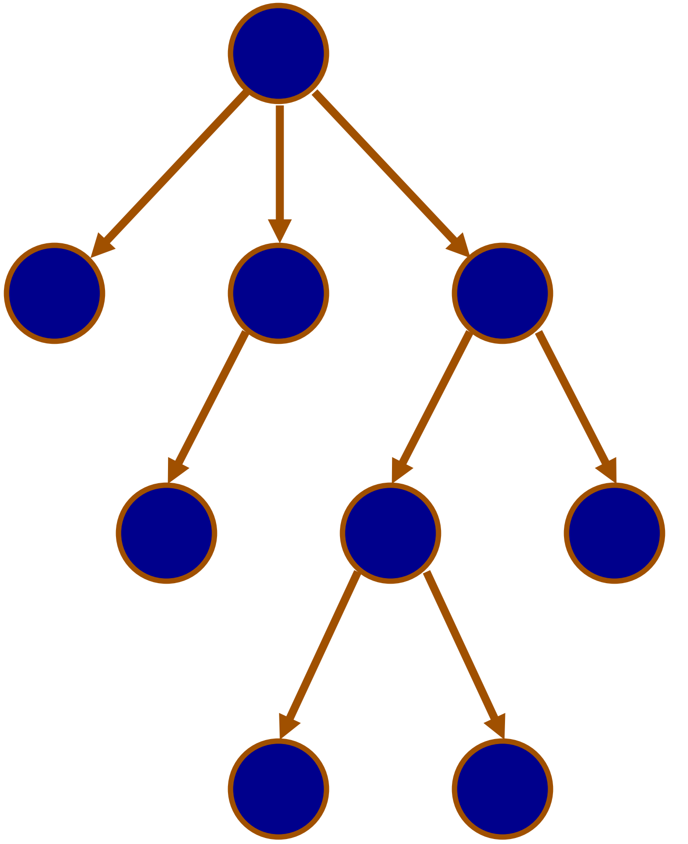
CS 261: Data Structures

Trees

Trees

- Ubiquitous – they are everywhere in CS

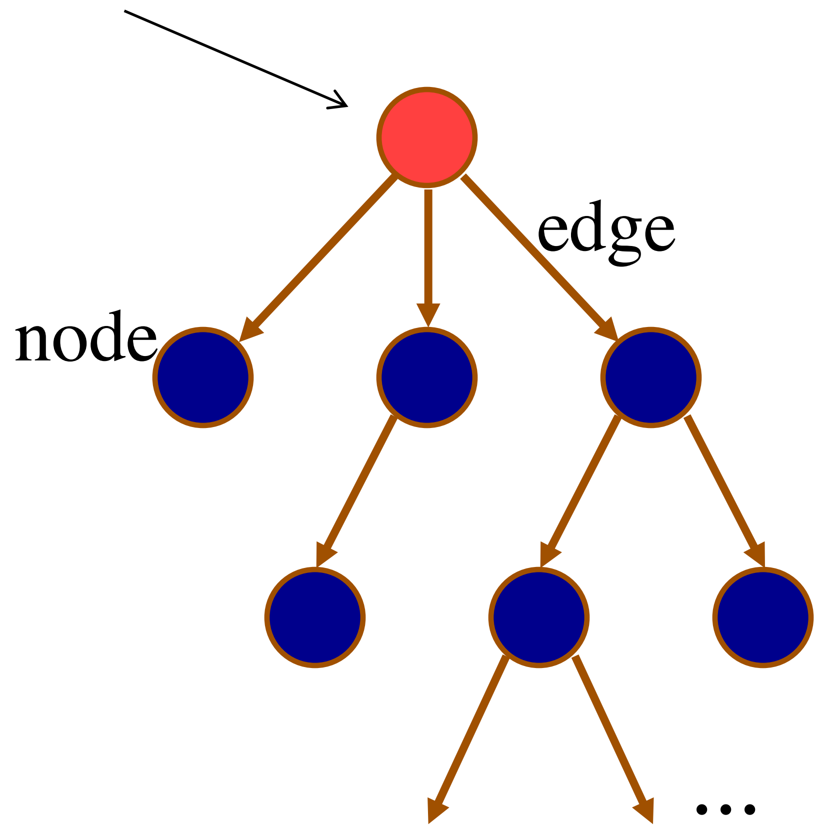
- Probably ranks third among the most used data structure:
  1. Vectors and Arrays
  2. Lists
  3. Trees



Tree Terminology

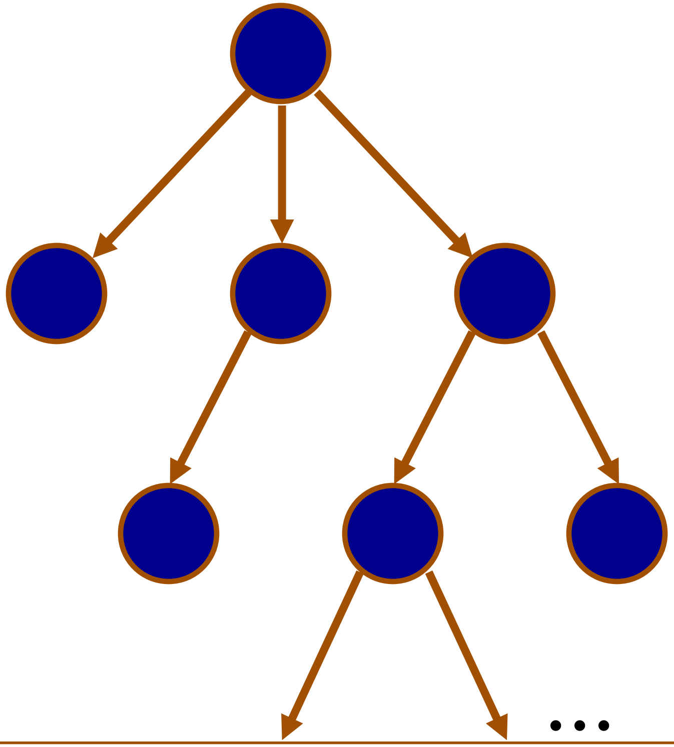
- Tree = Set of **nodes** connected by **arcs** (or **edges**)

- A directed tree has a single **root** node

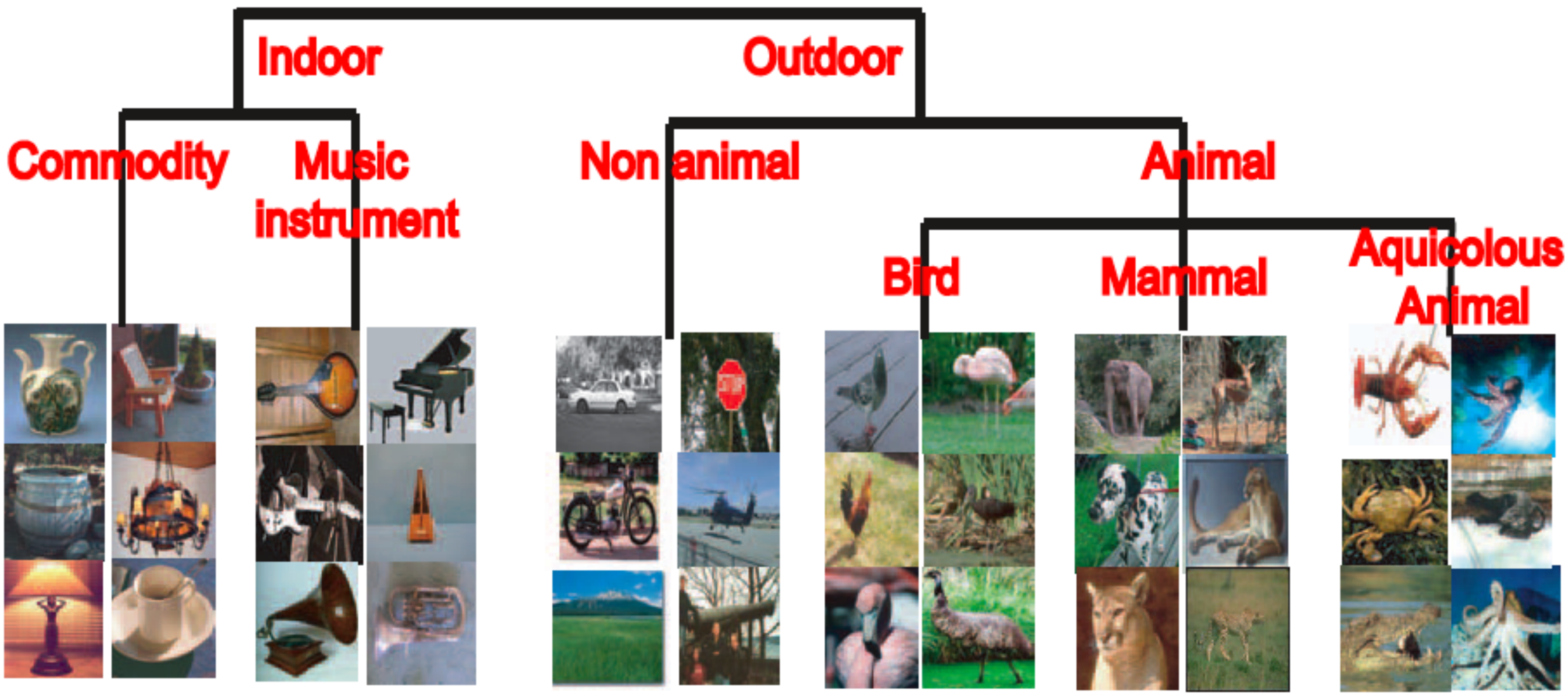


Tree Terminology

- A **parent** node points to (one or more) **children** nodes



Example: Object Taxonomy



Tree Characteristics

- Every node (except the root) has exactly one parent
- Nodes with no children are **leaf** nodes
- Nodes with children are **interior** nodes

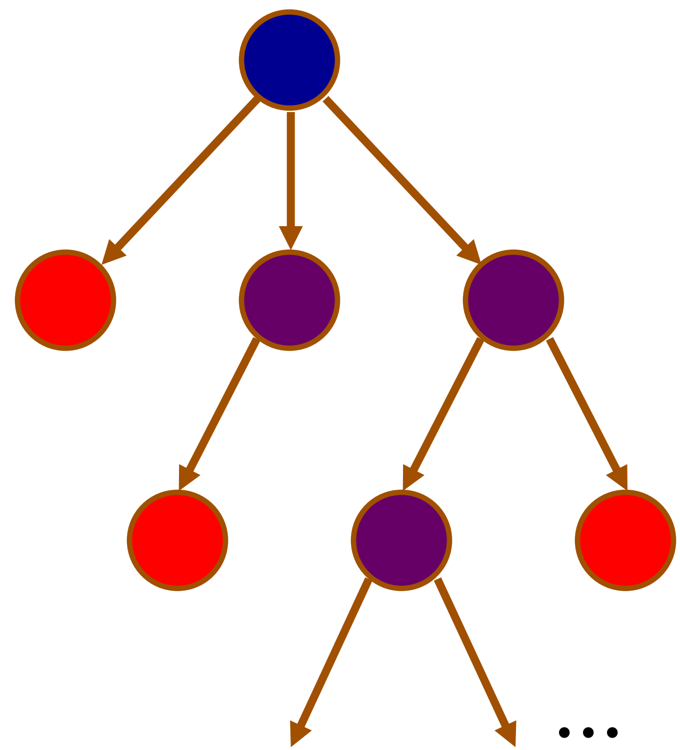
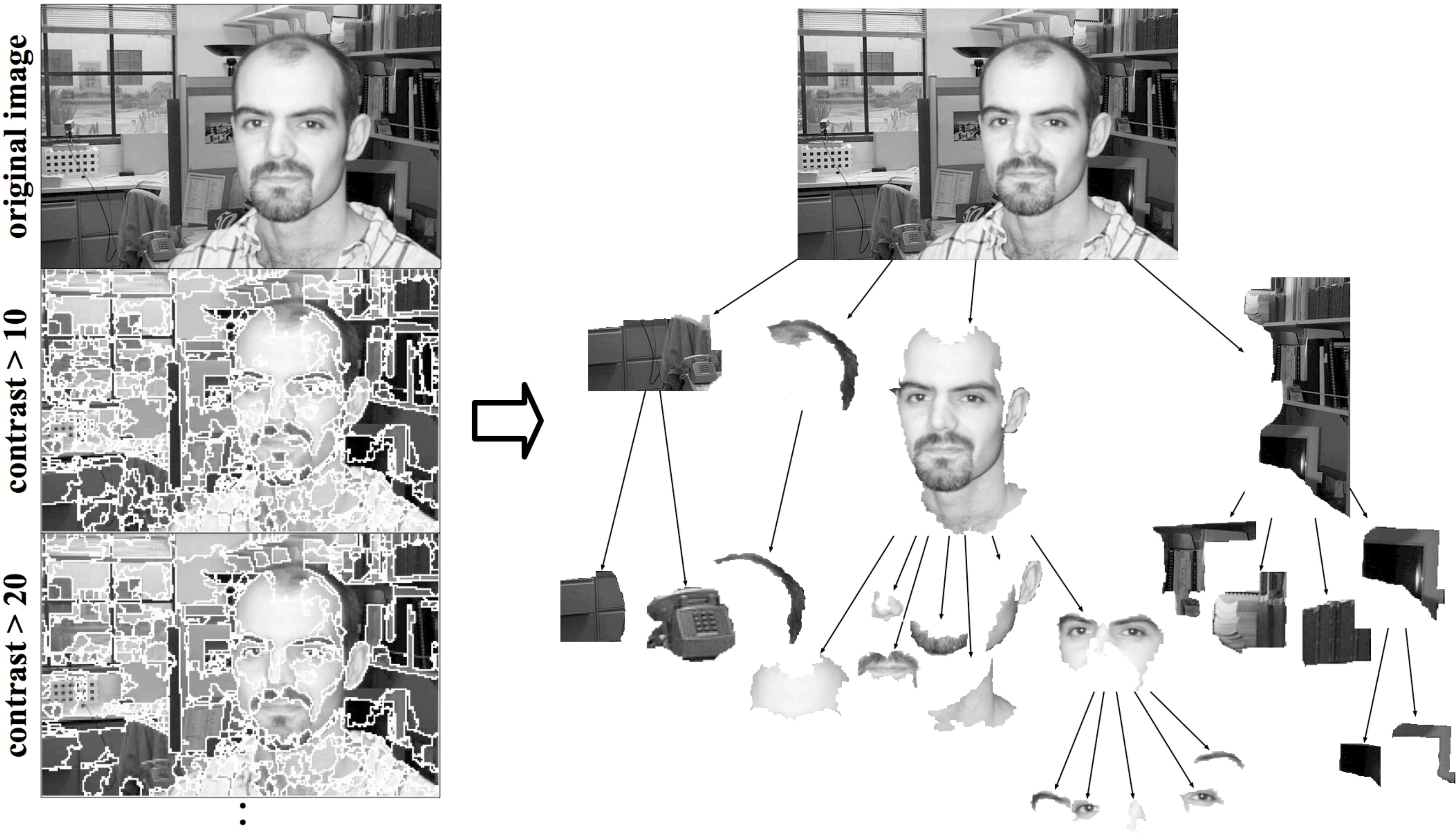


Image Representation = Segmentation Tree



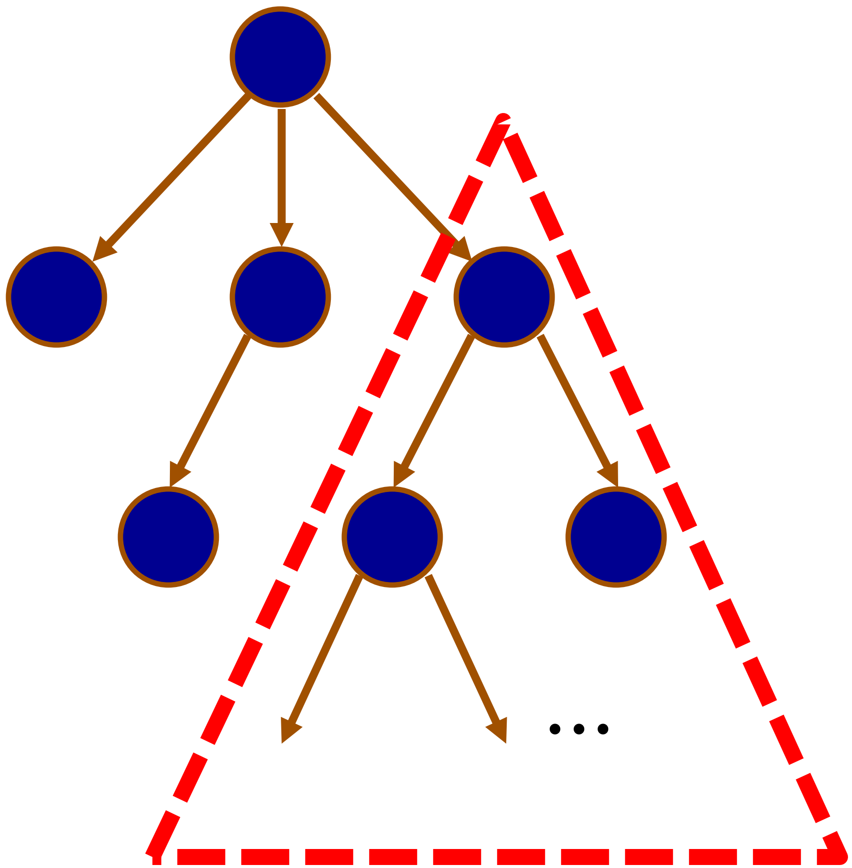
Tree Terminology

- **Descendants** of a node include children, and their children, and so on until the leaves.

- All nodes in a tree are descendants of the root (except for the root)

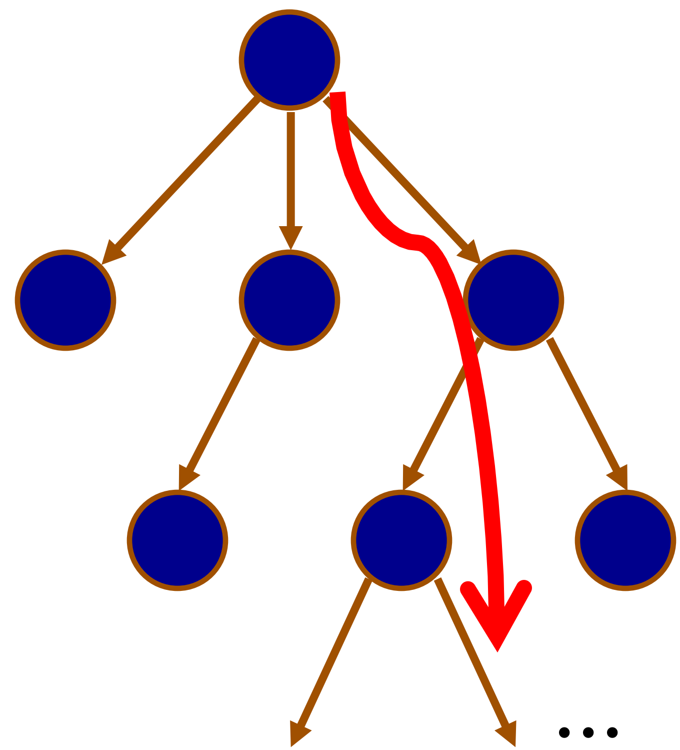
Tree Terminology

- An internal node is the root of a **subtree**

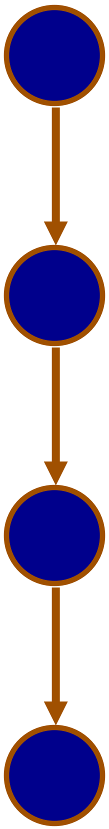


Tree Terminology

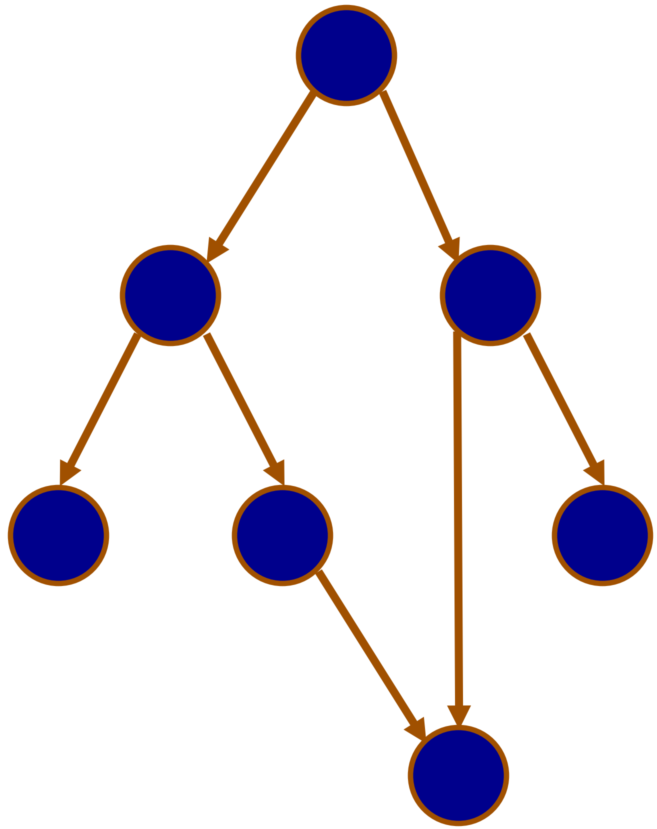
- There is a single, **unique path** from the root to any node
- A path's **length** is equal to the number of edges traversed



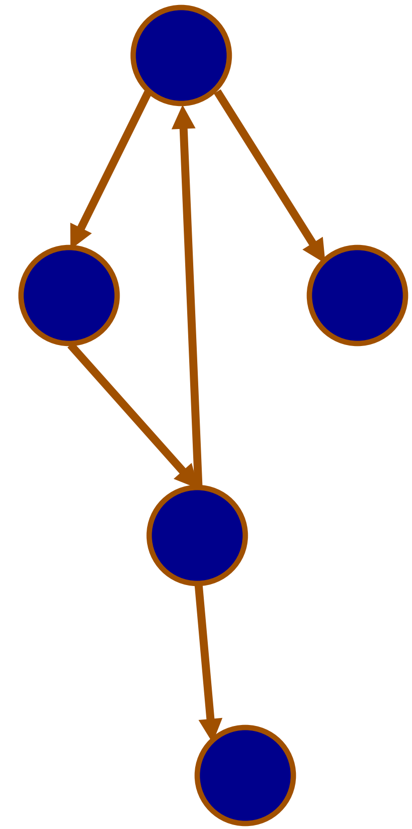
Are these trees?



Yes



No



No

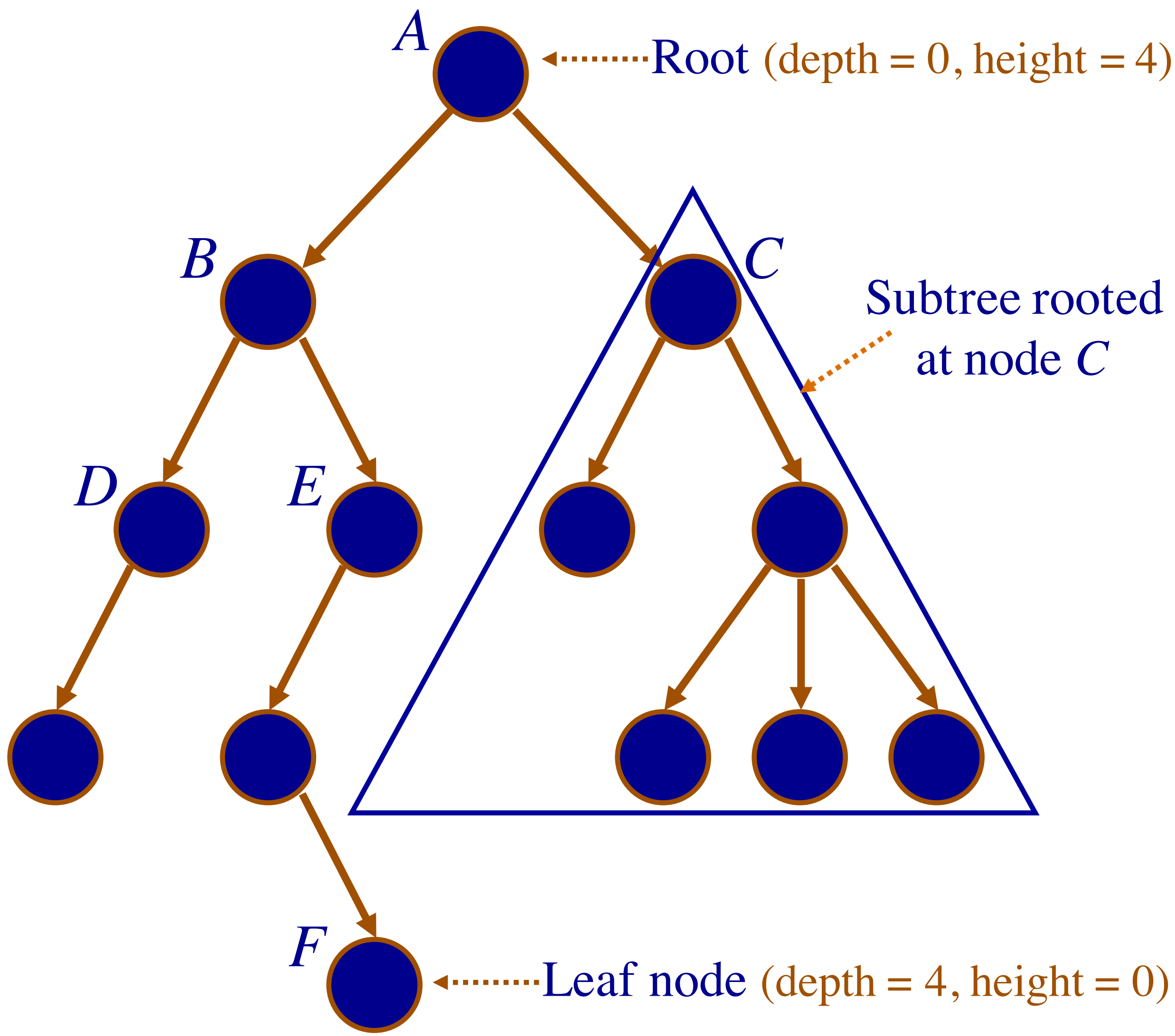
Tree Terminology

- **Height** of a node = Path length from that node to the farthest leaf
  - Height of a leaf node = 0
  - Height of the tree = Height of the root

- **Depth** of a node = Path length from the root to that node
  - Depth of the root = 0
  - Depth of the tree = Maximum depth of all its leaves
  - Depth of the tree = Height of the tree

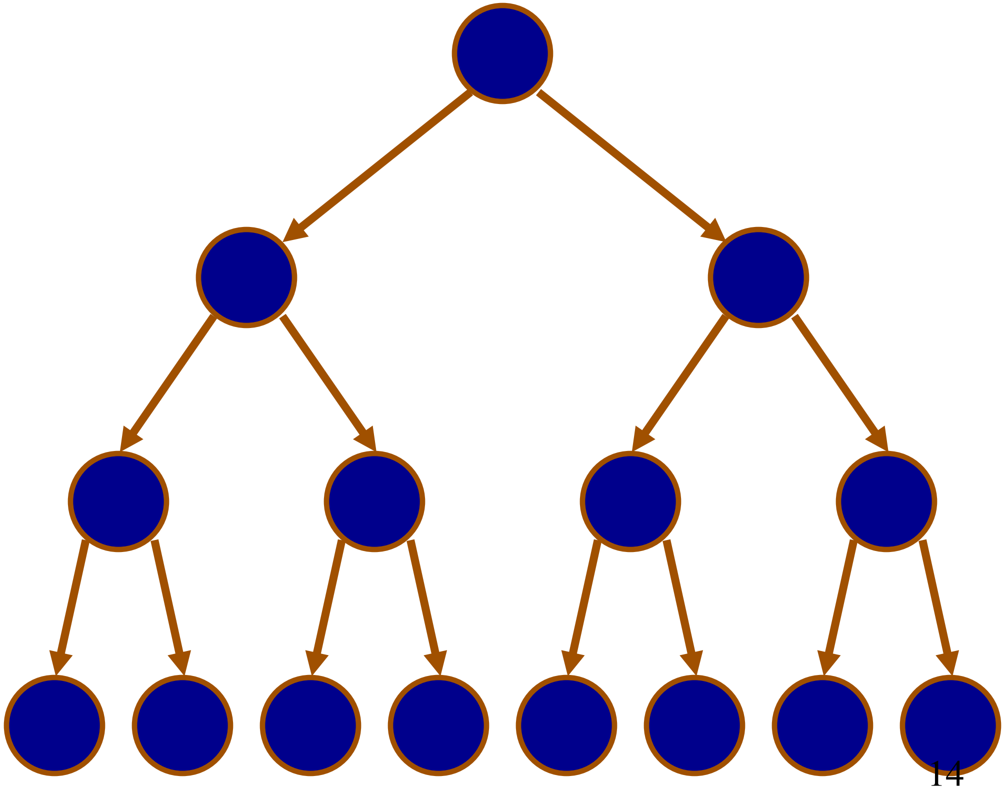
Example

- Nodes *D* and *E* are children of node *B*
- Node *B* is the parent of nodes *D* and *E*
- Nodes *B*, *D*, and *E* are descendents of node *A* (as are all other nodes in the tree…except *A*)
- *E* is an interior node
- *F* is a leaf node



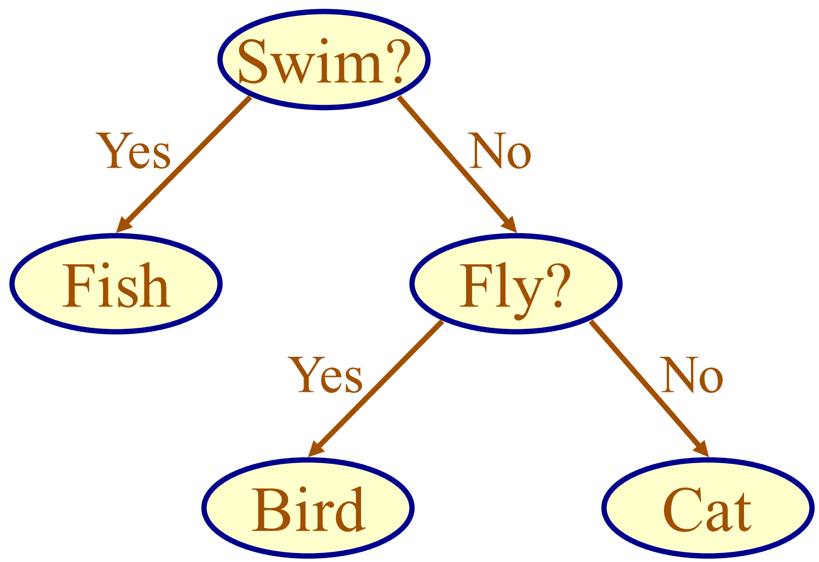
Binary Tree

- Internal nodes have no more than two children:
  - Children are referred to as “left” and “right”
  - a node may have only one child



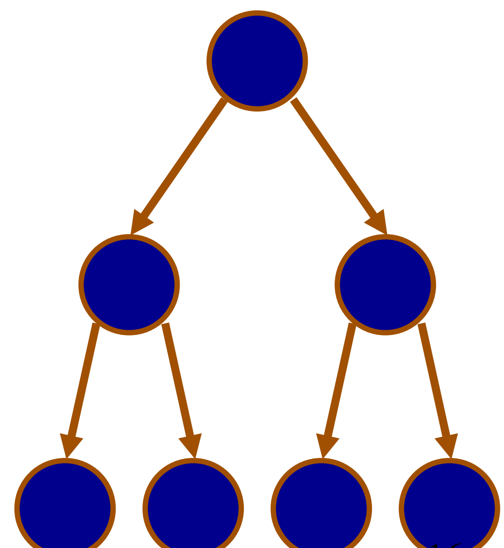
Example Application: Animal Game

Guess the animal!



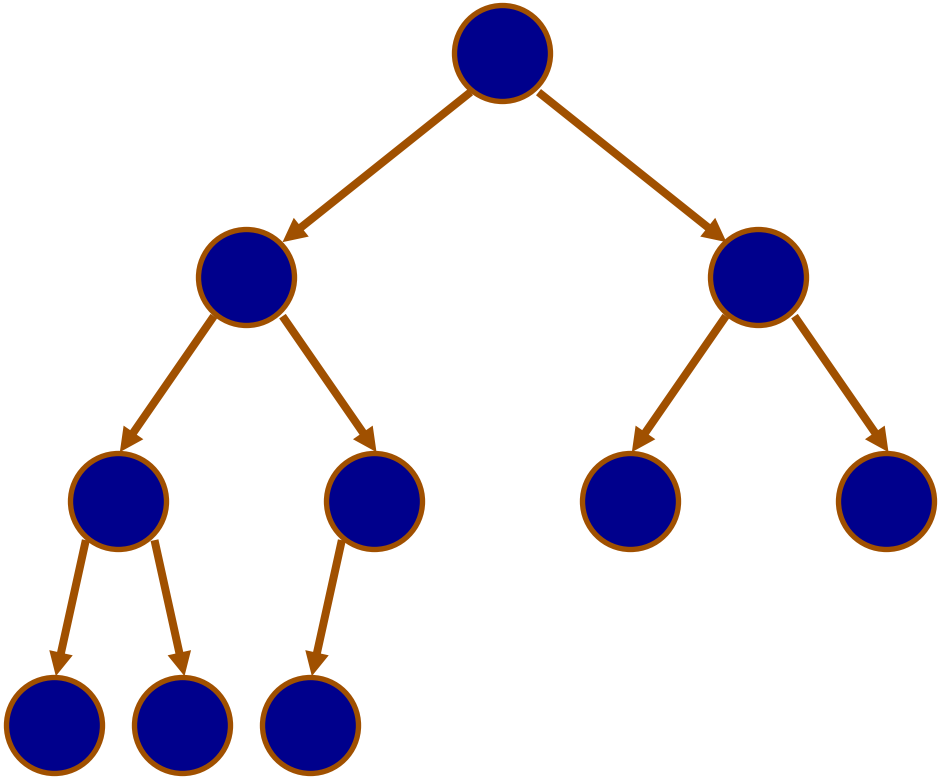
Binary Tree

- Nodes have no more than two children:
  - Children are generally referred to as “left” and “right”
- Full Binary Tree:
  - every leaf is at the same depth
  - Every internal node has **2** children
  - Depth of ***d*** will have $\mathbf{2^{d+1} - 1}$ nodes
  - Depth of ***d*** will have $\mathbf{2^{d}}$ leaves



Complete Binary Tree

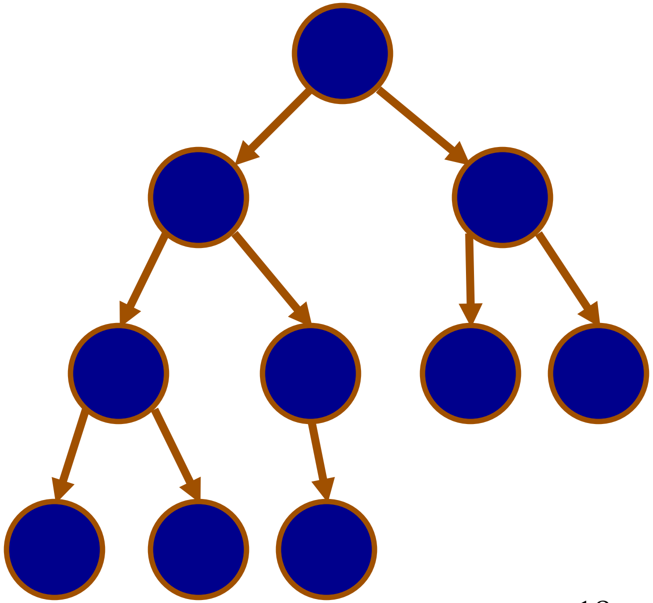
= Full binary tree, except for the bottom level which is filled from left to right



Complete Binary Tree

- What is the height of a complete binary tree that has ***n*** nodes?

- This is necessary for estimating time complexity, which is proportional to the path length



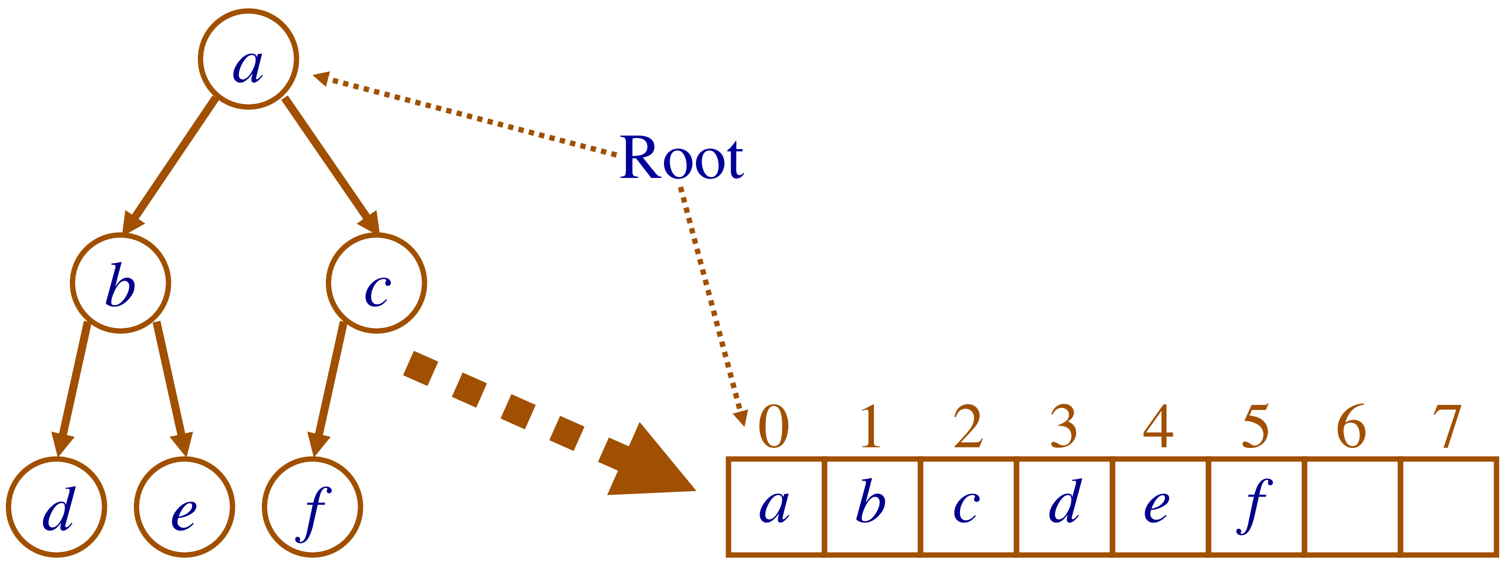
Dynamic Memory Implementation

```
struct Node {
   TYPE val;
   struct Node *left;        /* Left   child. */
   struct Node *right;       /* Right child. */
};
```

Like the **`Link`** structure in a linked list

Dynamic Array Implementation

Complete binary tree can be implemented using Dynamic Arrays in C

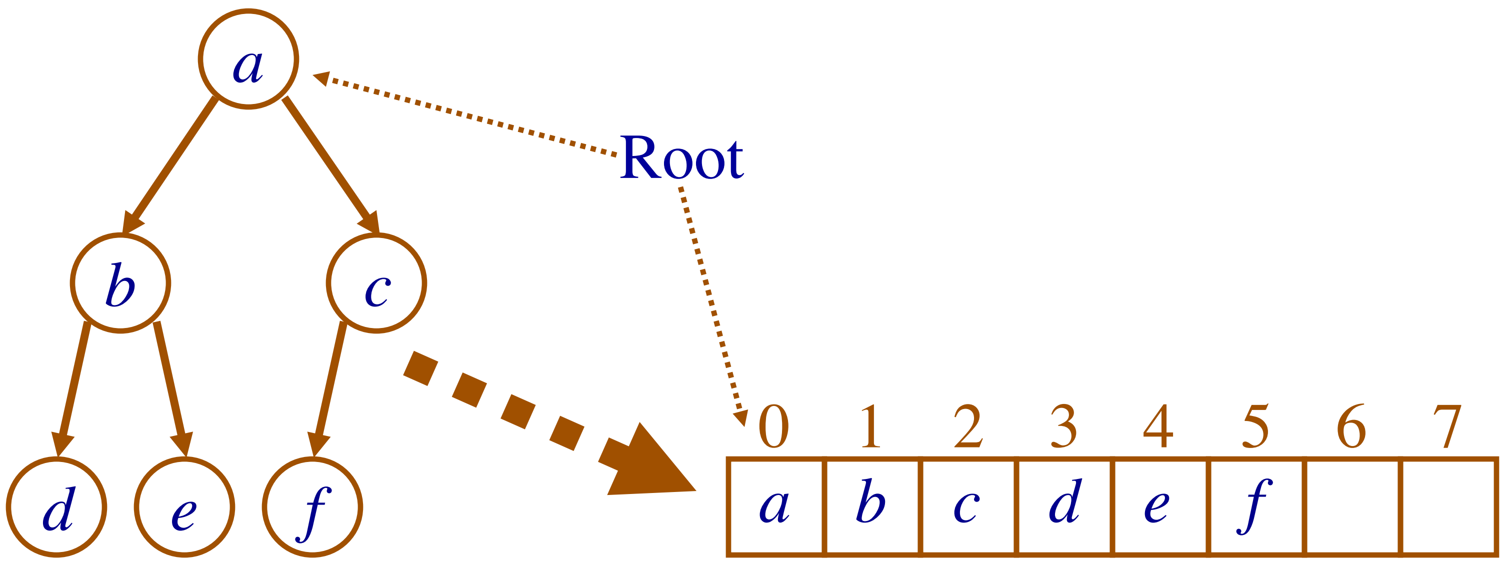


Children of node $\boldsymbol{i}$ are stored at locations

$$\mathbf{2}\boldsymbol{i}+\mathbf{1} \quad \text{and} \quad \mathbf{2}\boldsymbol{i}+\mathbf{2}$$

Dynamic Array Implementation

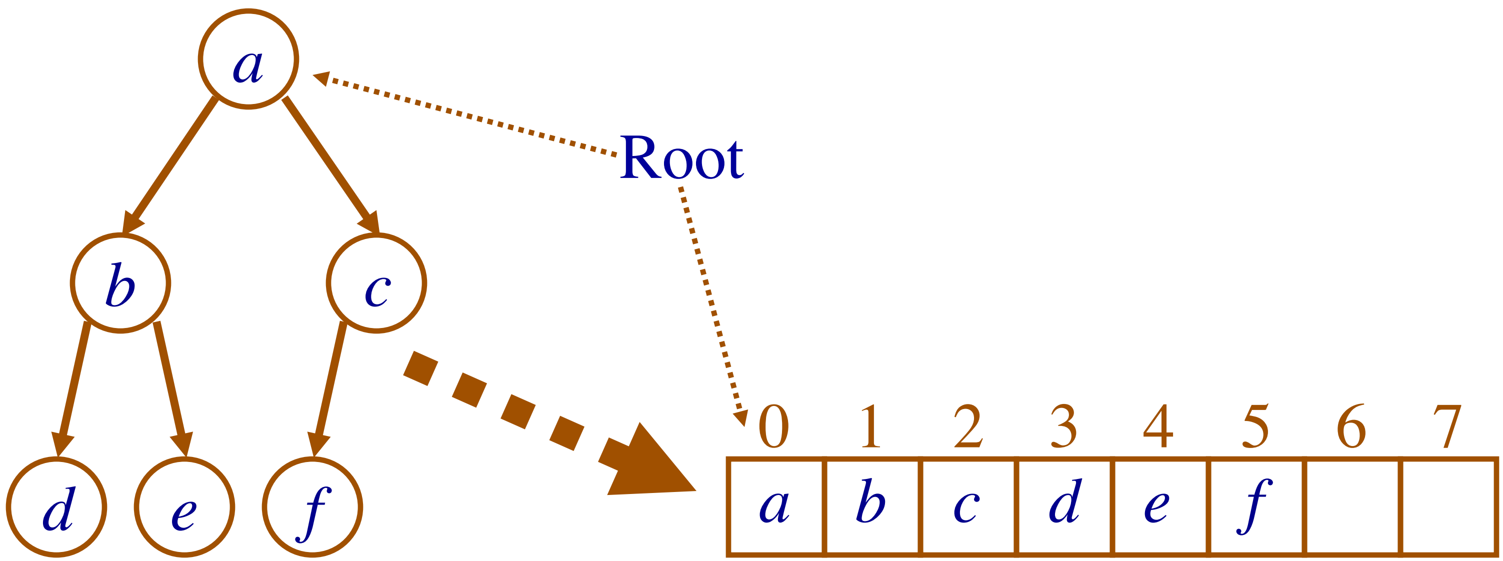
Complete binary tree can be implemented using Dynamic Arrays in C



Parent of node i is at **floor((*i* - 1) / 2)**

Dynamic Array Implementation

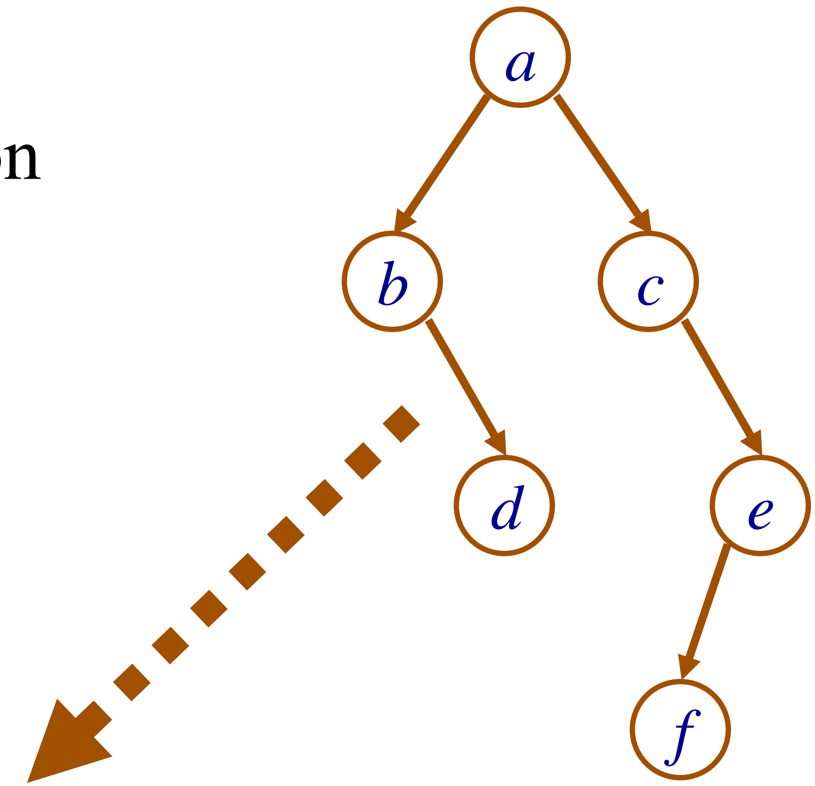
Incomplete binary trees?

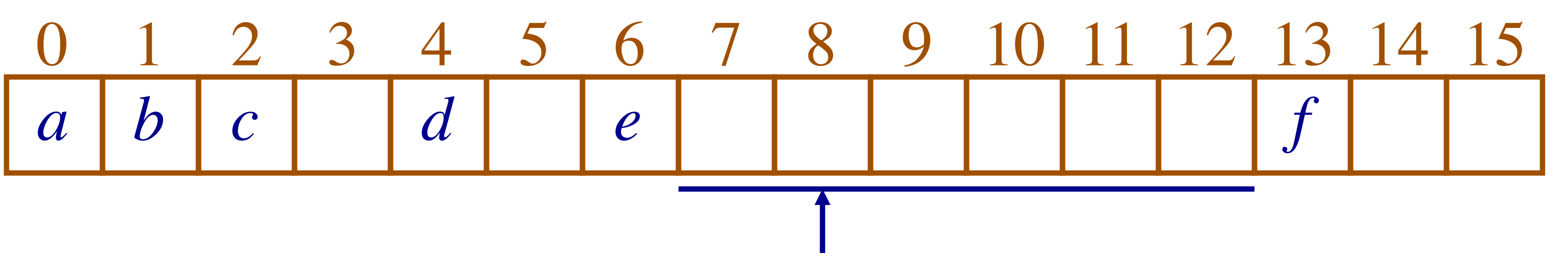


Why is this a bad idea if the tree is not complete?

Dynamic Array Implementation (cont.)

If the tree is not complete, a Dynamic Array implementation will be full of "holes"





Big gaps where a tree level is not filled!