CS261 Data Structures

Trees

Introduction and Applications

Goals

- Tree Terminology and Definitions
- Tree Representation
- Tree Application

Trees

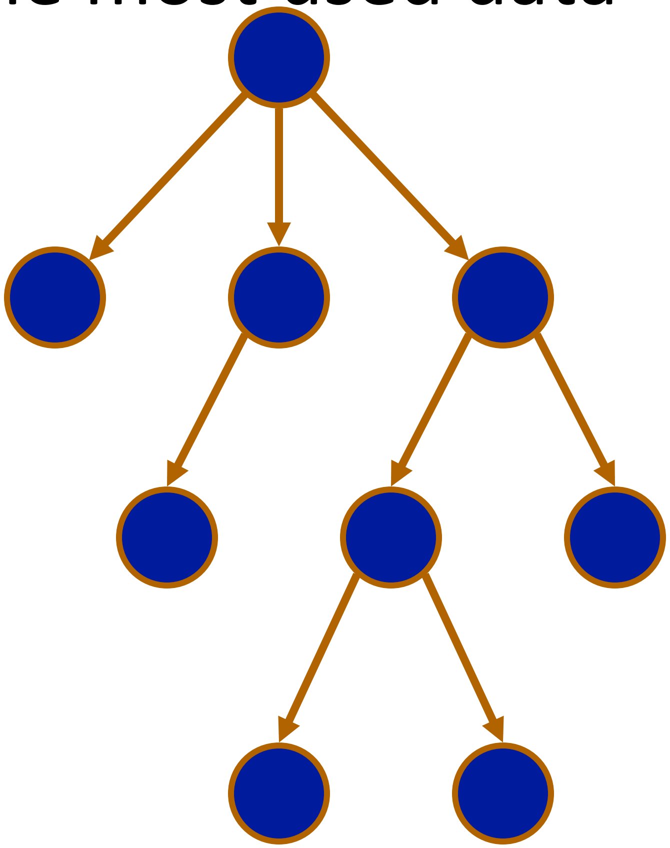
- Ubiquitous – they are everywhere in CS

- Probably ranks third among the most used data structure:

  1. Arrays/Vectors

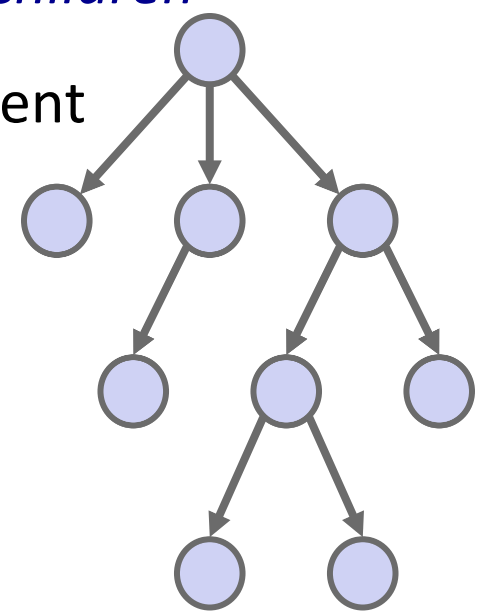
  2. Lists

  3. Trees



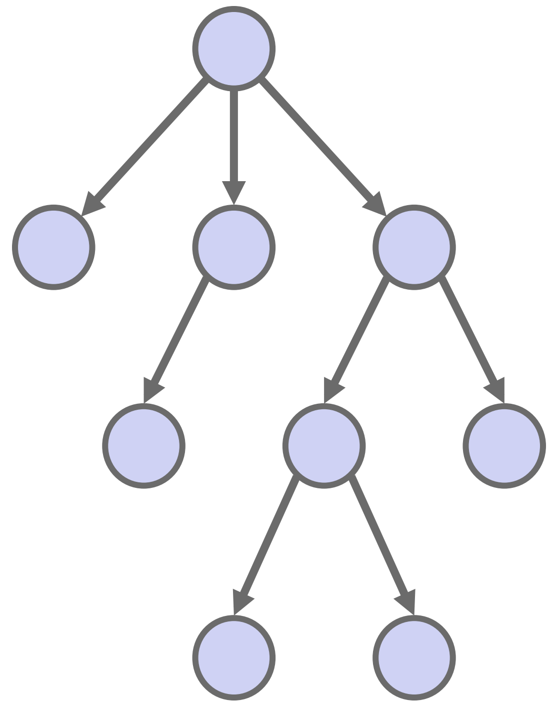
Tree Characteristics

- A tree consists of a collection of nodes connected by directed arcs
- A tree has a single *root* node
  - By convention, the root node is usually drawn at the top
- A node that points to (one or more) other nodes is the *parent* of those nodes while the nodes pointed to are the *children*
- Every node (except the root) has exactly one parent
- Nodes with no children are *leaf* nodes
- Nodes with children are *interior* nodes

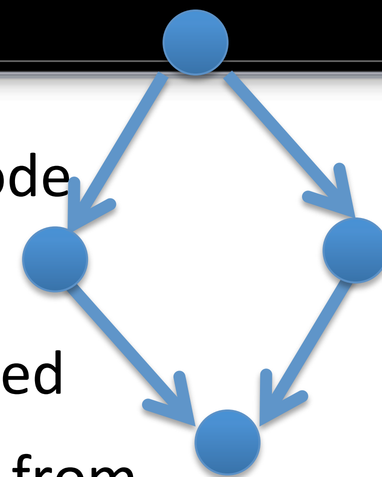


Tree Characteristics (cont…)

- Nodes that have the same parent are *siblings*
- The *descendents* of a node consist of its children, and their children, and so on
  - All nodes in a tree are descendents of the root node (except, of course, the root node itself)
- Any node can be considered the root of a *subtree*
- A subtree rooted at a node consists of that node and all of its descendents



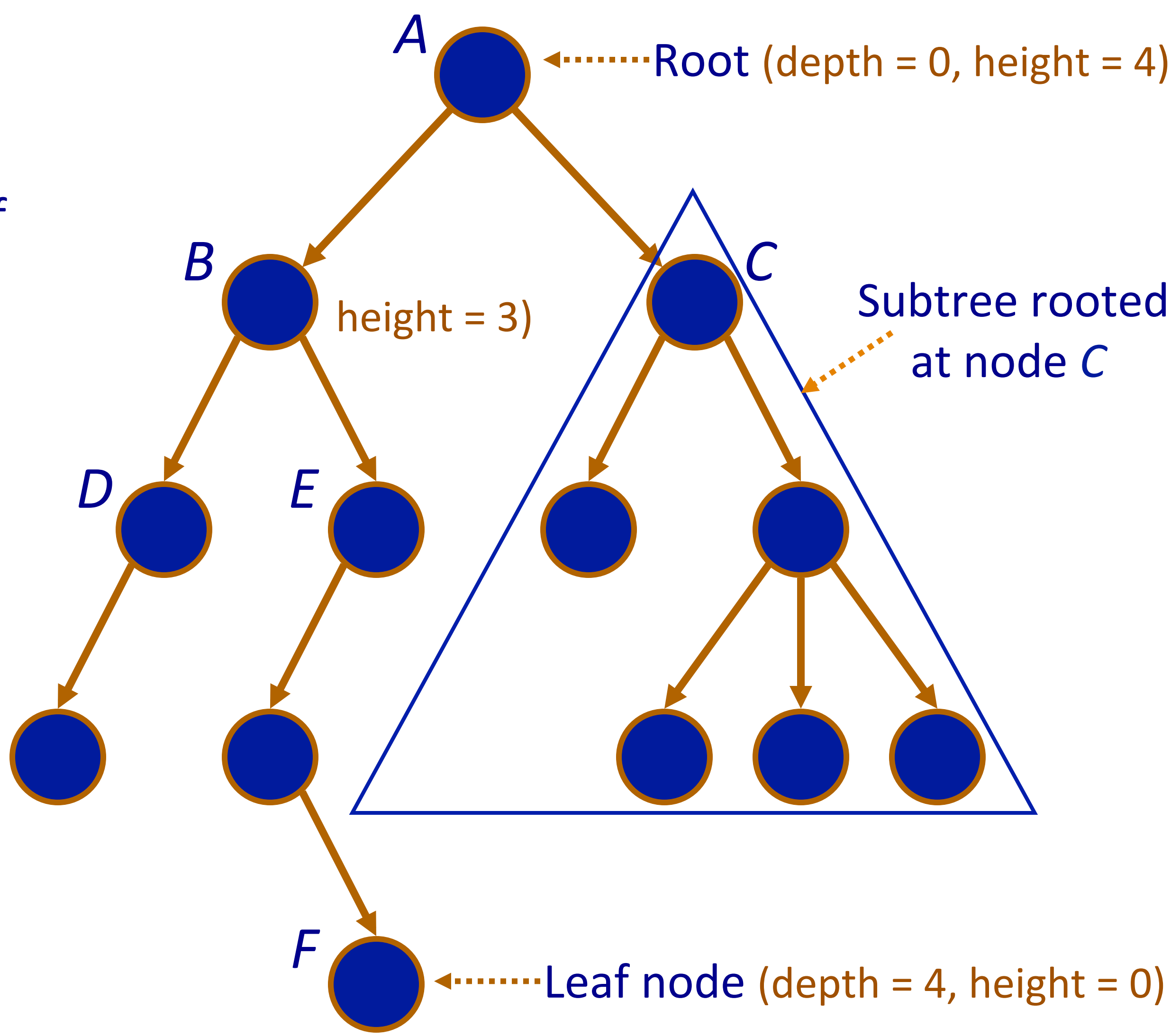
Tree Characteristics (cont.)



- There is a single, unique path from the root to any node
  - Arcs don't join together
- A path's *length* is equal to the number of arcs traversed
- A node's *height* is equal to the maximum path length from that node to a leaf node:
  - A leaf node has a height of 0
  - The height of a tree is equal to the height of the root
- A node's *depth* is equal to the path length from the root to that node:
  - The root node has a depth of 0
  - A tree's depth is the maximum depth of all its leaf nodes (which, of course, is equal to the tree's height)

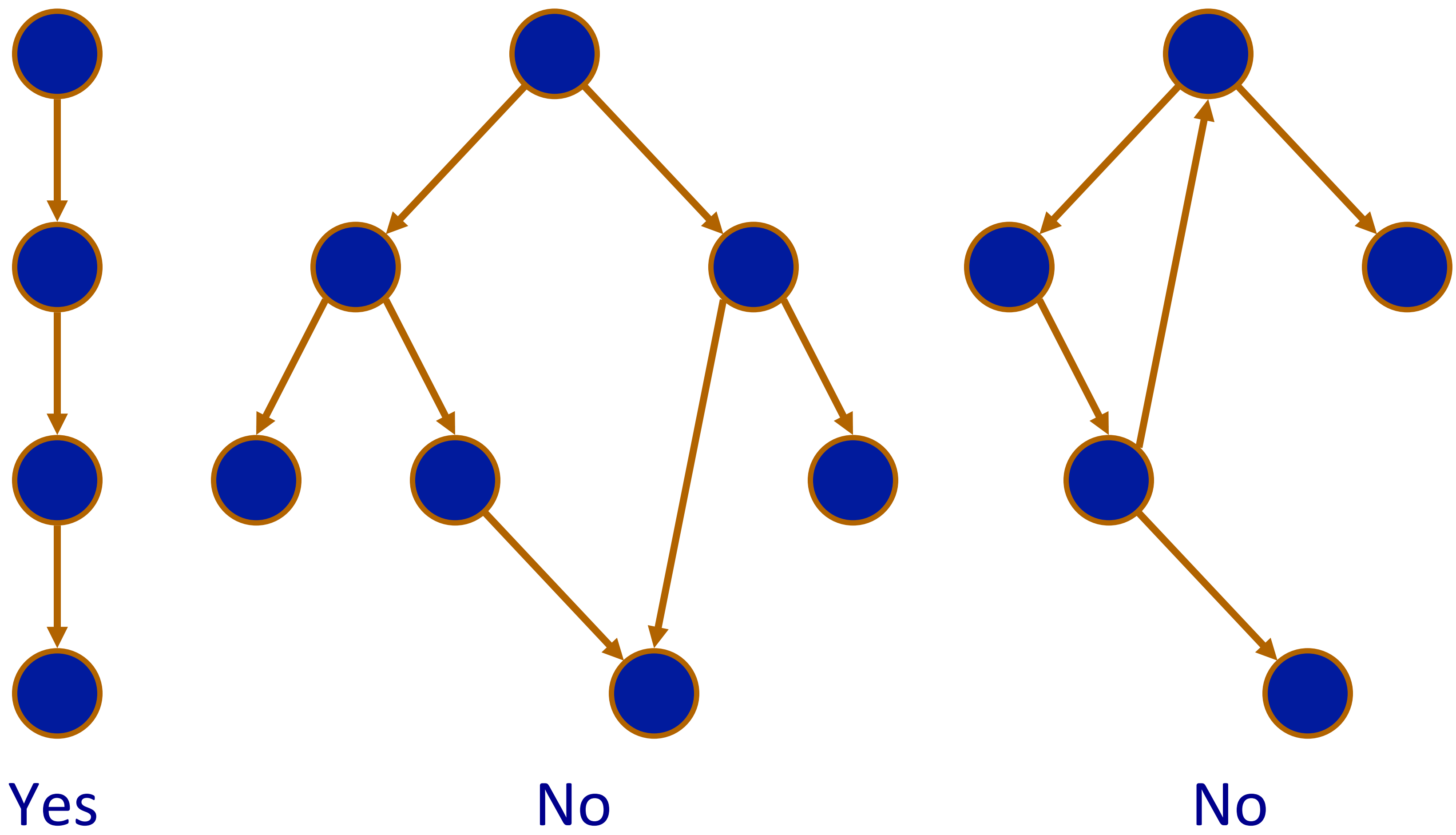
Tree Characteristics (cont.)

- 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



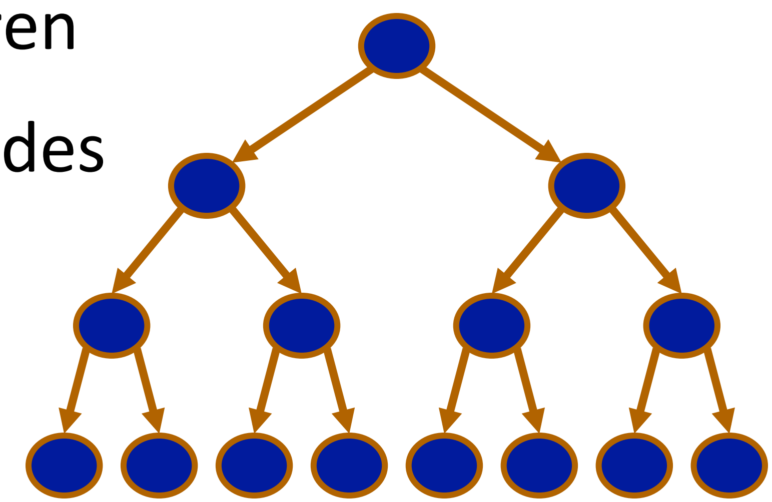
Tree Characteristics (cont.)

Are these trees?



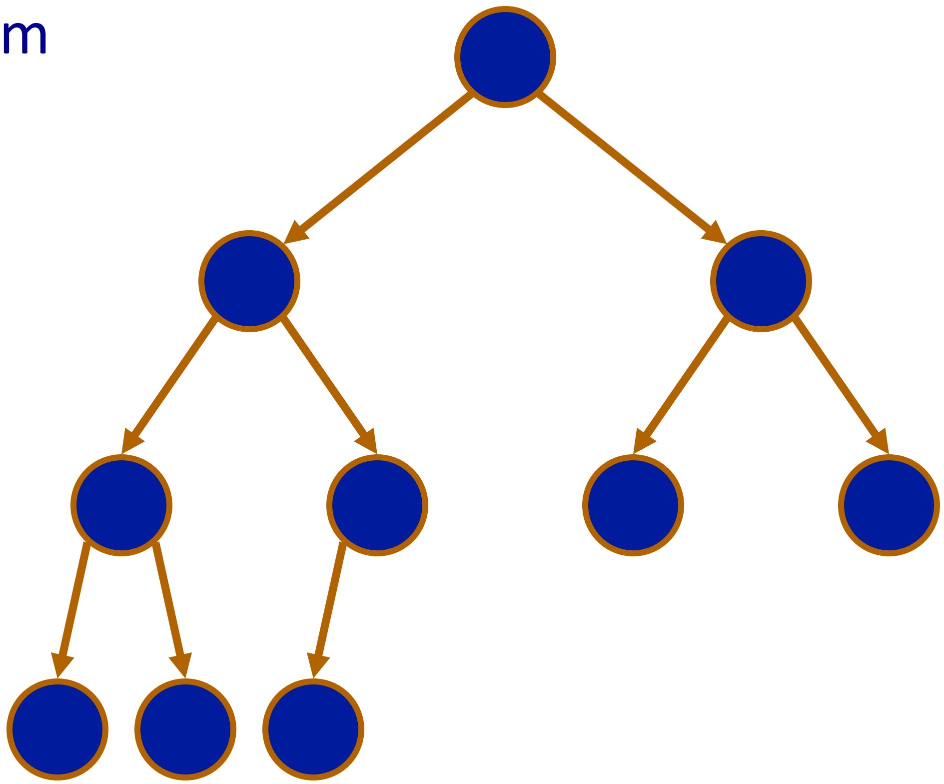
Binary Tree

- 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
  - Height of h will have $2^{h+1} - 1$ nodes
  - Height of h will have 2^h leaves



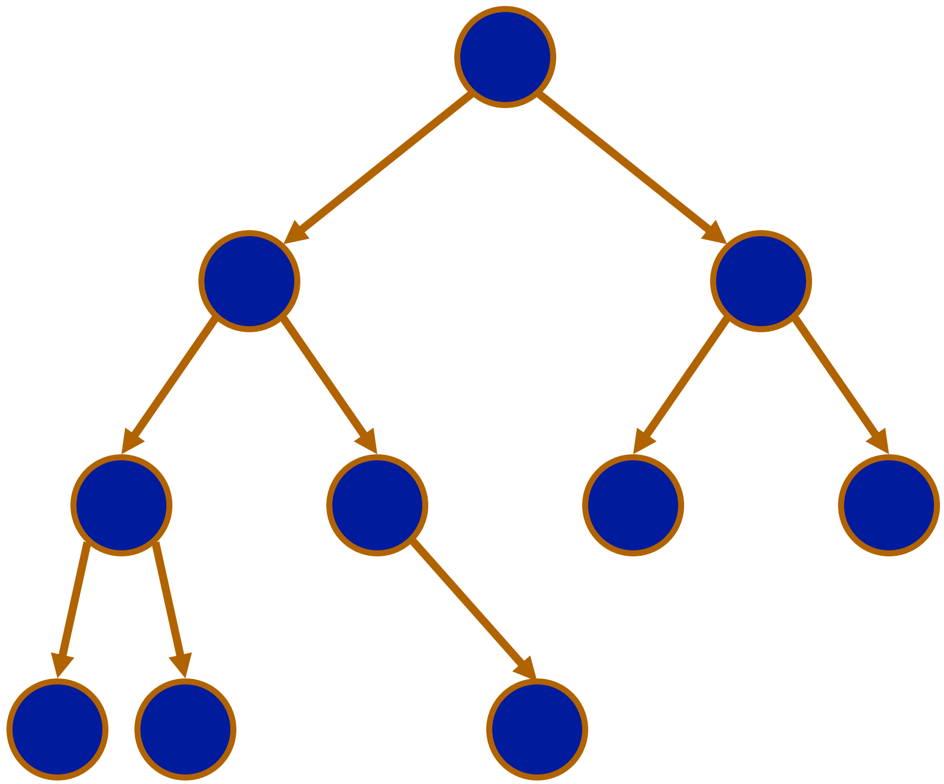
Binary Tree

- Complete Binary Tree:
  full except for the bottom level which is filled from left to right



Binary Tree

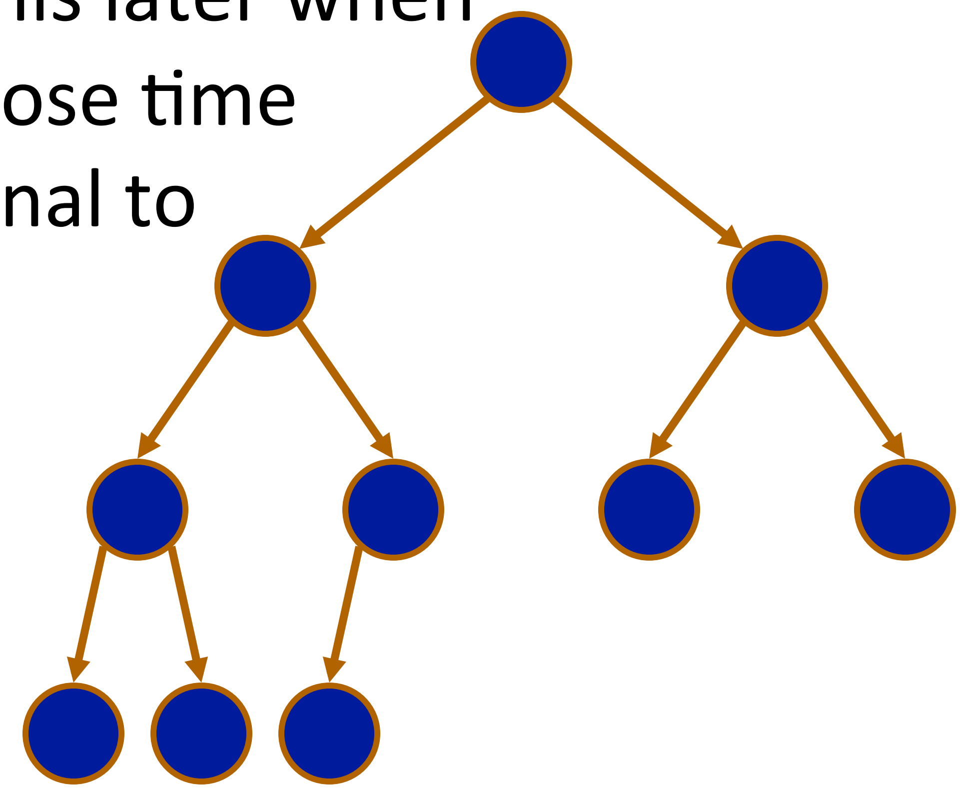
- Is this a complete binary tree?



Complete Tree: Height and Node Count

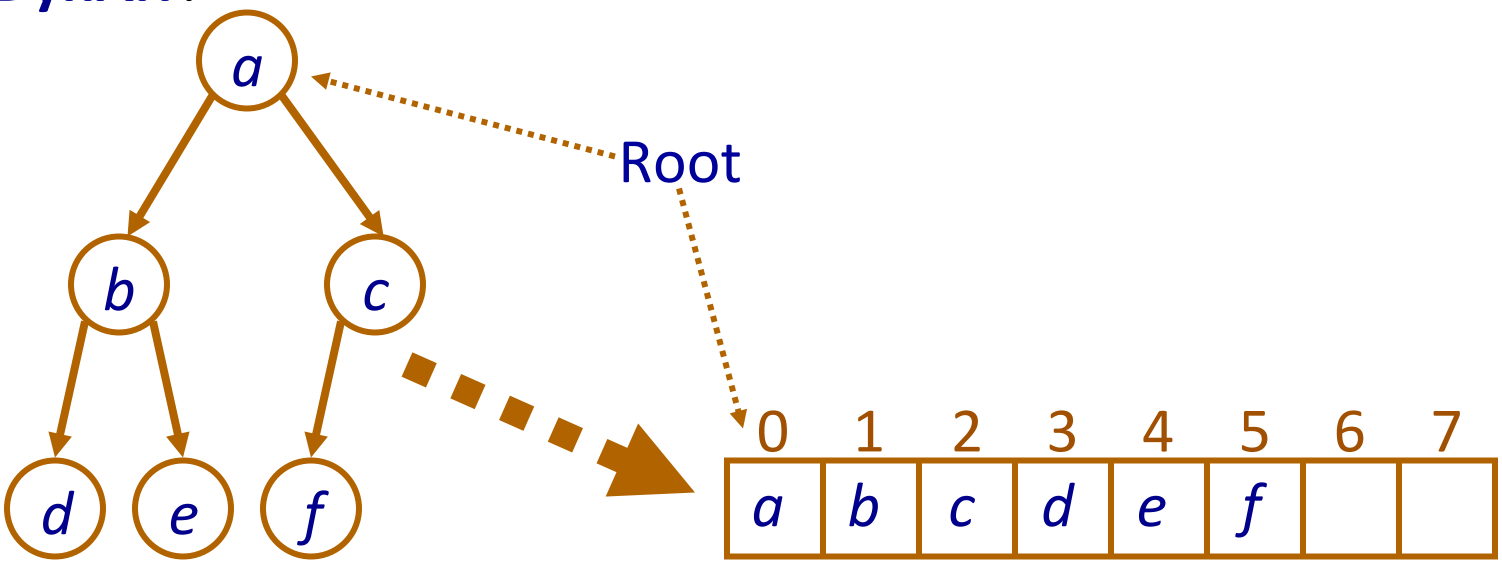
- What is the height of a complete binary tree that has *n* nodes?

- We will come back to this later when we have algorithms whose time complexity is proportional to the path length



Dynamic Array Representation

Complete binary tree has structure that is efficiently implemented with a **DynArr**:



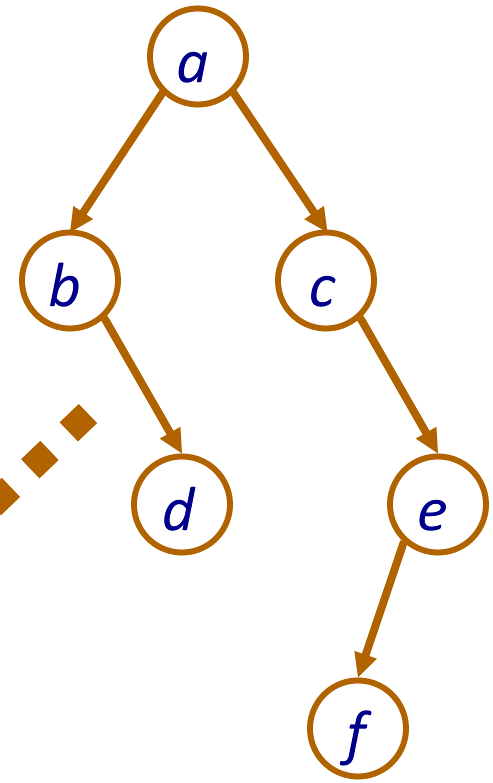
– Children of node i are stored at $2i + 1$ and $2i + 2$

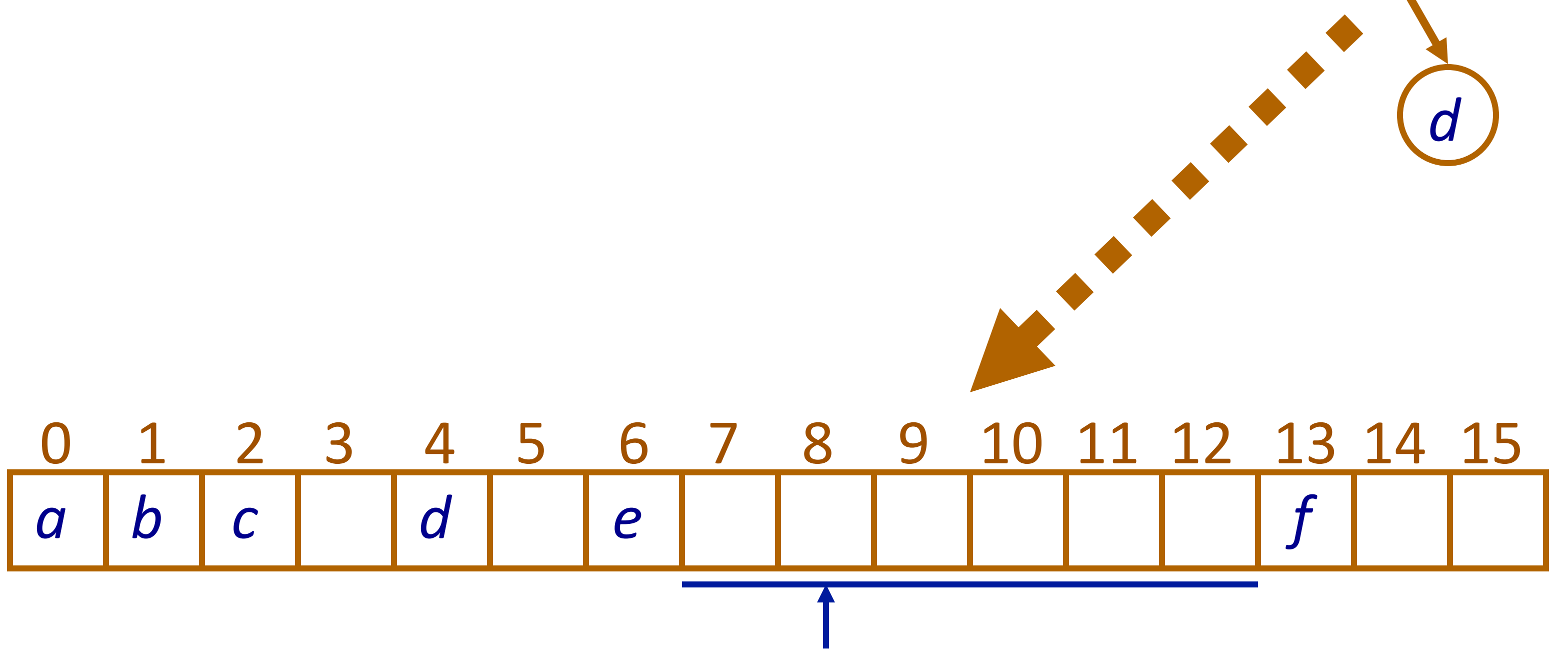
– Parent of node i is at $\text{floor}((i - 1) / 2)$

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

Dynamic Array Implementation (cont.)

If the tree is not complete (it is thin, unbalanced, etc.), the **DynArr** implementation will be full of holes





Big gaps where the level is not filled!

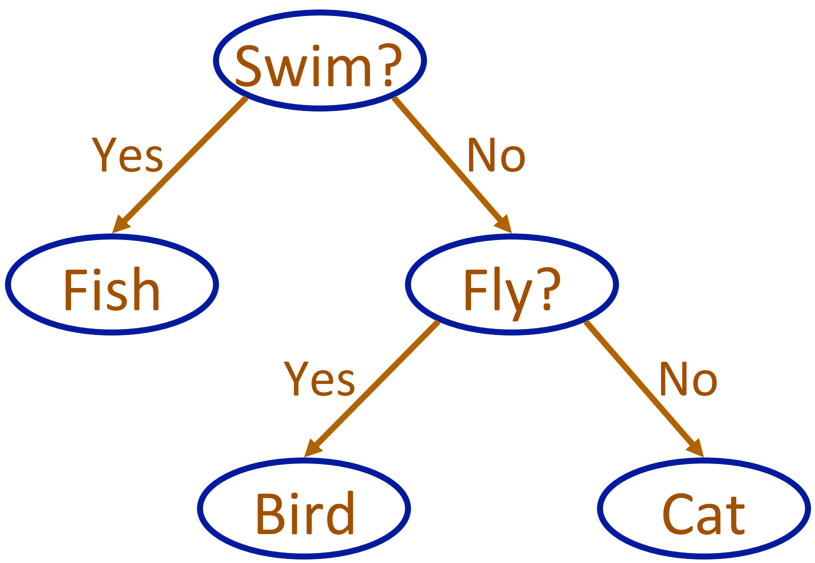
Dynamic Memory Implementation

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

Like the **Link** structure in a linked list: we will use this structure in several data structures

Binary Tree Application: Animal Game

- Purpose: guess an animal using a sequence of questions
  - –Internal nodes contain yes/no questions
  - –Leaf nodes are animals
- How do we build it?

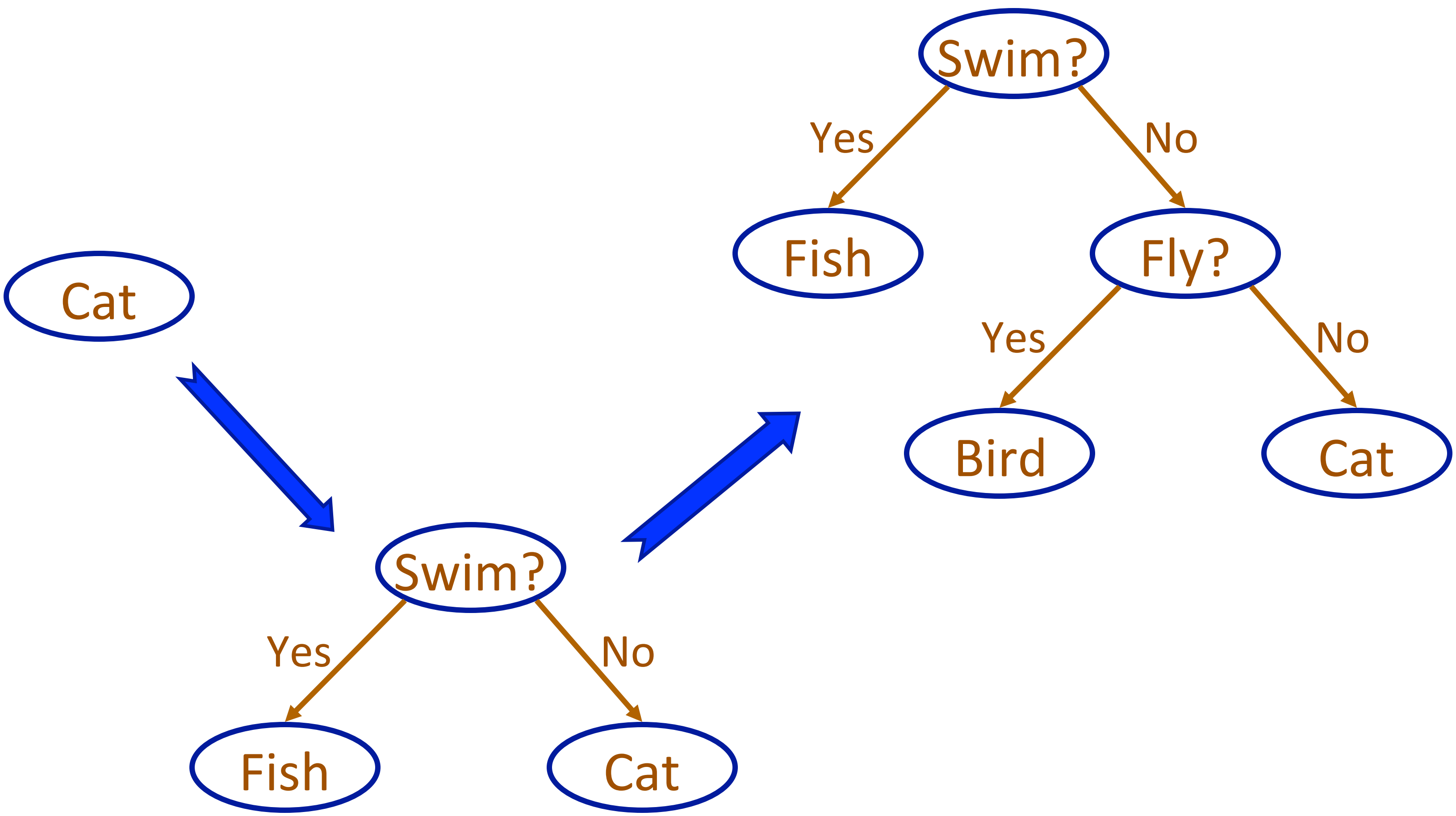


Binary Tree Application: Animal Game

Initially, tree contains a single animal (e.g., a "cat") stored in the root node

1. Start at root.
2. If internal node → ask yes/no question
   - Yes → go to left child and repeat step 2
   - No → go to right child and repeat step 2
3. If leaf node → guess "I know. Is it a …":
   - If right → done
   - If wrong → "learn" new animal by ***asking*** for a yes/no question that distinguishes the new animal from the guess

Binary Tree Application: Animal Game



Decision Tree

- If you can ask at most ***q*** questions, the number of possible answers we can distinguish between, ***n***, is the number of leaves in a binary tree with height at most q, which is at most 2^q
- Taking logs on both sides: $\log(n) = \log(2^q)$
- $\log(n) = q$: for n outcomes, we need q questions
- ***For 1,000,000 outcomes we need about 20 questions***

Still To Come…

- Implementation Concepts
- Implementation Code