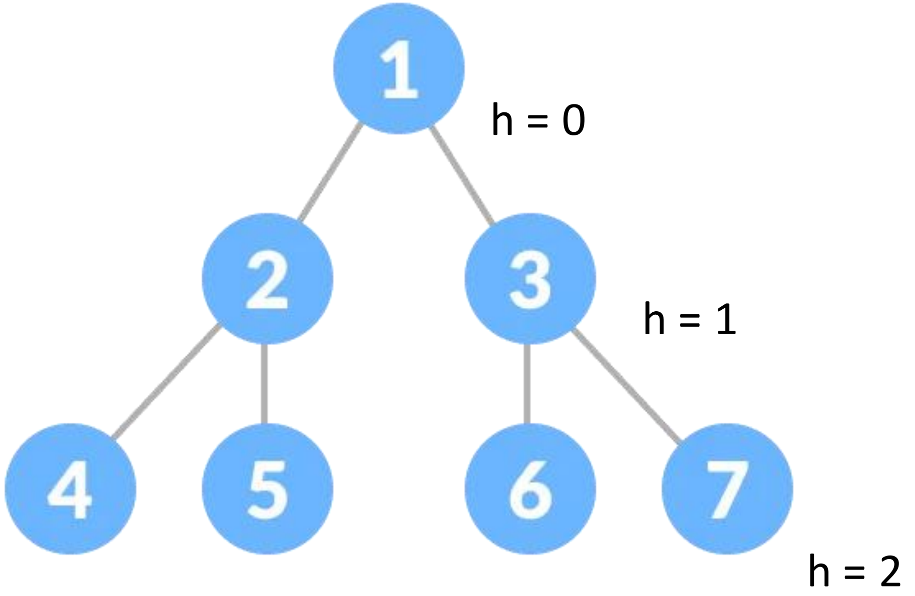
Types of BTs: Complete Binary Tree

- A complete binary tree is a binary tree in which every level, except possibly the last, is **completely filled**, and all nodes are as far left as possible.
- This means that:
  - All levels above the last level are fully filled.
  - The last level may not be fully filled, but if it has missing nodes, those nodes are only on the right side (i.e., **all leaf nodes lean to the left**).



Types of BTs: Perfect Binary Tree

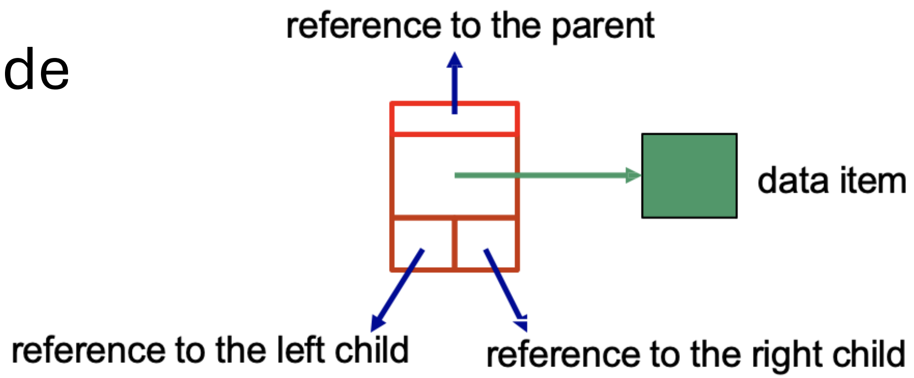
- A perfect binary tree is a binary tree in which every internal node has exactly **two child nodes** and <u>all the leaf nodes are at the same level</u>.
- A perfect binary tree of height h has $2^{h+1} - 1$ node.
- A perfect binary tree of height h has 2^h leaf nodes.



Linked Binary Tree Implementation

- To represent a binary tree, we will use a linked structure of nodes
  - root: reference to the node that is the root of the tree
  - count: keeps track of the number of nodes in the tree

- First, how will we represent a *node of a binary tree*?
- A binary tree node will contain
  - a **reference** to the data stored in the node
  - **references** to its left and right children
  - [optionally] a reference to its parent

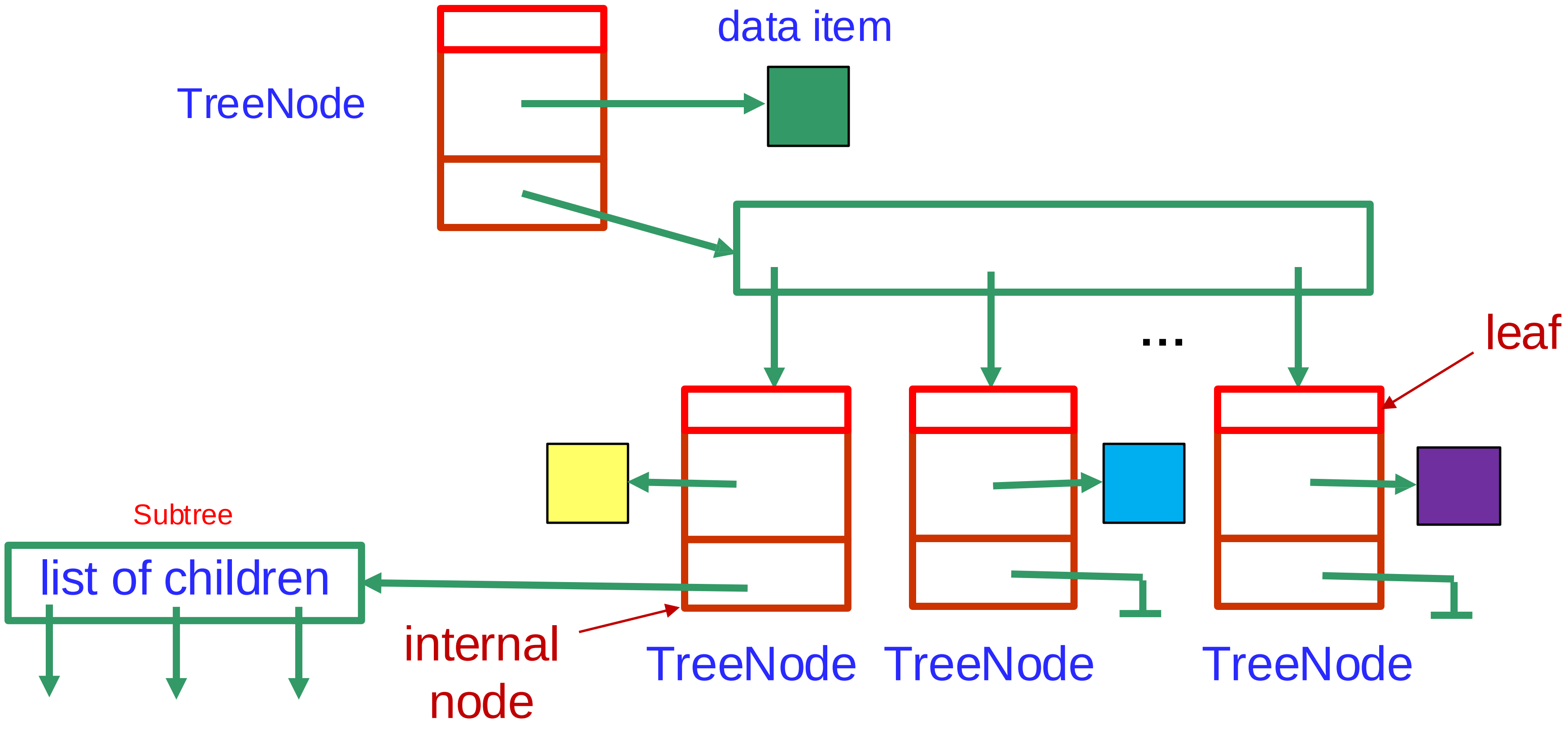


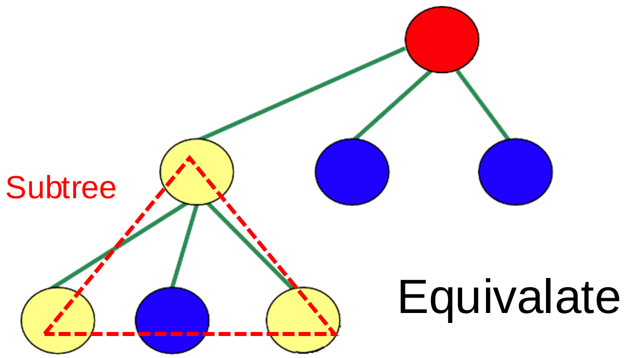
Linked Binary Tree Implementation

```java
public class BinaryTreeNode<T> {
private T dataItem;
private BinaryTreeNode<T> parent, leftChild, rightChild;
/* Creates a new tree node with the specified data. */
BinaryTreeNode (T newData) {
dataItem = newData;
leftChild = null; rightChild = null; parent = null;
}
// Getter and setter methods
public BinaryTreeNode<T> getParent() {
…
```

A TreeNode Implementation (v1)

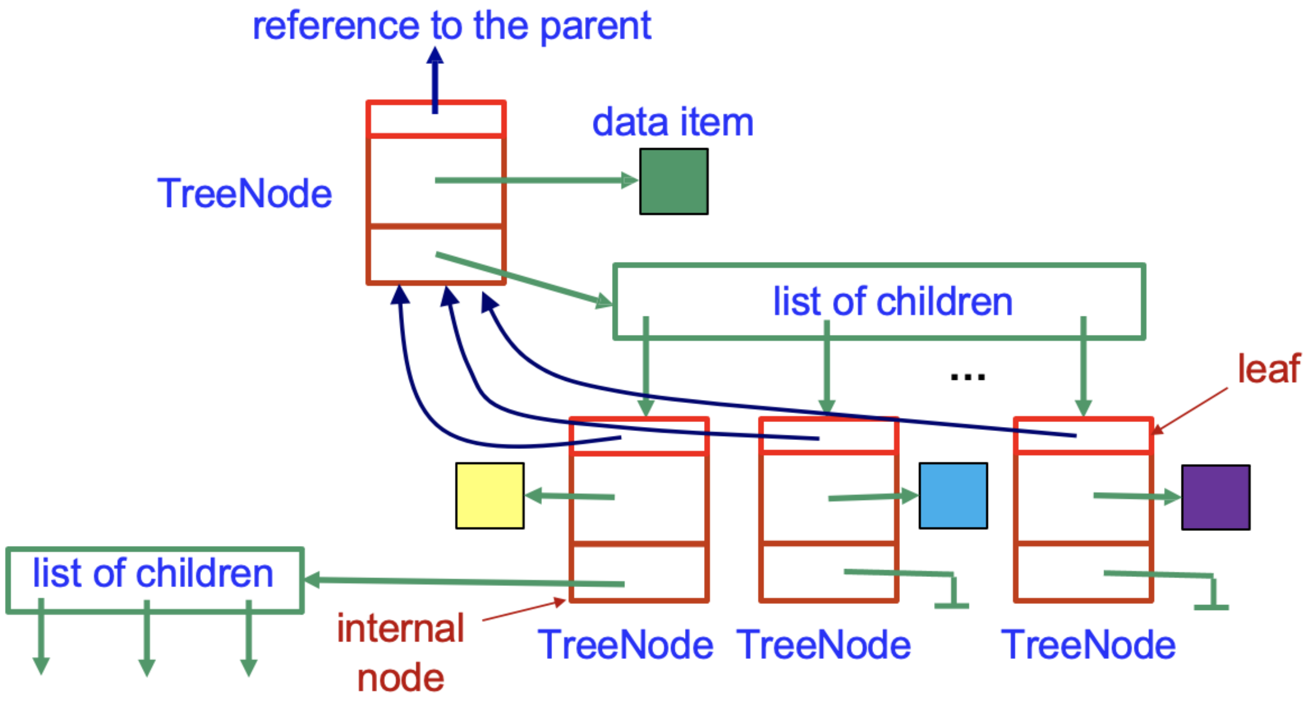
Memory:





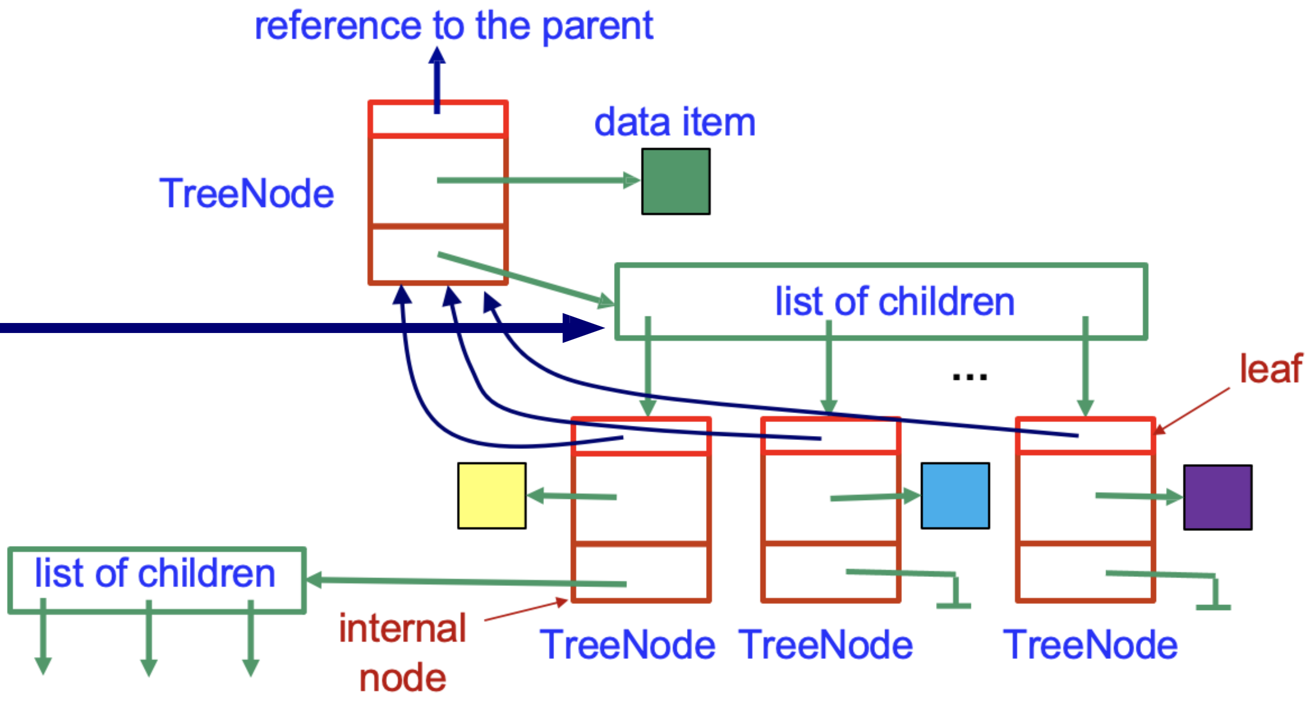
A TreeNode Implementation (v2)

- If the tree is not binary, then each node will have a **reference** to the data item it stores, a **reference** to its parent, and a **reference** to a list of its children.



A TreeNode Implementation (v2)

- The children of a node can be stored in an array, a circular array, a singly linked list, a doubly linked list, or any other data structure implementing a list.



Linked Binary Tree Implementation

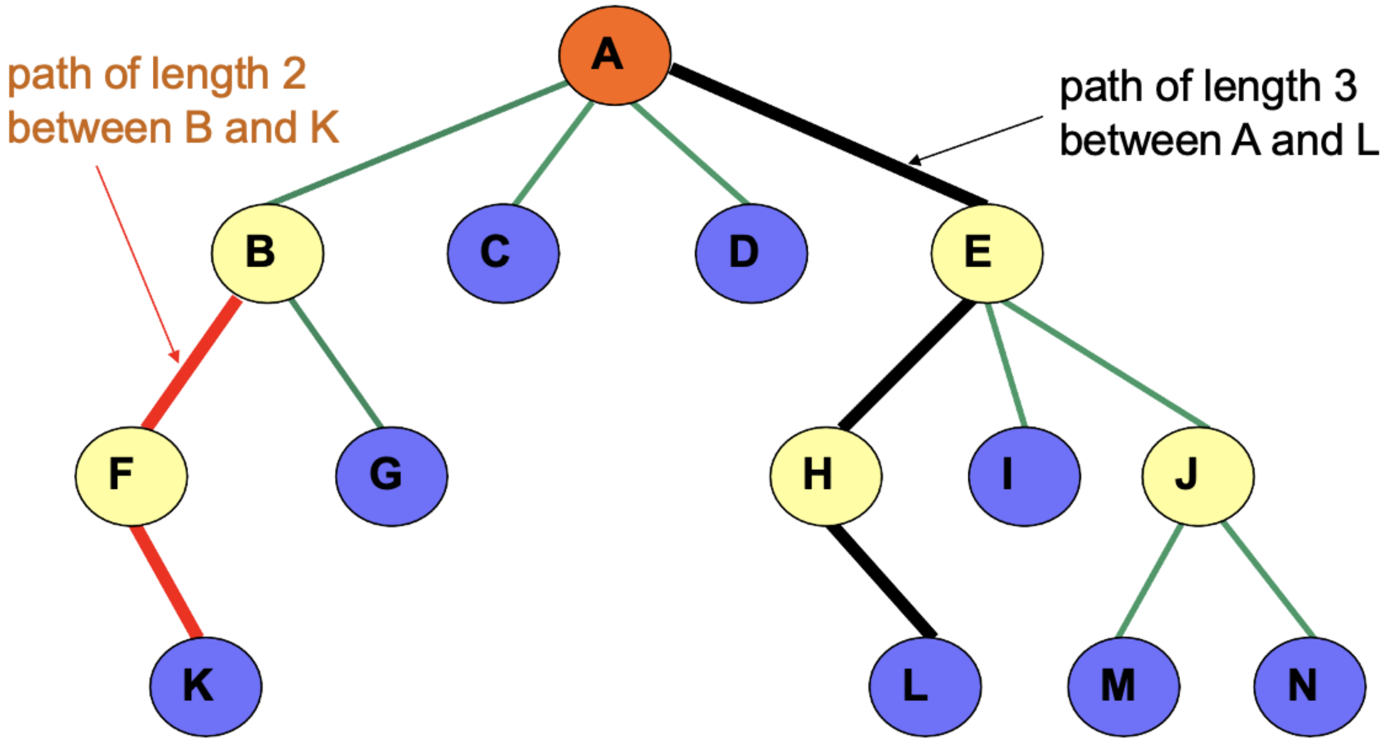
- This implementation uses an array to store the list of children:

```
public class TreeNode<T> {
private final int DEFAULT_CAPACITY = 10;
private T dataItem;
private TreeNode<T> parent; //optional
private TreeNode<T>[] children;
private int numChildren;

/* Creates a new tree node with the specified data. */
TreeNode (T newData) {
dataItem = newData;
parent = null; //optional
children = new TreeNode<T>[DEFAULT_CAPACITY];
numChildren = 0;
}
```

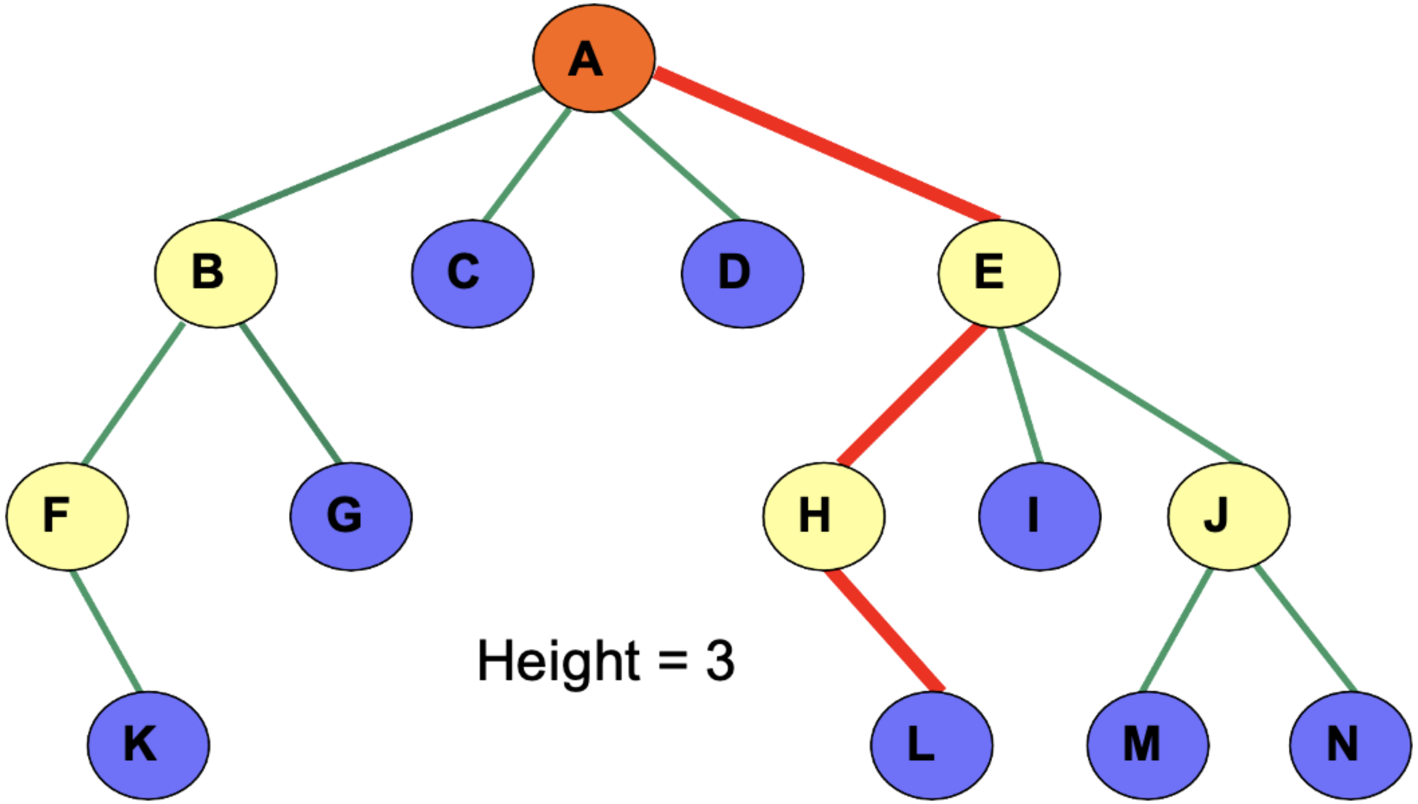
Tree Terminology

- A path is a sequence of edges leading from one node to another
- Length of a path: number of edges on the path



Tree Terminology

- Height of a tree: length of the longest path from the root to a leaf
- What is the height of a tree that has only a root node?

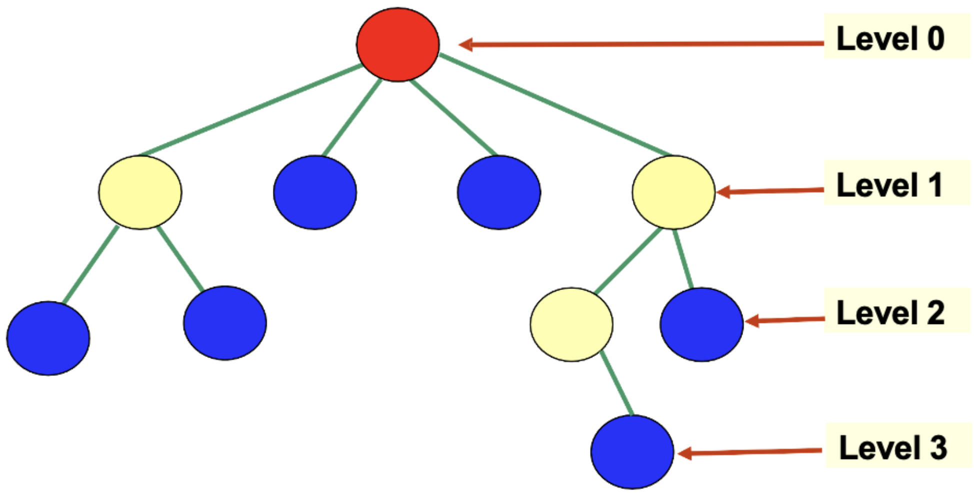


Level of a Node

- Level of a node: number of edges between the root and the node
- It can be defined recursively:
  - The level of the root node is 0
  - The level of a node that is not the root is the level of its parent + 1.

What is a tree's height (h) in terms of levels?

In terms of levels, the tree has a height of **3**, because there are 3 levels.



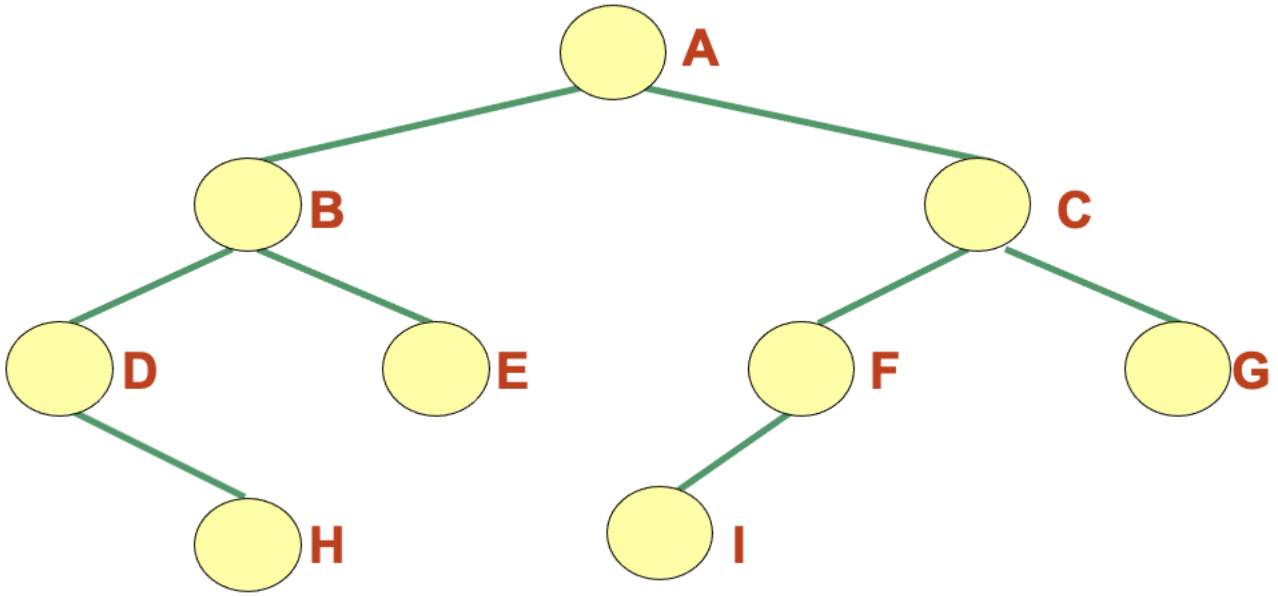

```
Algorithm level (node)
Input: node of a tree
Output: level of the node
```

```
public int level (TreeNode<T> node) {
// Input: node of a tree
// Output: level of the node
if (node.getParent() == null) return 0;
else return 1 + level(node.getParent());}
```

Tree Traversals

- Given the root node of a tree, a traversal requires visiting each node once.
- Note that the only node we know of in a tree is its root. Using a tree traversal, we must be able to access all the other nodes in the tree from the root node.
- Common tree traversals:
  - preorder
  - postorder
  - level-order
- For binary trees, there is another traversal:
  - inorder

Binary Tree Traversals



We will consider only traversals of binary trees. We will study the different tree traversals using this and other trees.

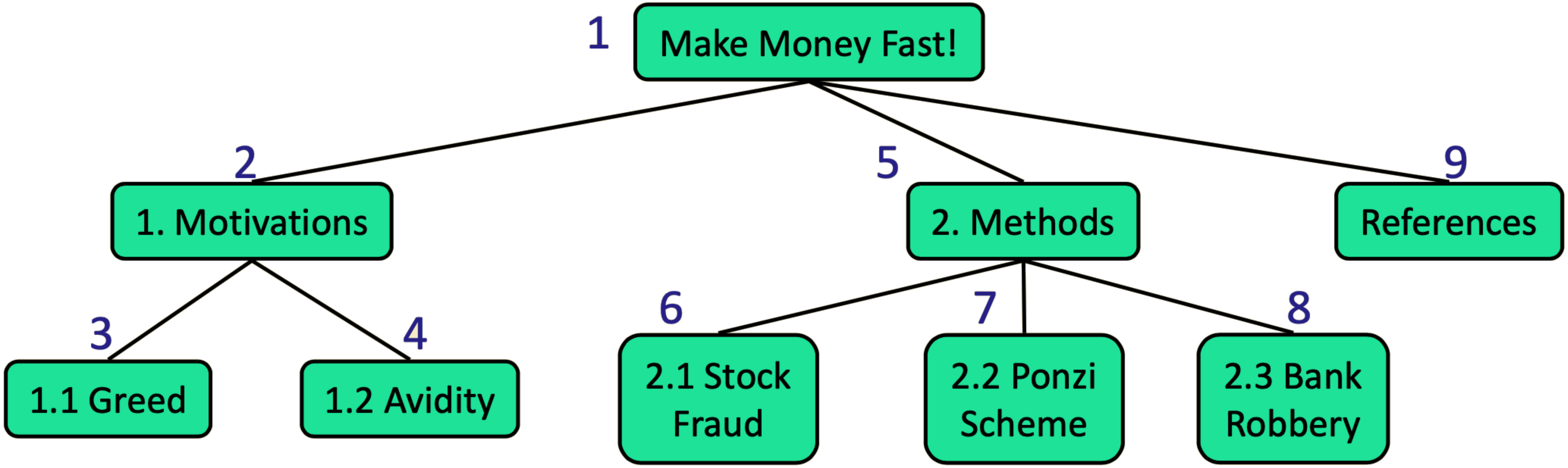
Pre-order Traversal

- If the tree is not empty:
  - Visit the root node of the tree
  - Perform pre-order traversal of the left subtree
  - Perform pre-order traversal of the right subtree
- This is a recursive algorithm for performing a pre-order traversal of a tree.
  - *What is the base case?*
  - *What is the recursive case?*

```
public void preorder (BinaryTreeNode<T> r)
{ if (r != null) {
  visit(r); // This method depends on the
  // application traversing the tree
  preorder (r.getLeftChild());
  preorder (r.getRightChild());
  }
}
```

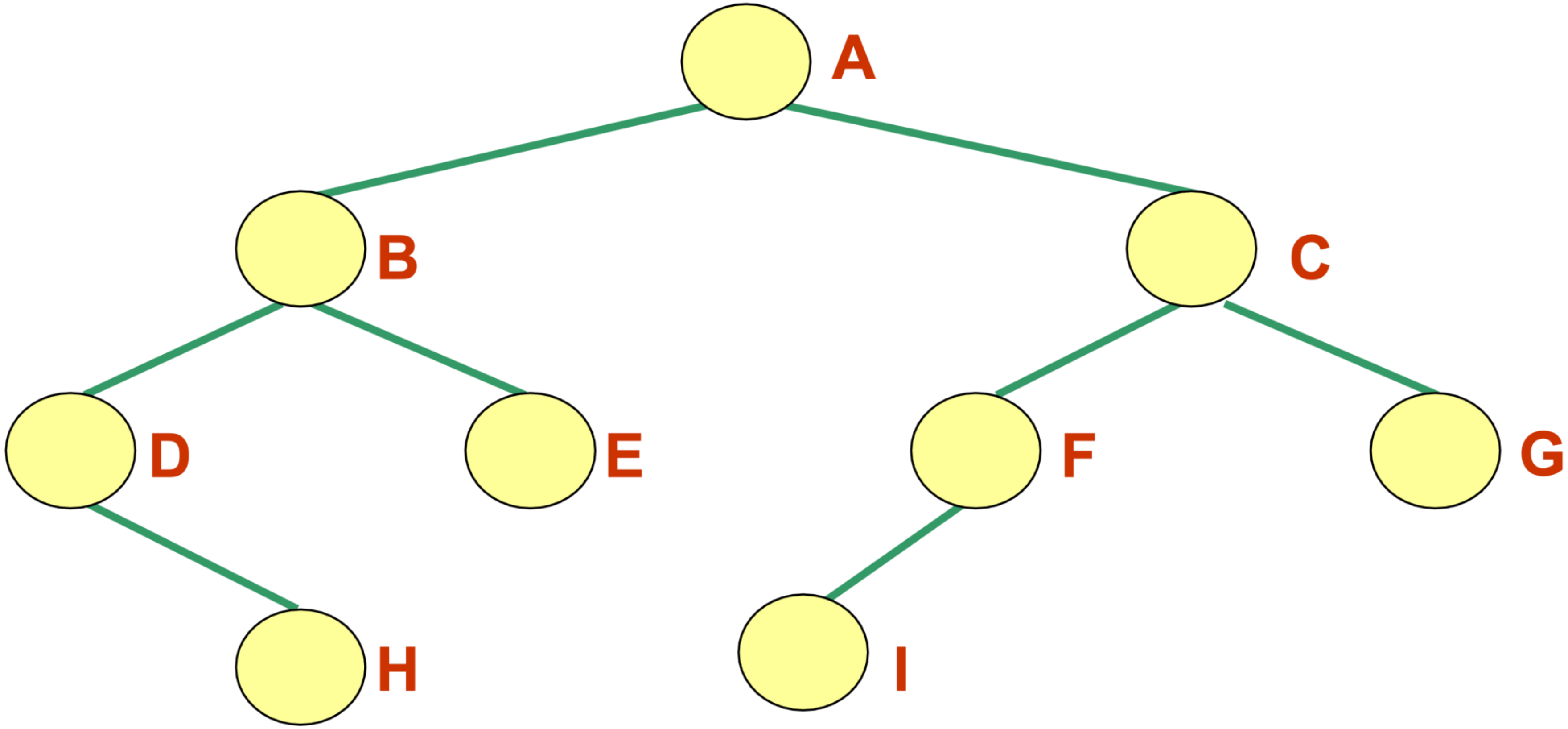
Pre-order Traversal

- A node is visited before its descendants.
- When is it applied?
  - Use when must perform computations for a node before any computations for its descendants



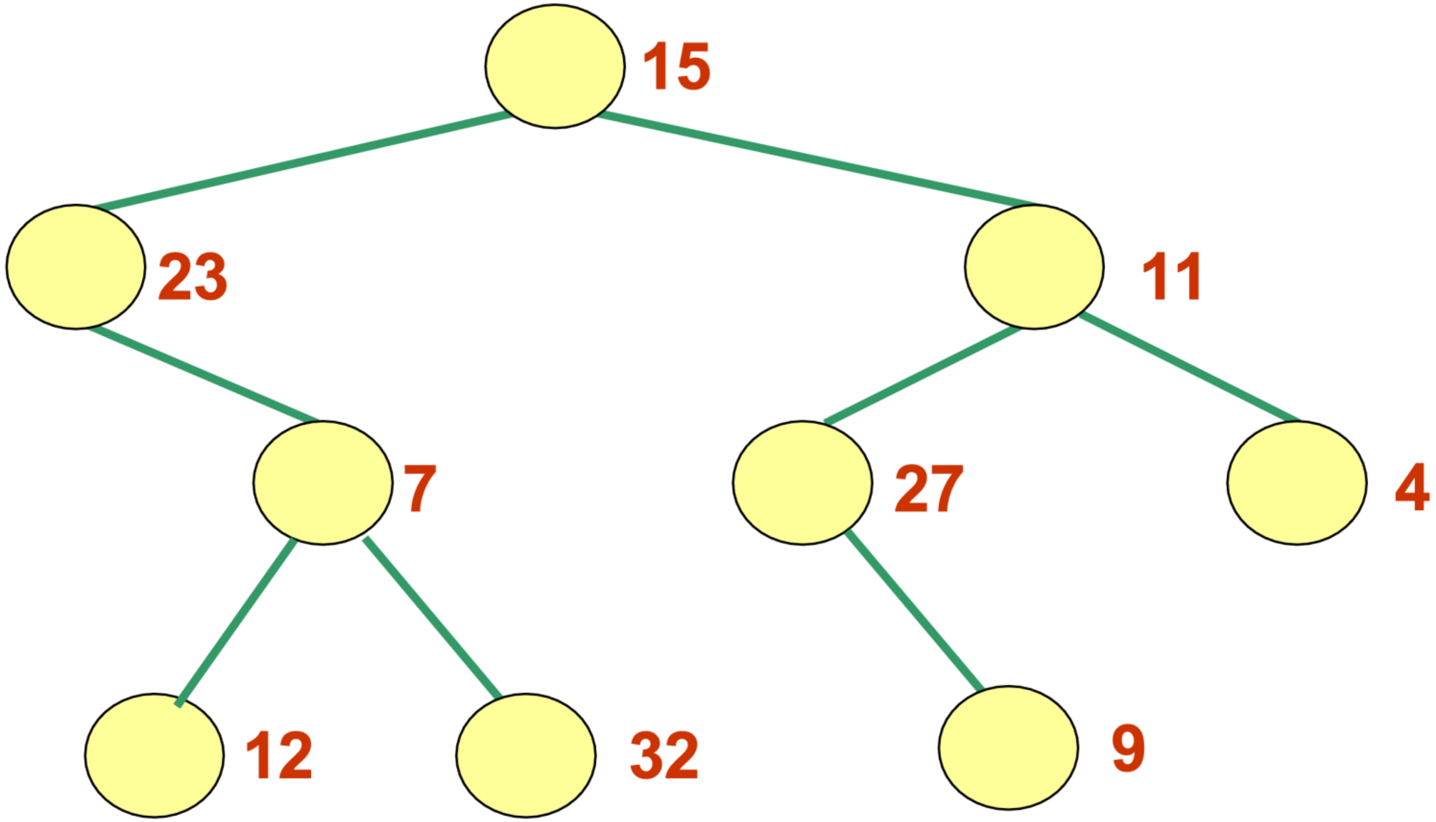
Pre-order Traversal Example 1

- Pre-order traversal: A B D H E C F I G



Pre-order Traversal Example 2

- Pre-order traversal: 15 23 7 12 32 11 27 9 4



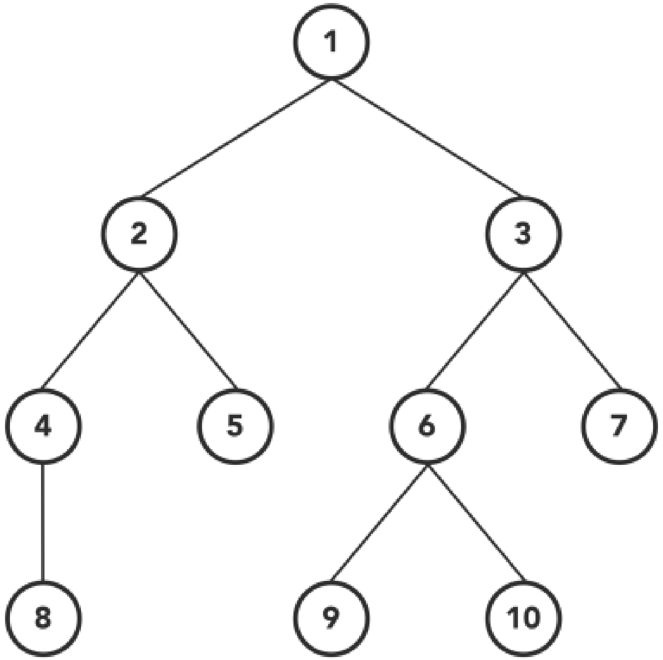
In-order Traversal

- If the tree is not empty,
  - Perform in-order traversal of the left subtree
  - Visit the root node of the tree
  - Perform in-order traversal of the right subtree

- This is a recursive algorithm for performing an in-order traversal of a tree.
  - *What is the base case?*
  - *What is the recursive case?*

```
public void inorder
(BinaryTreeNode<T> r) {
if (r != null) {
  inorder (r.getLeftChild());
  visit(r);
  inorder (r.getRightChild());
  }
}
```

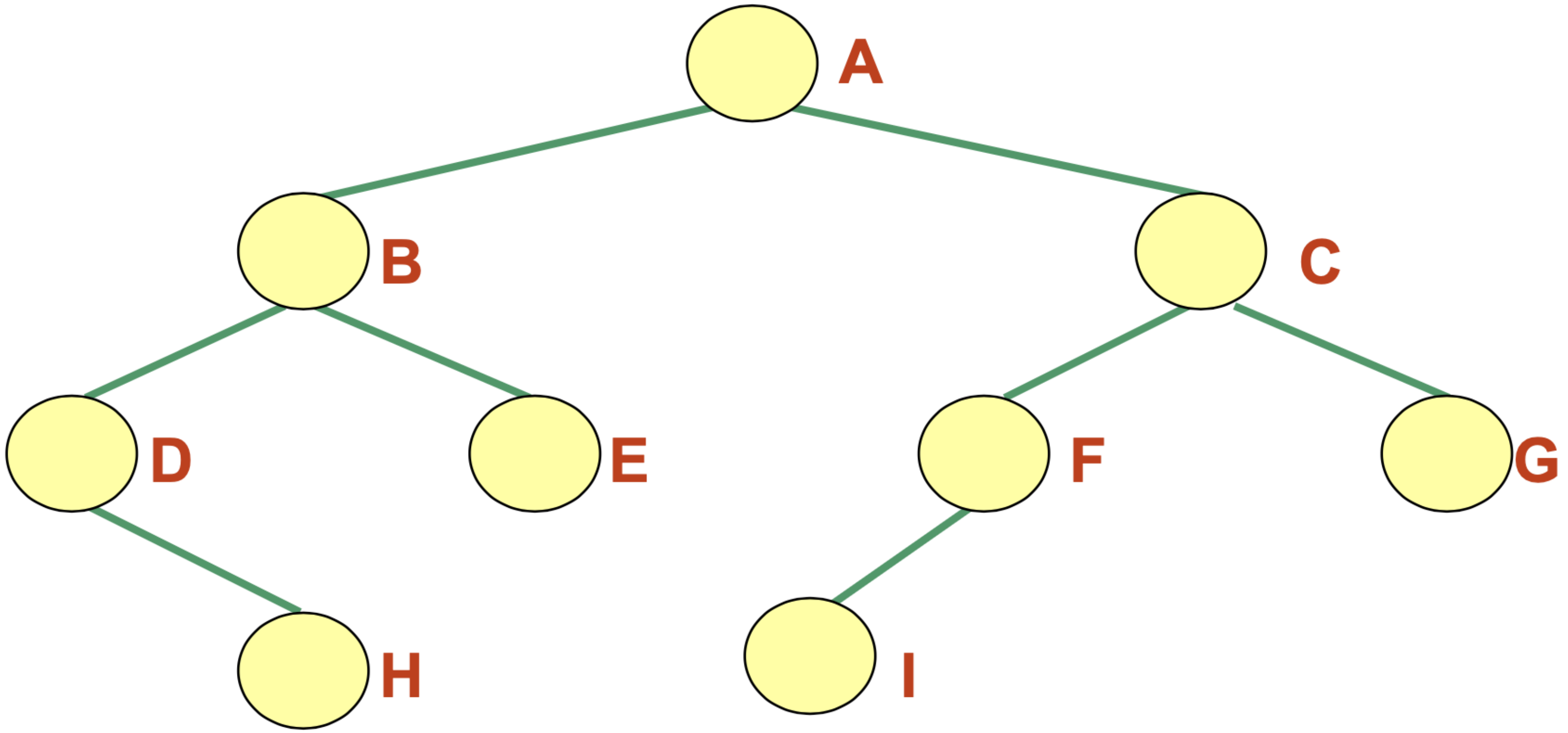
In-order Traversal




```
public void inorder (BinaryTreeNode<T> r)
{
if (r != null) {
  inorder (r.getLeftChild());
  visit(r);
  inorder (r.getRightChild());
   }
}
```

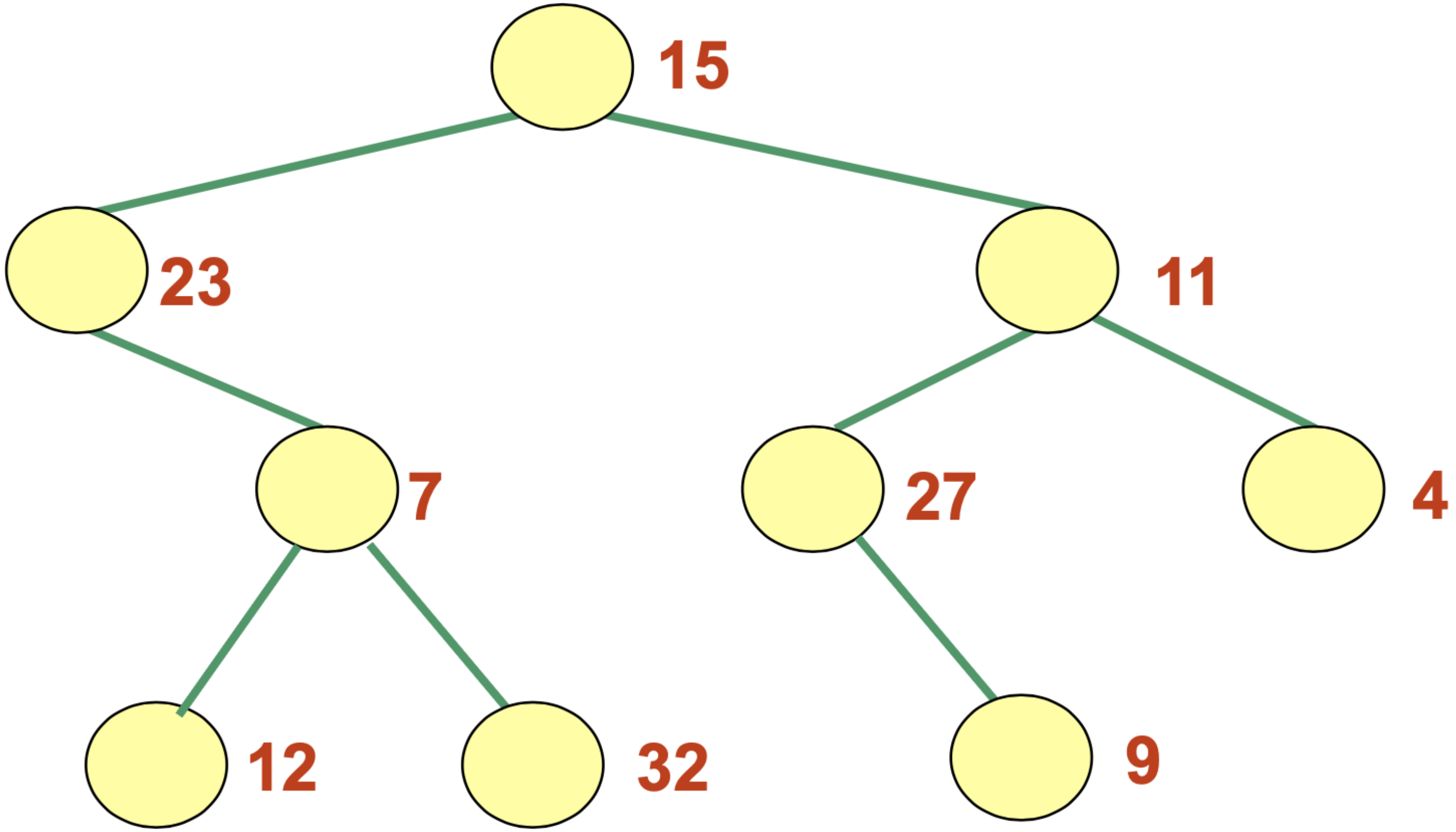
In-order Traversal Example 1

- In-order traversal: D H B E A I F C G



In-order Traversal Example 2

- In-order traversal: 23 12 7 32 15 27 9 11 4



Post-order Traversal

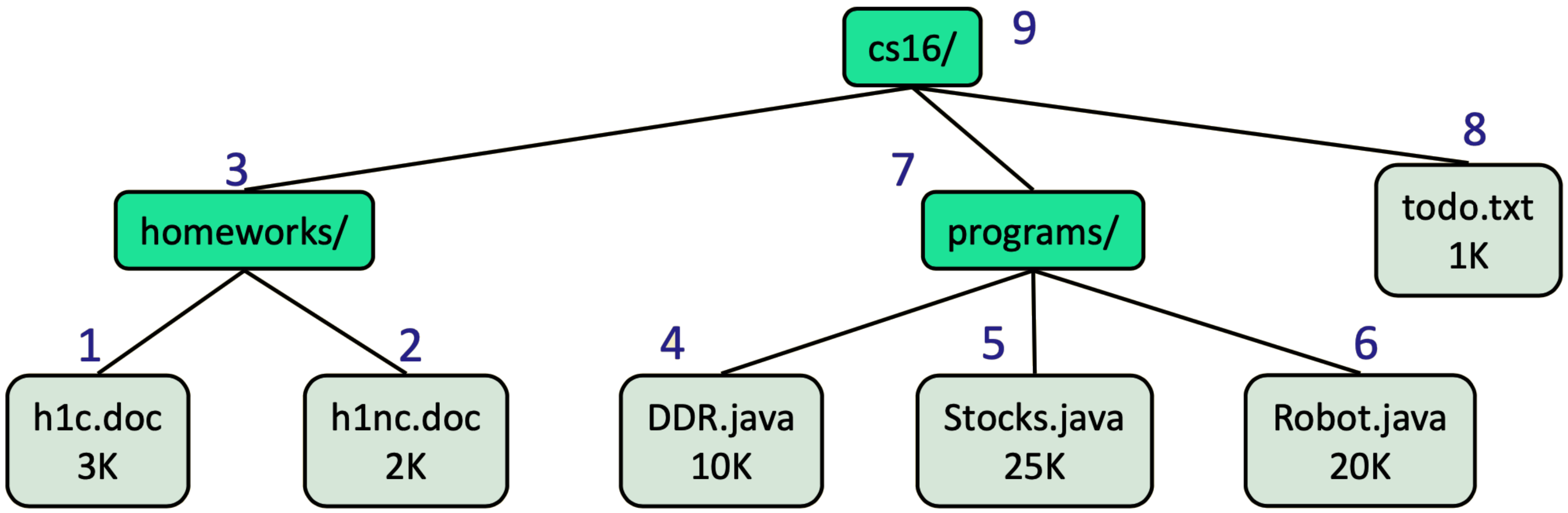
- If the tree is not empty,
  - Perform post-order traversal of the left subtree
  - Perform post-order traversal of the right subtree
  - Visit the root node of the tree

- This is a recursive algorithm for performing a post-order traversal of a tree.
  - *What is the base case?*
  - *What is the recursive case?*

```java
public void postorder (BinaryTreeNode<T> r)
{
  if (r != null) {
  postorder (r.getLeftChild());
  postorder (r.getRightChild());
  visit(r);
 }
}
```

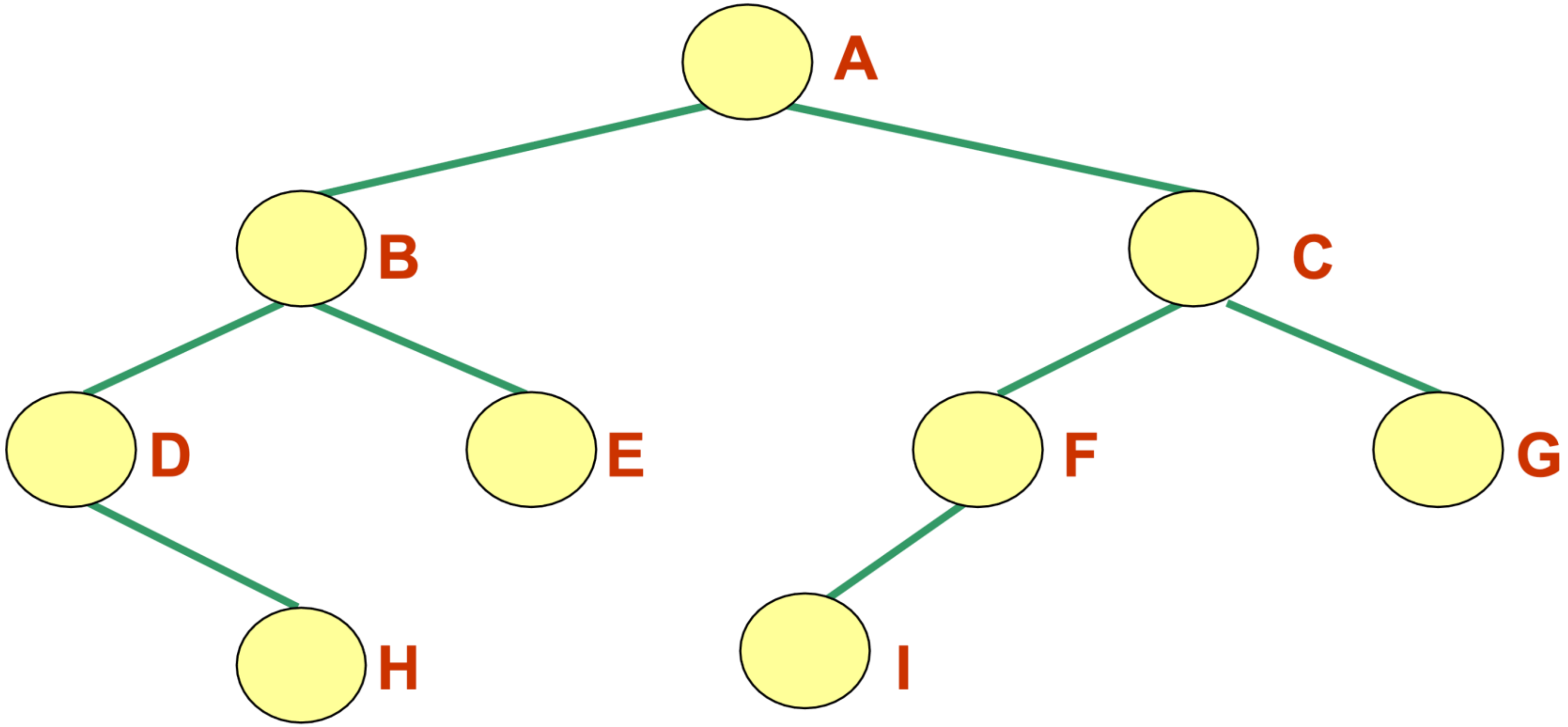
Post-order Traversal

- Node is visited after its descendants
- When is it applied?
  - Visit leaf nodes first
  - trying to delete a tree



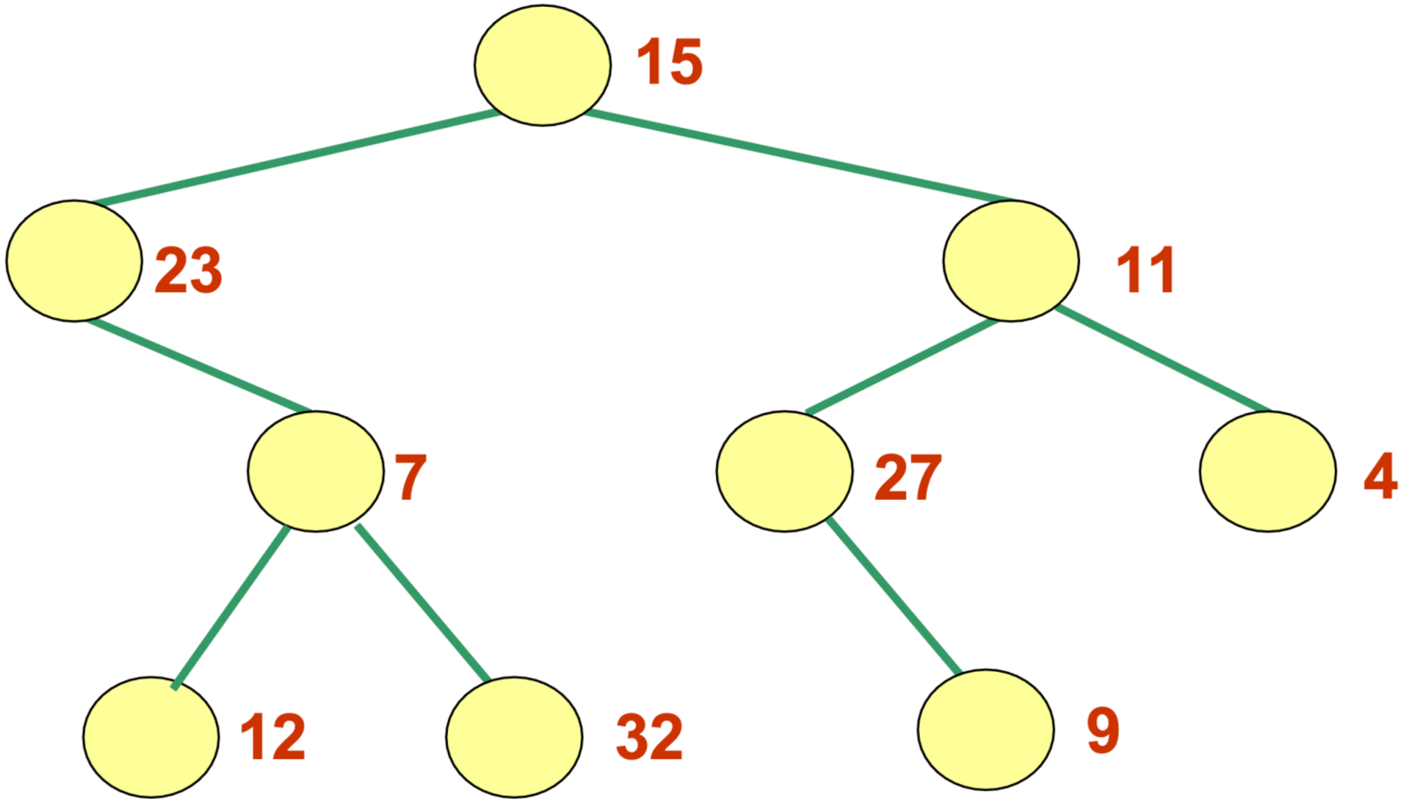
Post-order Traversal Example 1

- Post-order traversal: H D E B I F G C A



Post-order Traversal Example 2

- Post-order traversal: 12 32 7 23 9 27 4 11 15




Thank you