# ICS 240
# Introduction to Data Structures

Jessica Maistrovich

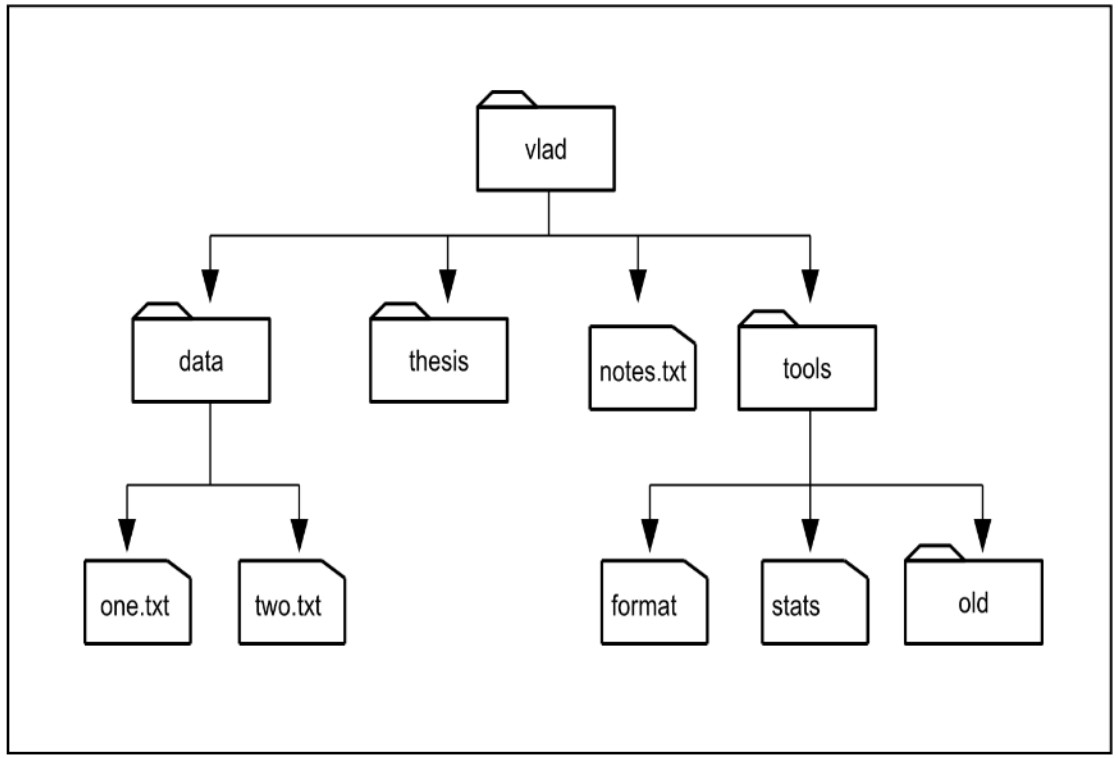
Metropolitan State University

# Trees

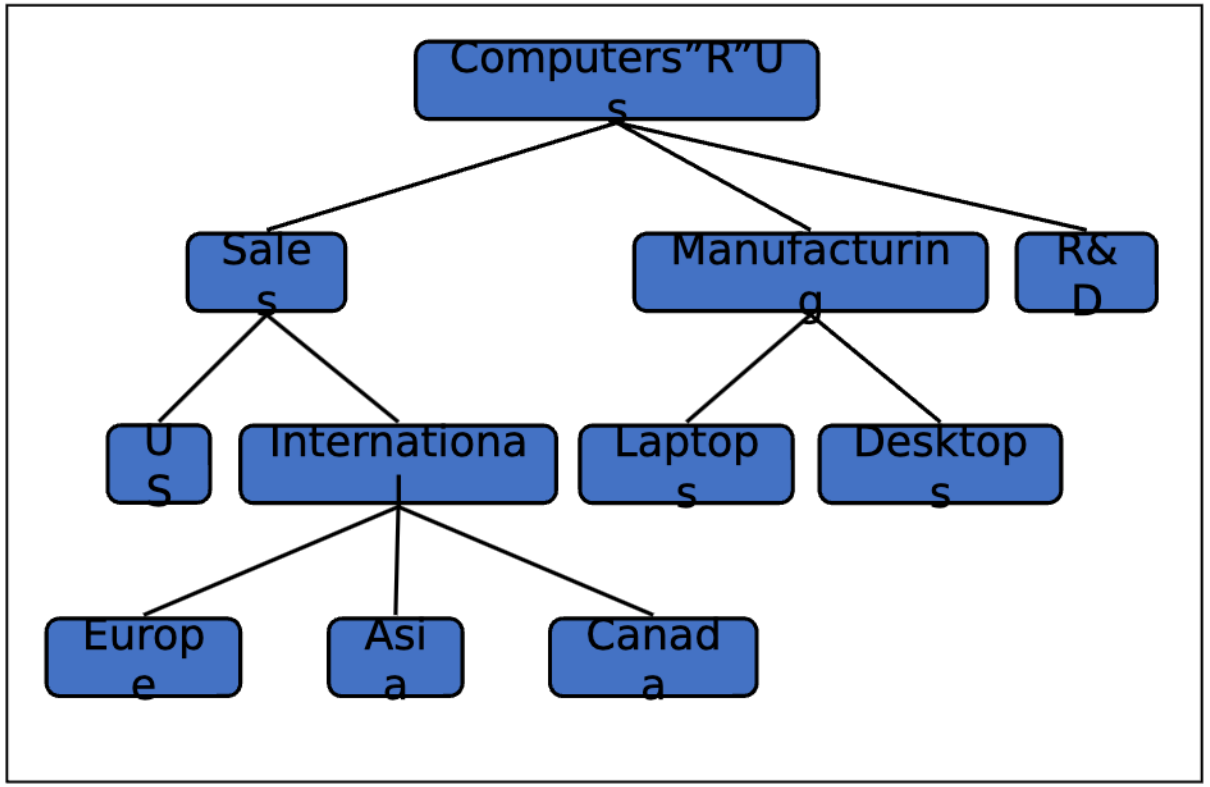
New Data Structure!

# Array or linked list?

## File System

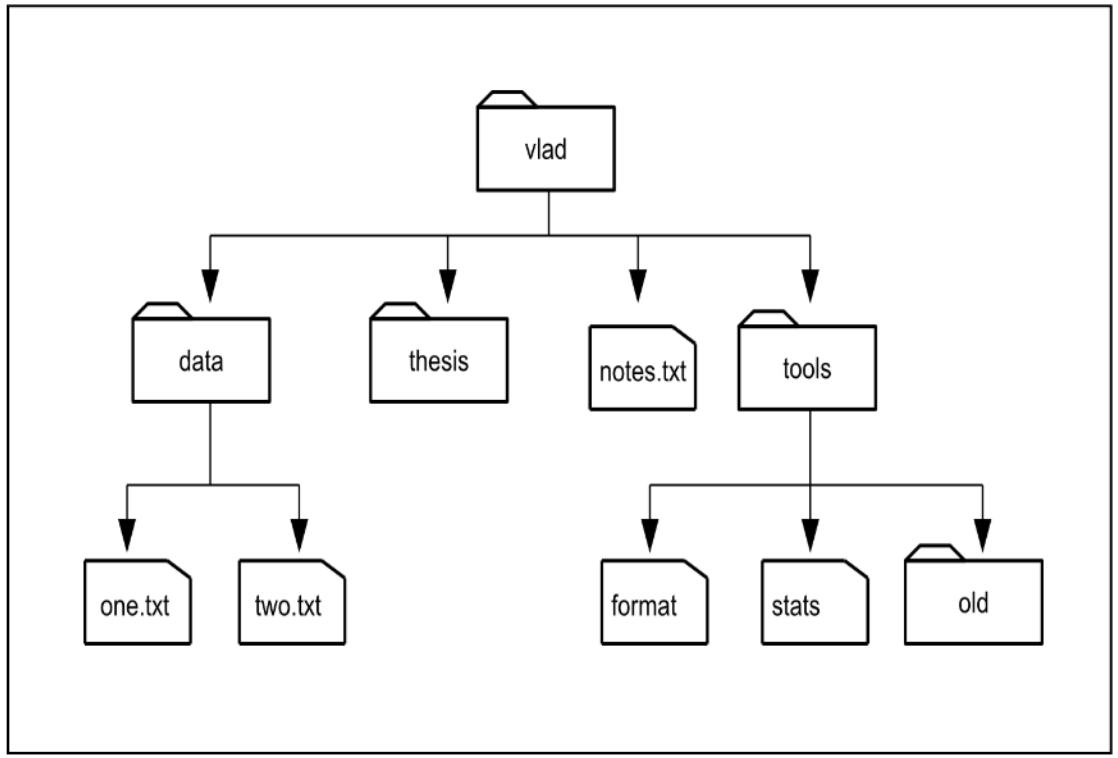



## Organization Char

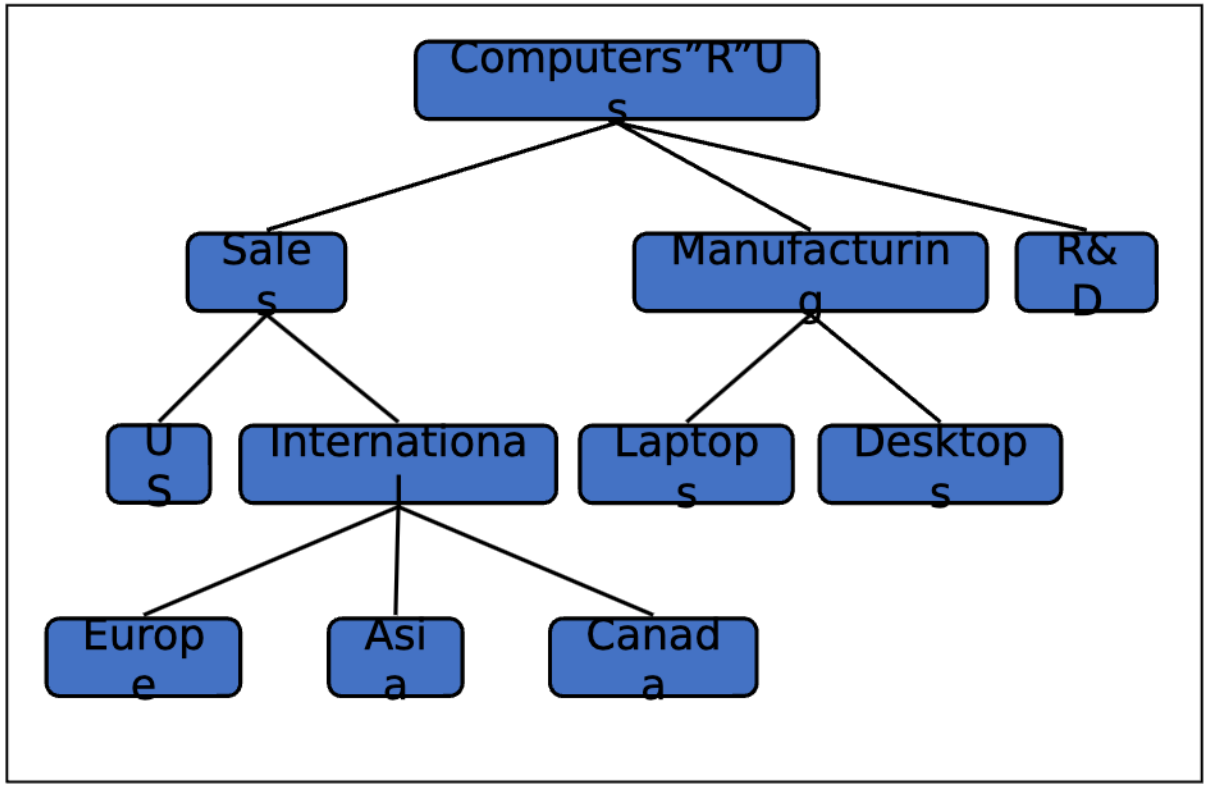


# Look like nodes!  What if we arranged our nodes in this shape?

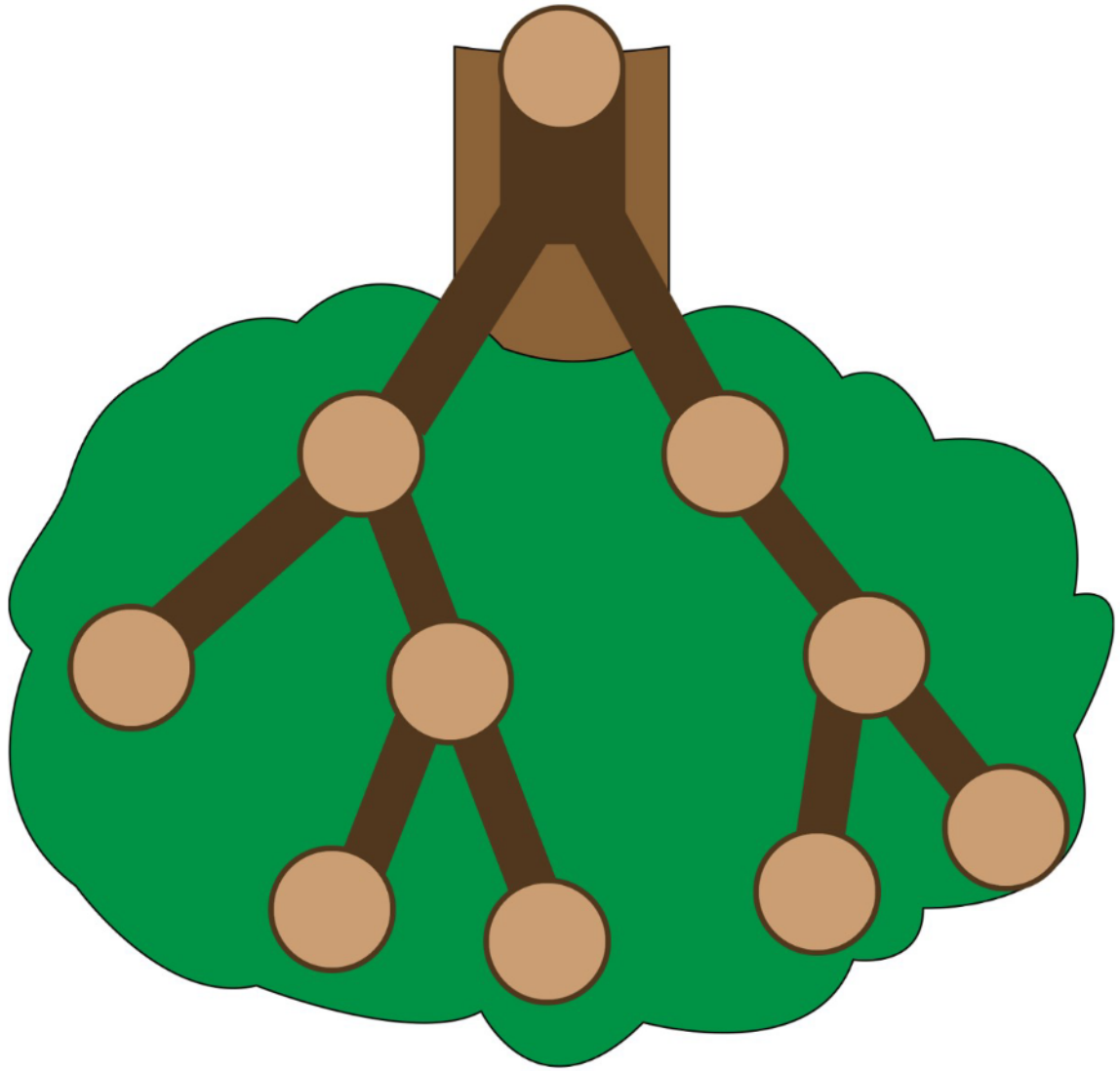
## File System



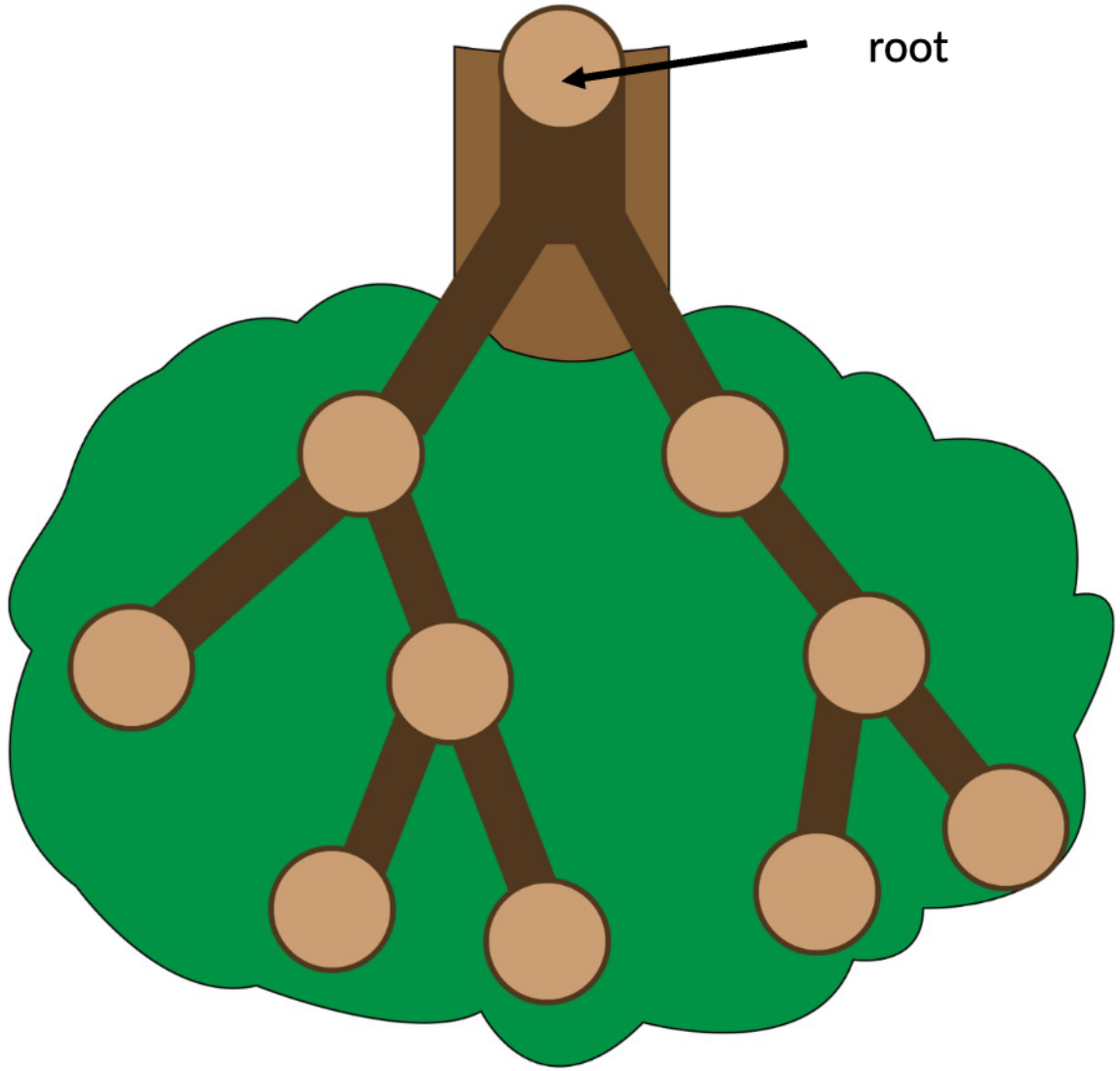

## Organization Char



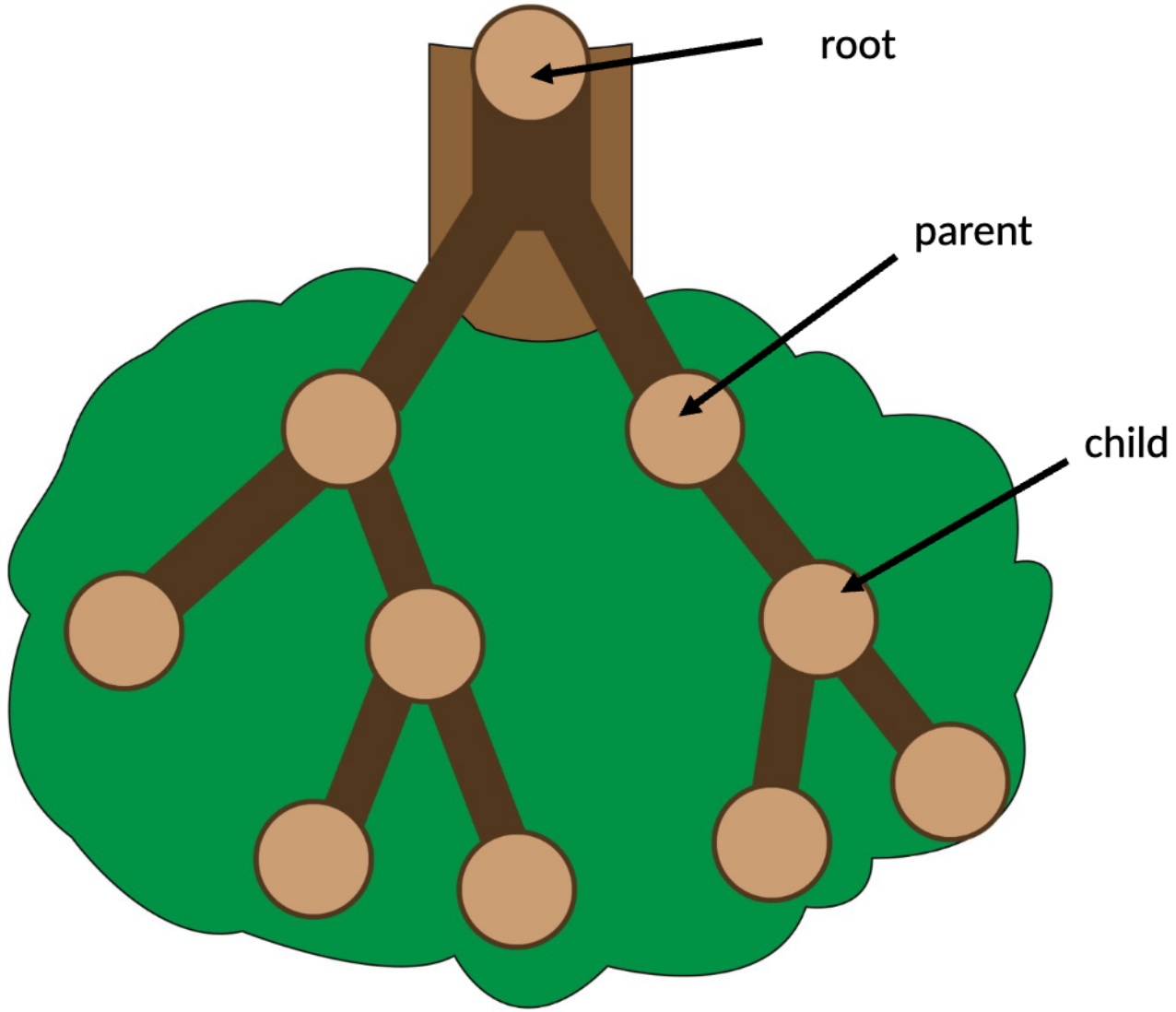
# Tree Data Structure



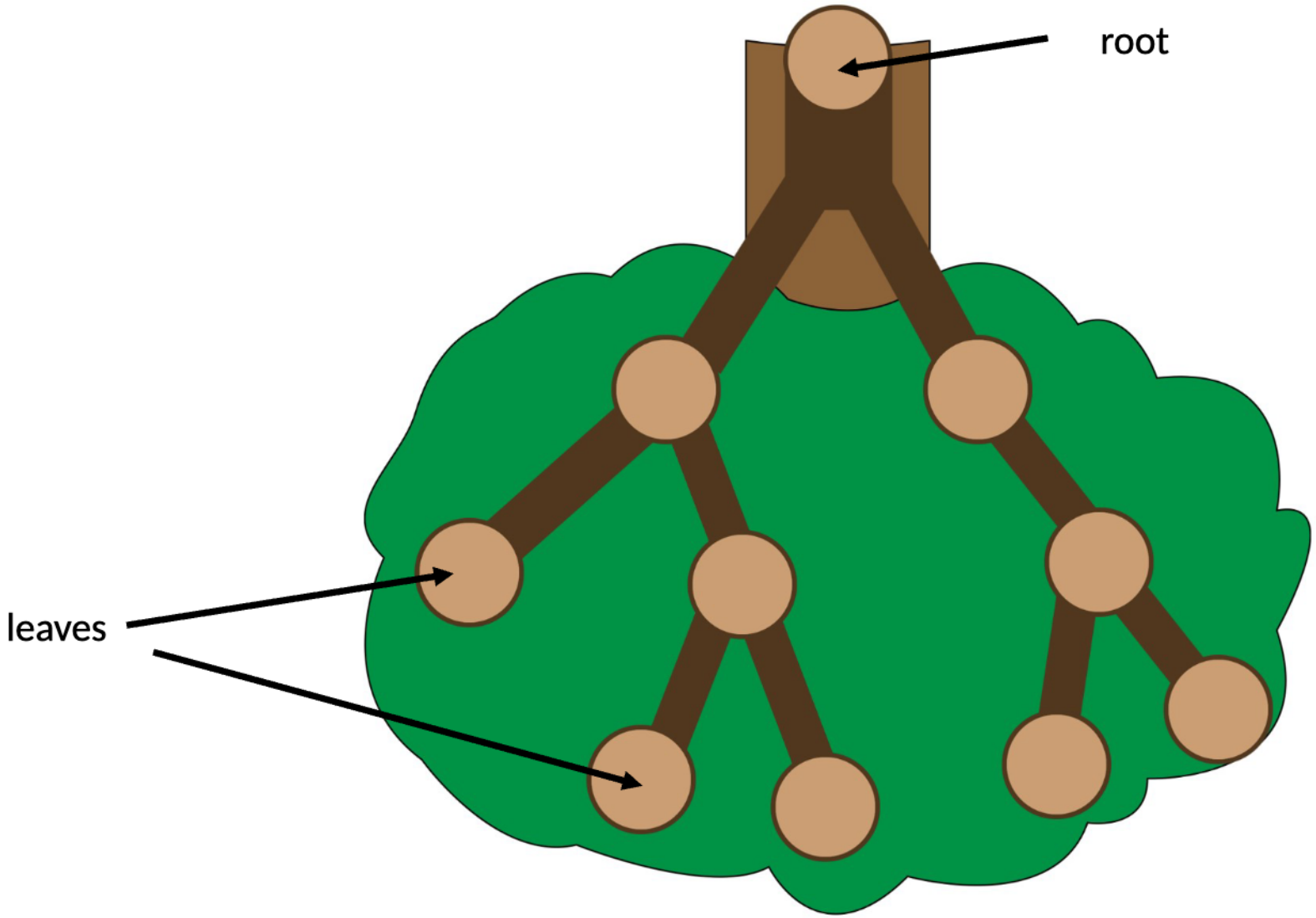
# Tree Data Structure



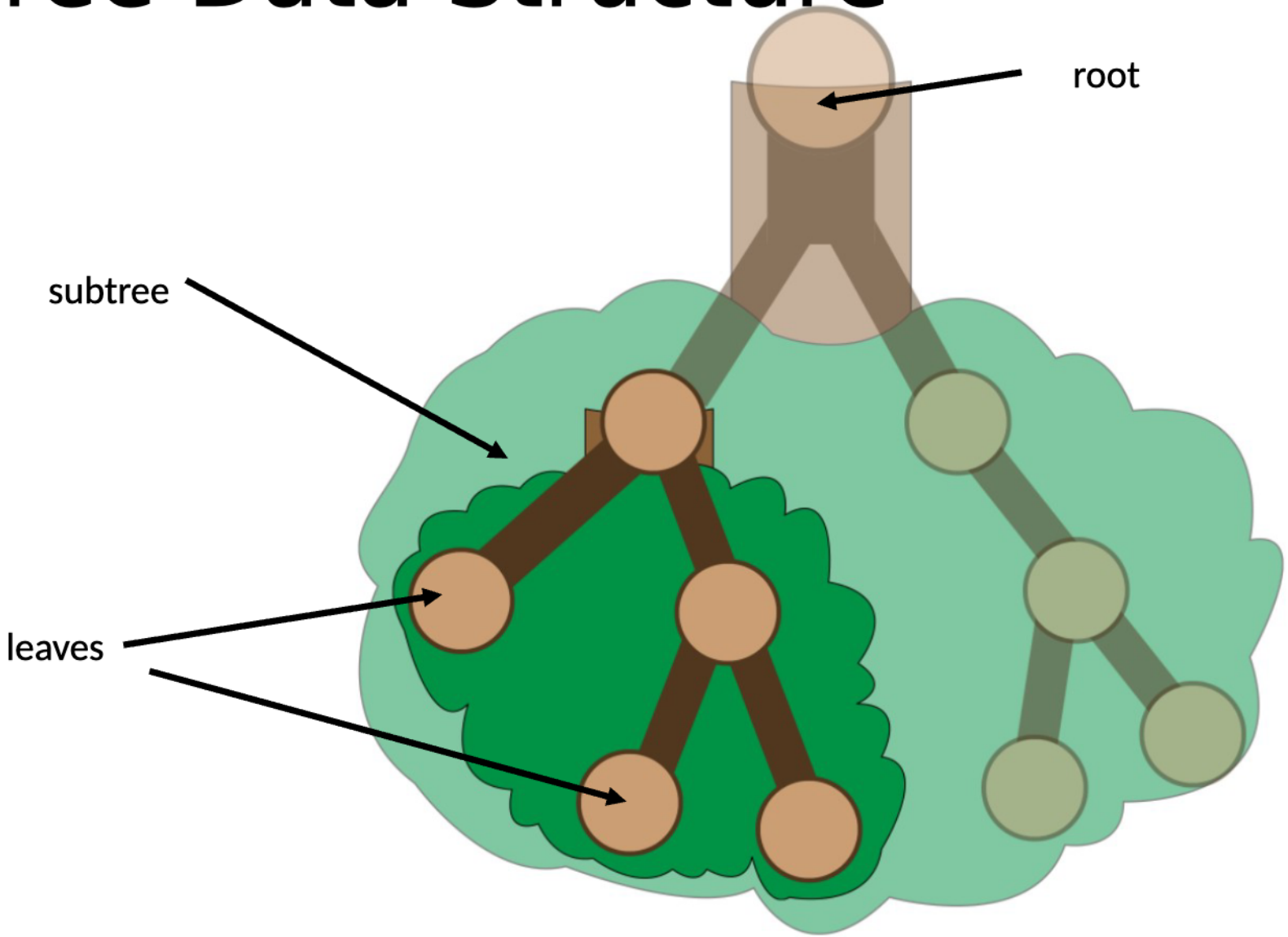
# Tree Data Structure



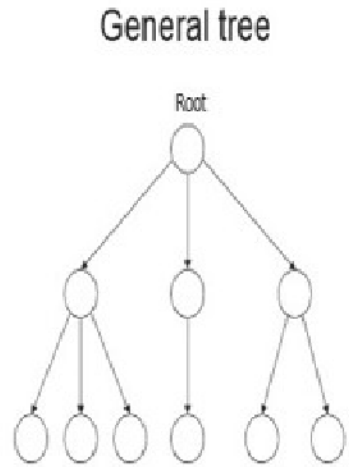
# Tree Data Structure



# Tree Data Structure



# Introduction to the `Tree` Data Structure




- A tree is a **non-linear** (or hierarchical) data structure
  - No notion of first, second…. element in the list
  - Elements in the tree have relationships such as `parent`, `child`, `ancestor` and `descendant`
- Trees can be used to improve the efficiency of collection classe they combine the advantages of both **ordered arrays** and **linked lists:**
  - Similarities to ordered list: search can be done in O(log n)
  - Similarities to linked list: insert and delete are fast and size can grow dynamically

# Slow Insertion/Deletion in an Ordered Array

- If the array is sorted, search can be done $O(\log n)$ using binary search.
- On the other hand, if you want to insert a new object into an ordered array:
  - You first need to find where the object will go,
  - And then move all the objects with greater keys up one space in the array to make room for it.
  - These multiple moves are time consuming, requiring, on the average, moving half the items - N/2 moves or O( n ) time
  - Deletion involves the same multi-move operation, and is thus equally slow
- If you're going to be doing a lot of insertions and deletions, an ordered array is a bad choice

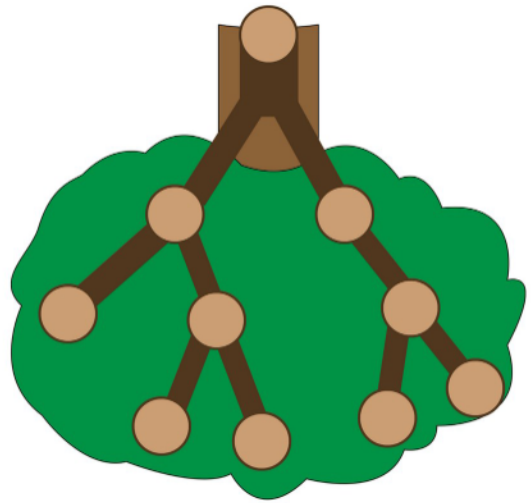
# Slow Search in a Linked List

- Insertions and deletions are quick to perform on a linked list
  - These operations are accomplished simply by changing a few references
  - These operations require `O(1)` time (the fastest Big-O time)
- Unfortunately, however, finding a specified element in a linked list is not so easy
  - Start at the beginning of the list and visit each element until you find the one you're looking for.
  - Visit an average of `n/2` objects, comparing each one's key with the desired value
  - This is slow, requiring $O(n)$ time.
  - Notice that $O(n)$ is considered fast for a sort but slow for data structure operations insert, delete or search

# Ordered Linked List?!!

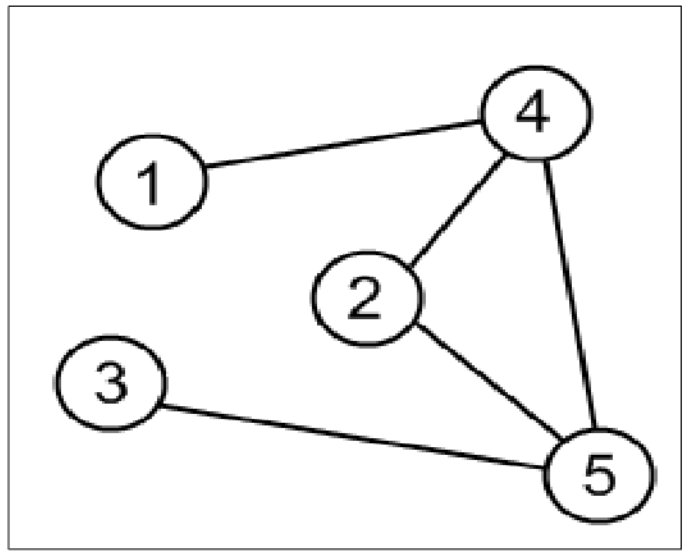
- You might think you could speed things up by using an ordered linked list in which the elements were arranged in order
- But this doesn't help! You still must start at the beginning and visit the elements in order because there's no way to access a given element without following the chain of references to it
- Of course, in an ordered list, it's faster to visit the nodes in order than it is in a non-ordered list, but that doesn't help to find an arbitrary object

# Trees to the Rescue

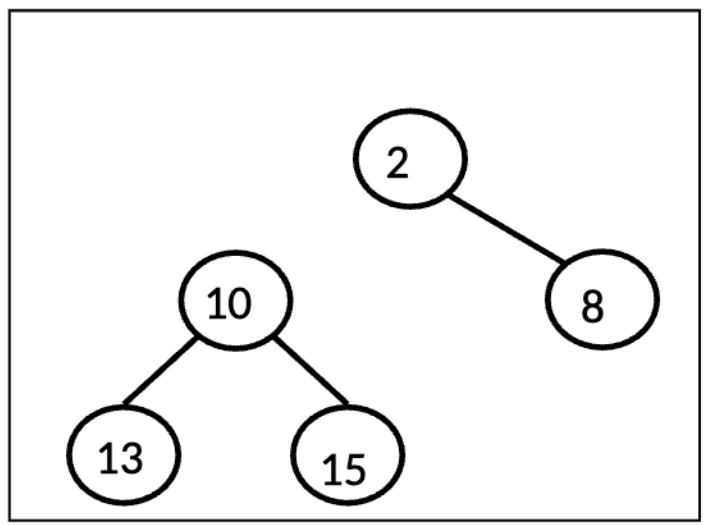


- A **tree** is a finite set of nodes that must follow these rules:
  - There is one special node called the **root**
  - Each node can be linked with one or more different nodes, called its **children**. If a node `c` is a child of another node `p`, then we say `p` is `c`'s **parent**
  - Each node except the root has exactly one parent, while the root has no parent
  - If you start at any node and move up to the node's parent, and keep on going up again to that node's parent and keep moving upward to each node's parent, you will eventually reach the root

# Not a Tree



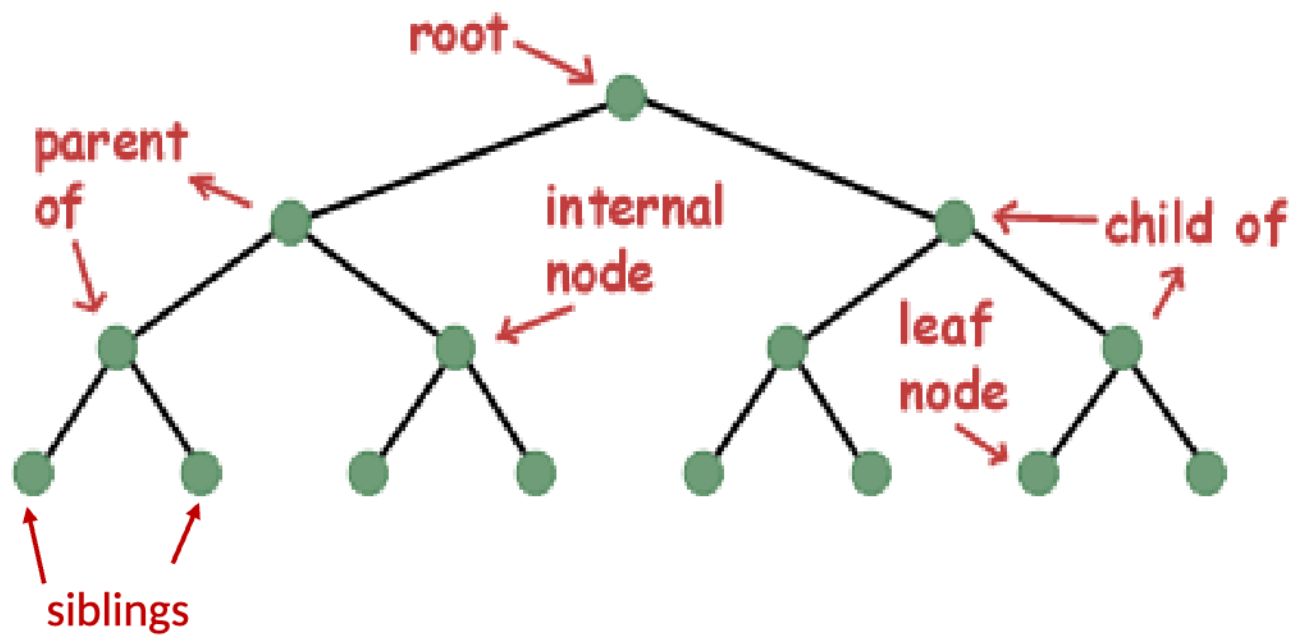

This is not a tree because there is a closed cycle between 2,4 and 5




This is not a tree because if you start at 15, you cannot reach the root

# Tree Terminology

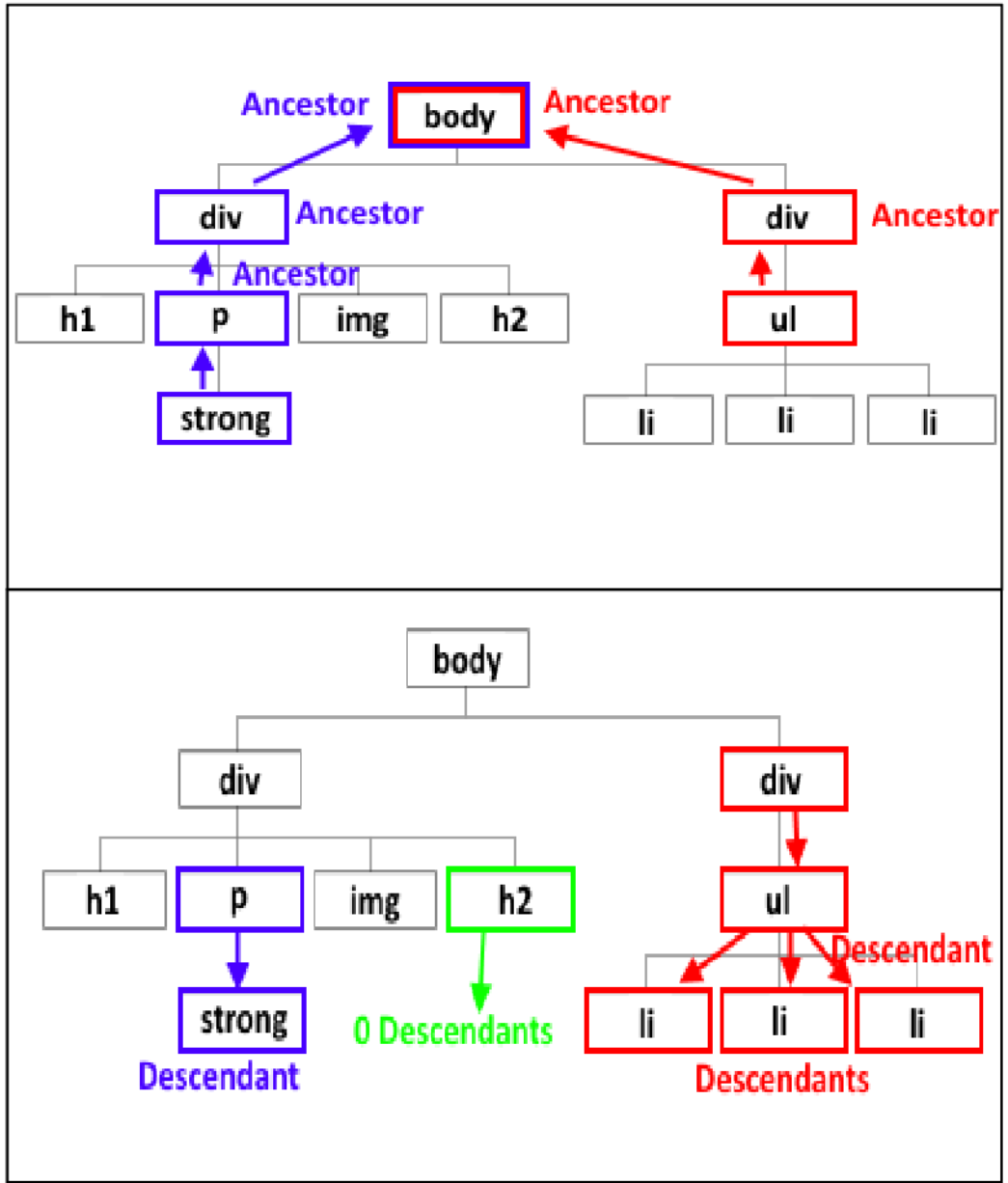
- node
- root
- internal node vs. leaf node
- child of
- parent
- siblings



# More Tree Terminology

- ancestors of a node, n:
  - All nodes in the path from n to the root

- descendants of a node, n:
  - All nodes that have n as an ancestor

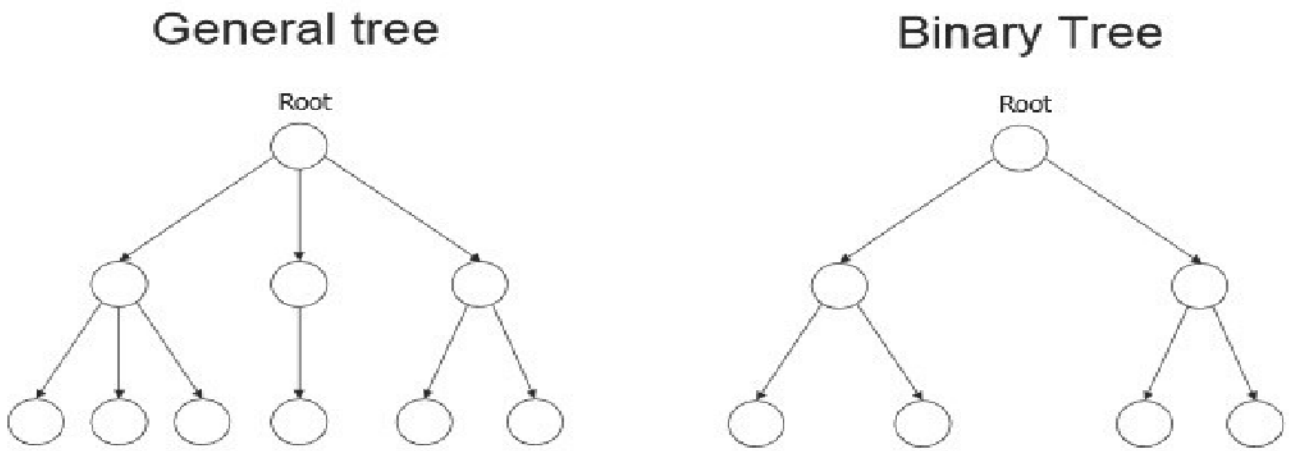


# Binary Trees

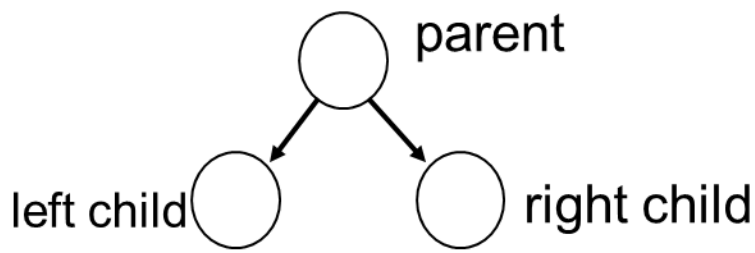
A specific type of tree

# Binary Trees

- A binary tree is a special type of tree where each node can have **two** children at most

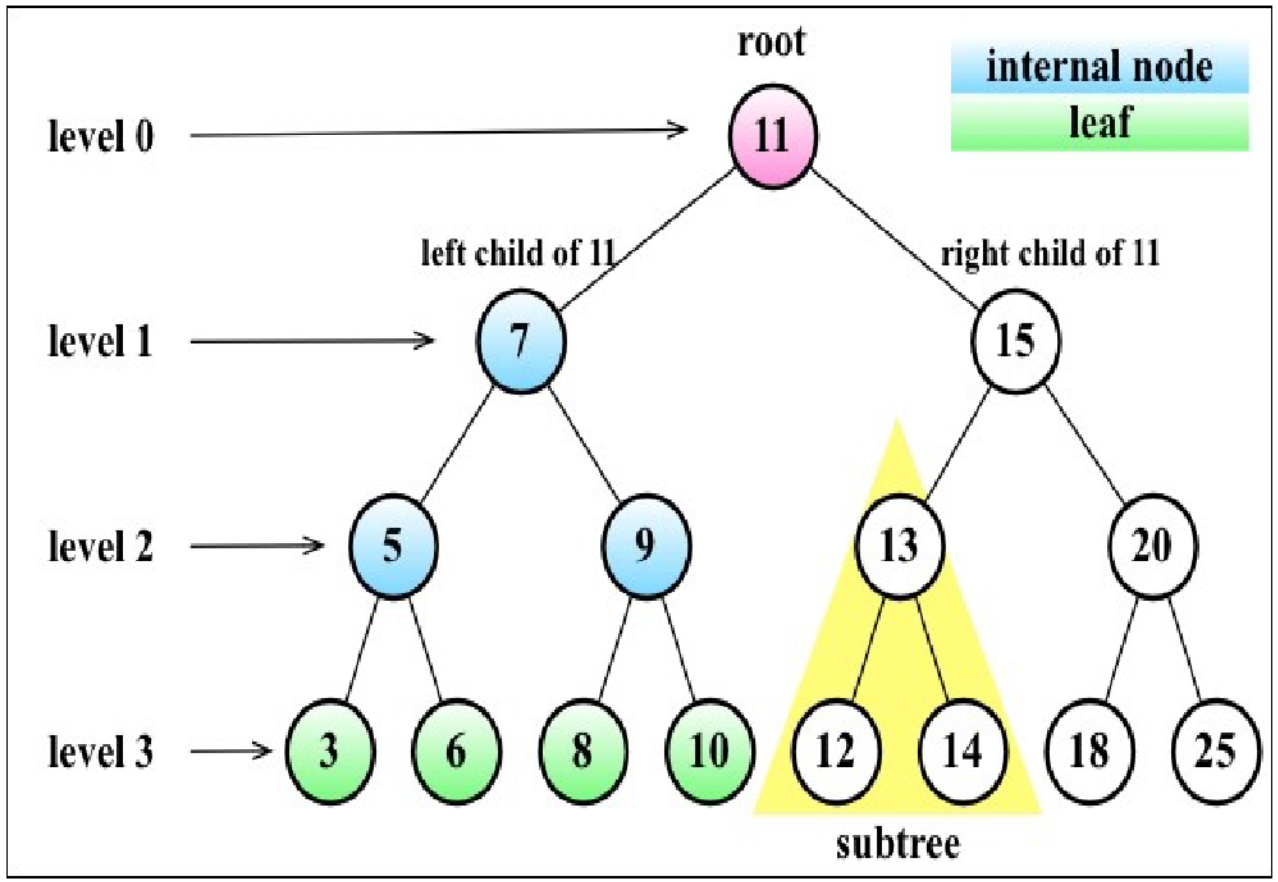



- Usually called leftChild and rightChild



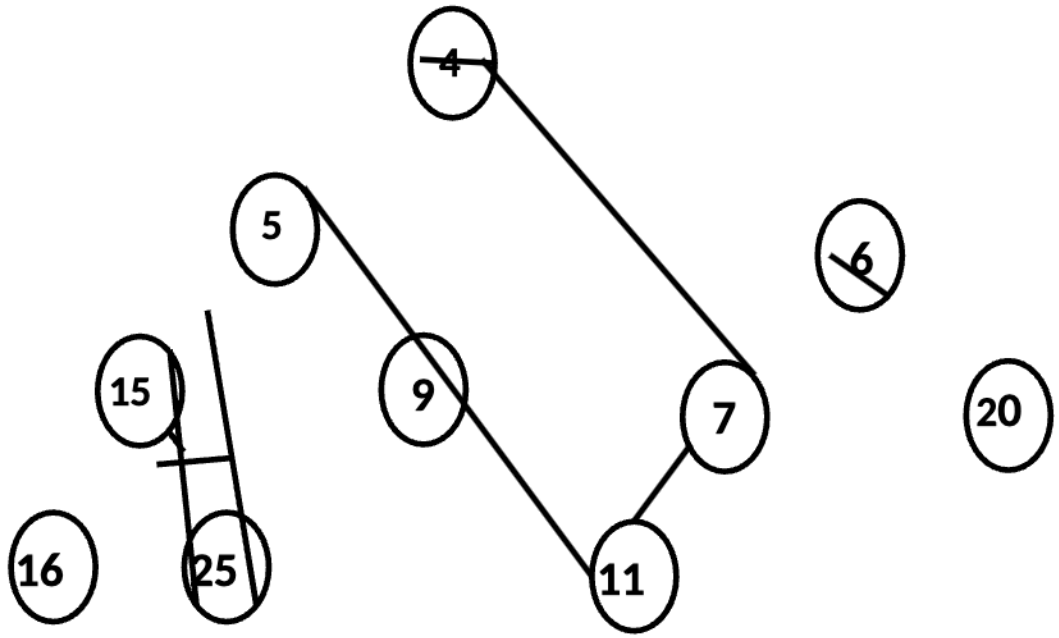
# Even More Tree Terminology

- left child vs. right child
- left subtree vs. right subtree
- level
  - root is at level 0
  - depth of node is the level at which the node exists
  - Depth (or height) of the tree is highest depth of its nodes



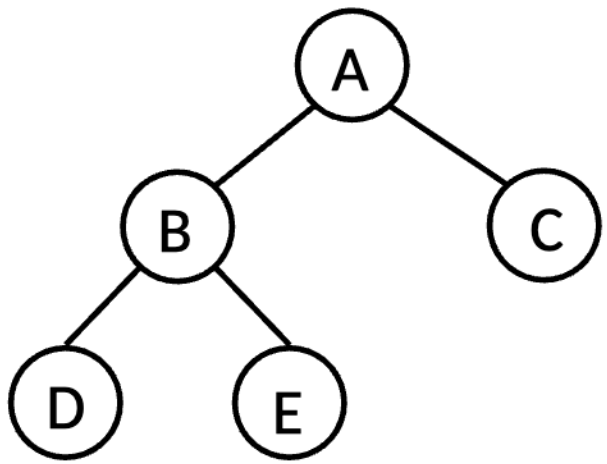
# Binary Tree Properties: `size`, `depth` and `height`

- The `size` of a binary tree is the number of nodes in it
  - For example, this tree tree has a `size` of 10
- The `depth` of a node is its distance from the root
  - For example:
    - 4 is at depth zero
    - 15 is at depth 2
- The `height` of a binary tree is the depth of its deepest node
  - For example, the `height` of this tree is 3



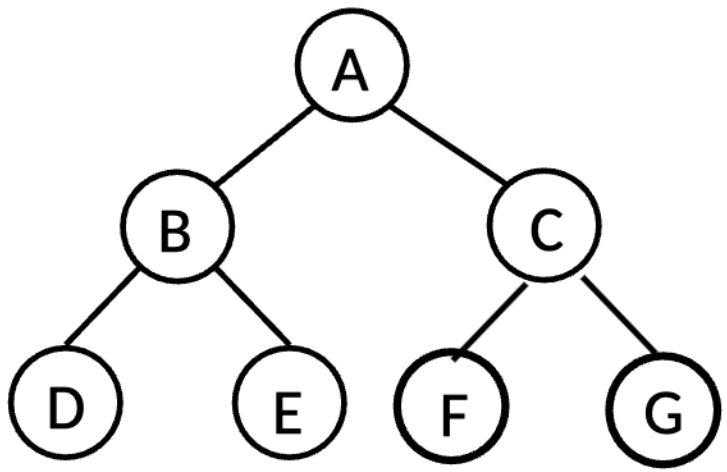
# Binary Tree Properties: `complete` Tree

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



# Binary Tree Properties: `full` Tree

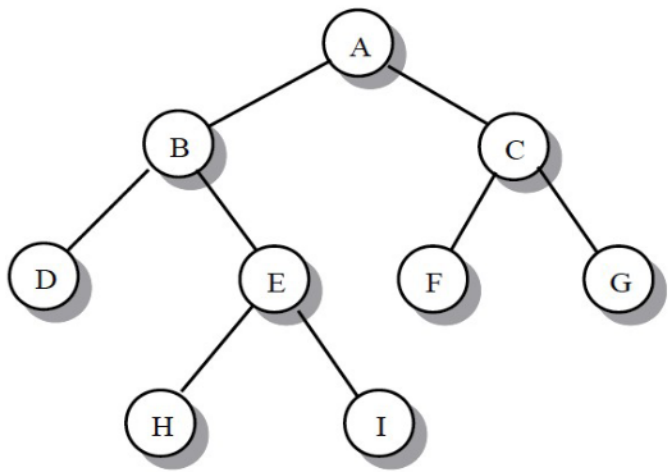
A **full binary tree** is a **binary tree** in which every level is completely filled



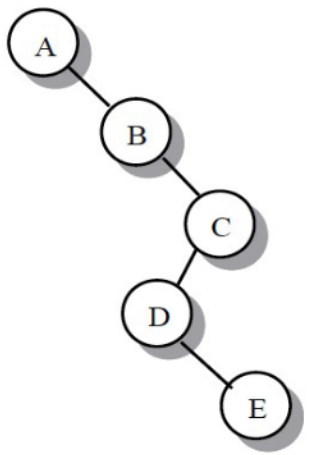
# Binary Tree Properties: `Balanced` vs. `Degenerate`

- A binary tree is balanced if every level (except the lowest level) is *almost* "full":
  - Which means level `i` contains $2^i$ nodes
- In most applications, a reasonably balanced binary tree is desirable.

`balanced` binary tree size 9, height 3




`unbalanced (degenerate)` binary tree with size 5, height 4

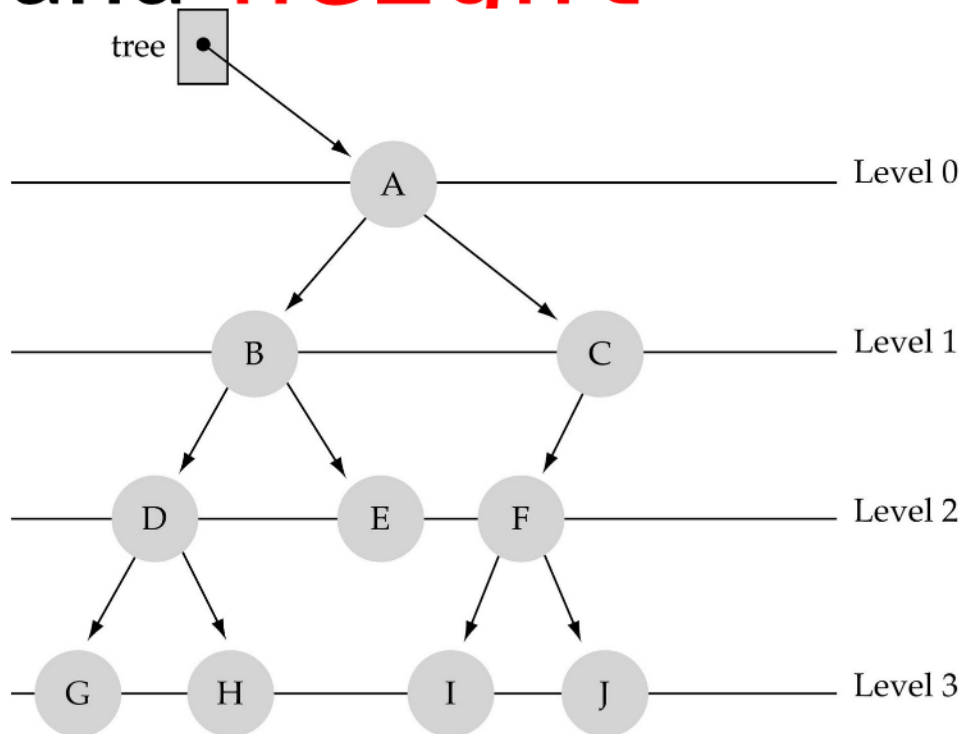


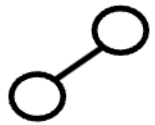
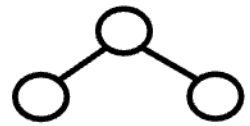
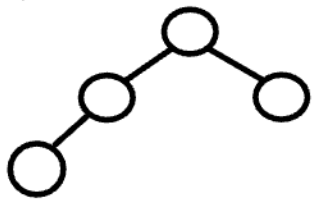
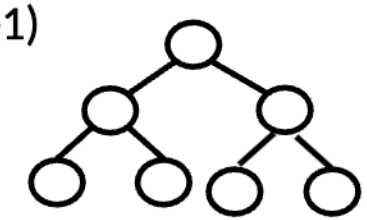
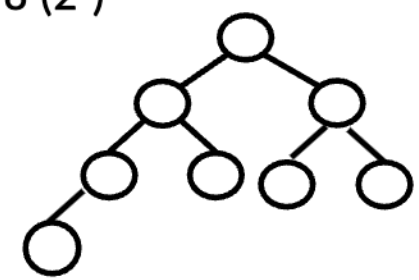
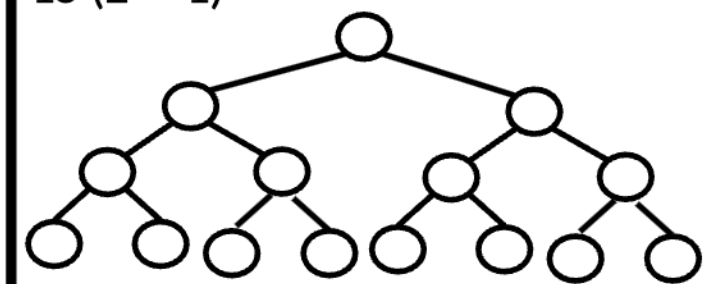
# Relationship Between `size` and `height`

- Maximum number of nodes at level `i` is $2^i$
- For a tree with `height h`:
  - Minimum number of nodes is `h+1`
    - One node at each level
  - Maximum number of nodes is $2^{h+1} - 1$:
    - Proof:
      - There are at most $2^i$ nodes in level `i`
      - Then the number of nodes in a tree with height `h` (levels are from `0` to `h`) is

$$2^0+2^1+2^2+2^3+....+2^h = 2^{h+1} - 1$$

- For a tree with `size n`:
  - Maximum `height is n-1`
    - One node at each level
  - Minimum `height is log(n+1)-1`



| Height | Minimum number of nodes in a complete tree | Maximum number of nodes in a complete tree (a full tree) |
|---|---|---|
| 0 | 1 ($2^0$)  | 1 ($2^{0+1}-1$)  |
| 1 | 2 ($2^1$)  | 3 ($2^{1+1}-1$)  |
| 2 | 4 ($2^2$)  | 7 ($2^{2+1}-1$)  |
| 3 | 8 ($2^3$)  | 15 ($2^{3+1}-1$)  |