IT 179

16

Trees

Chapter Objectives

- To learn how to use a tree to represent a hierarchical organization of information
- To learn how to use recursion to process trees
- To understand the different ways of traversing a tree
- To understand the differences between binary trees, binary search trees, and heaps
- To learn how to implement binary trees, binary search trees, and heaps using linked data structures and arrays

Chapter Objectives (cont.)

- To learn how to use Lambda Expressions and Functional Interfaces to simplify coding
- To learn how to use a binary search tree to store information so that it can be retrieved in an efficient manner
- To learn how to implement a priority queue using a heap
- To learn how to use a Huffman tree to encode characters using fewer bytes than ASCII or Unicode, resulting in smaller files and reduced storage requirements

Trees—Introduction

- All previous **data organizations** we've studied are linear—each element can have only one predecessor and successor
- Accessing all elements in a linear sequence is $O(n)$
- **Trees** are nonlinear and hierarchical
- Tree nodes can have multiple successors, but only one predecessor

Trees—Introduction (cont.)

- Trees can represent hierarchical organizations of information:
  - class hierarchy
  - disk directory and subdirectories
  - family tree
- Trees are recursive data structures because they can be defined recursively
- Many methods to process trees are written recursively

List and Tree Form of a Directory

C:

Users

Administrator

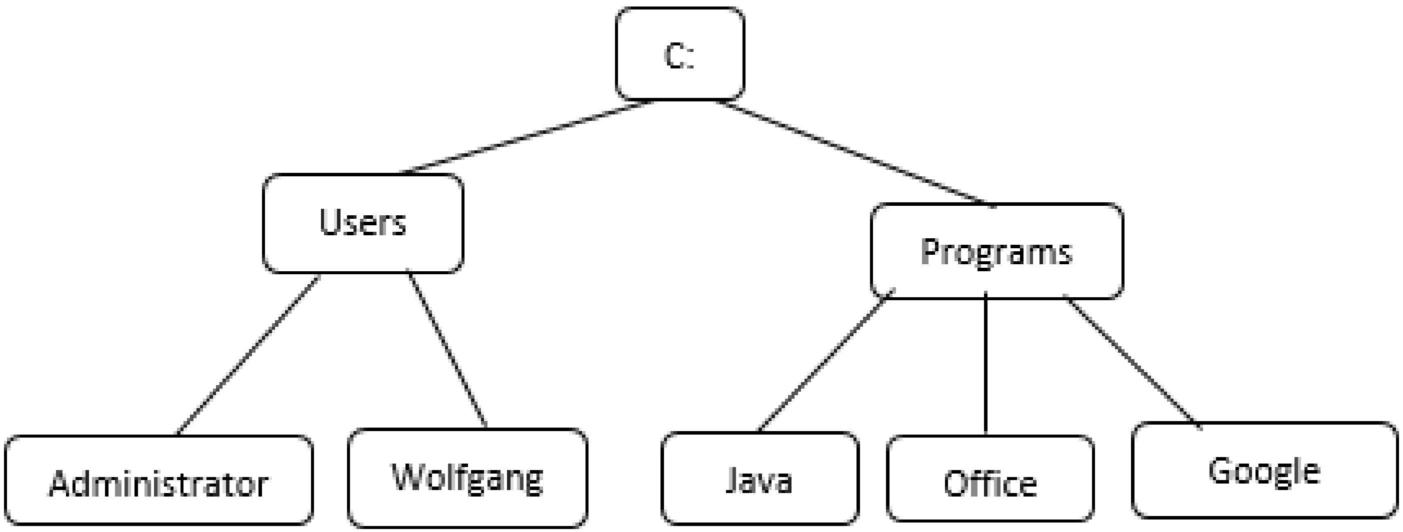
Wolfgang

Programs

Java

Office

Google

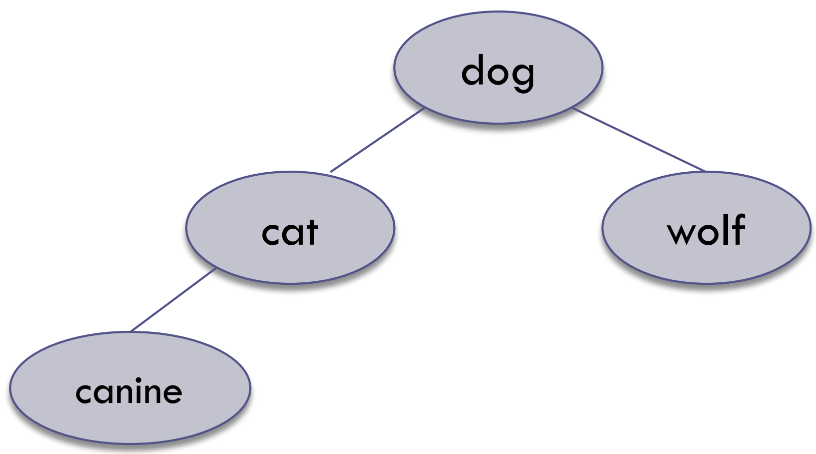


Tree Terminology and Applications

Section 6.1

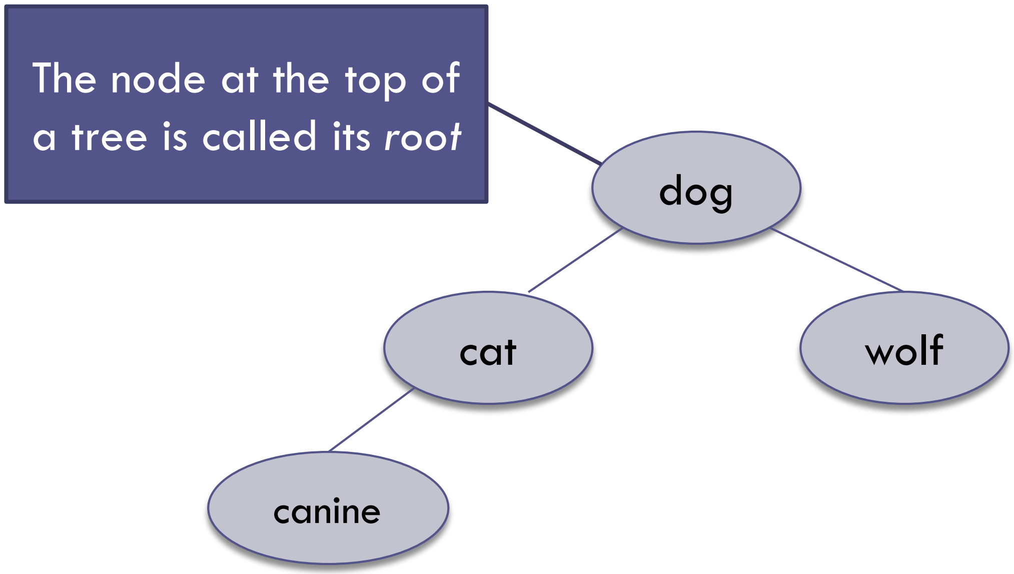
Tree Terminology

A tree consists of a collection of elements or nodes, with each node linked to its successors



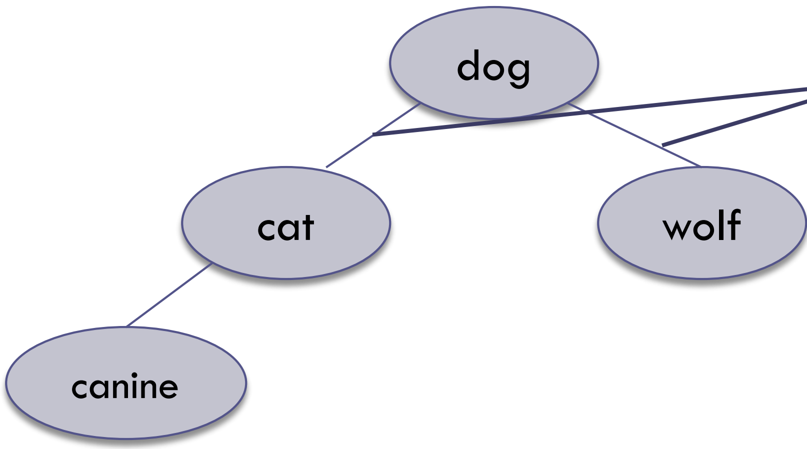
Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors



Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors

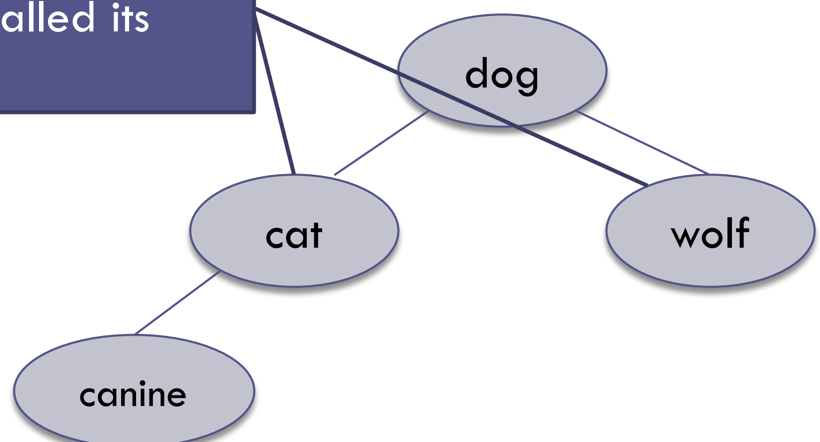


The links from a node to its successors are called *branches*

Tree Terminology (cont.)

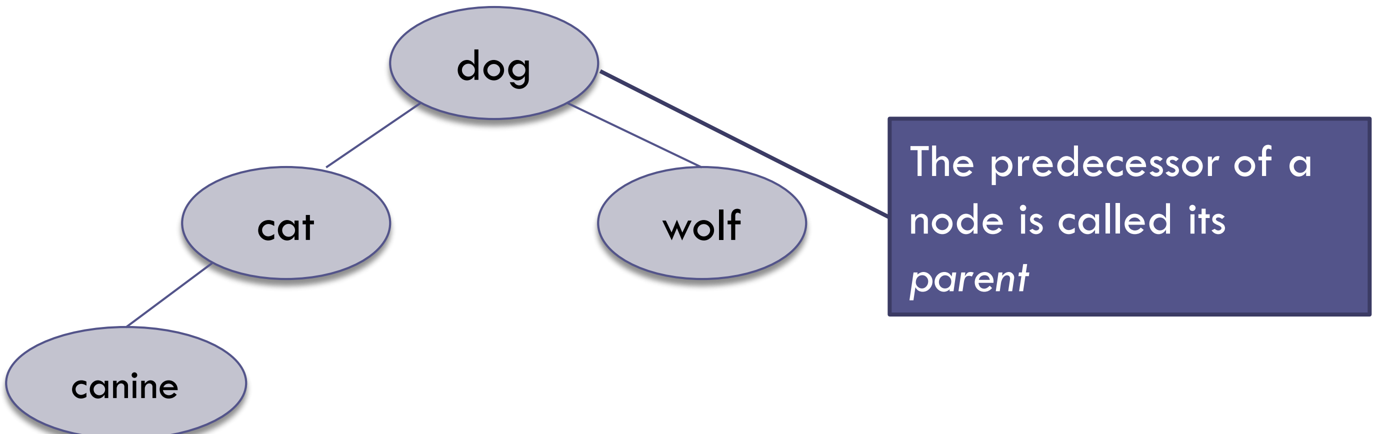
A tree consists of a collection of elements or nodes, with each node linked to its successors

The successors of a node are called its *children*



Tree Terminology (cont.)

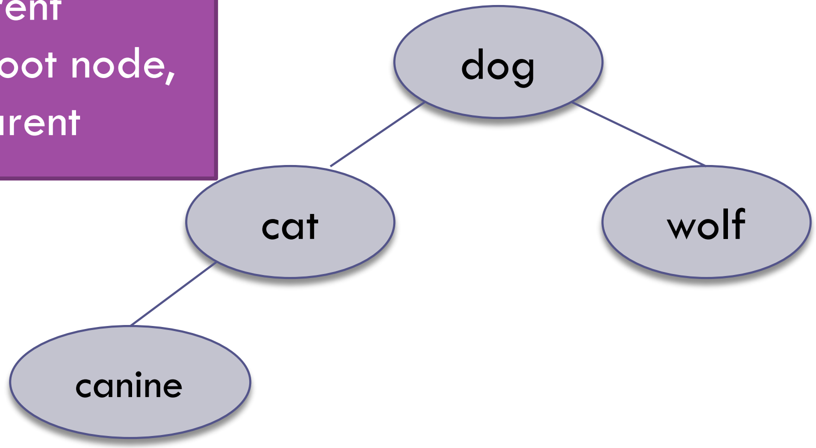
A tree consists of a collection of elements or nodes, with each node linked to its successors



Tree Terminology (cont.)

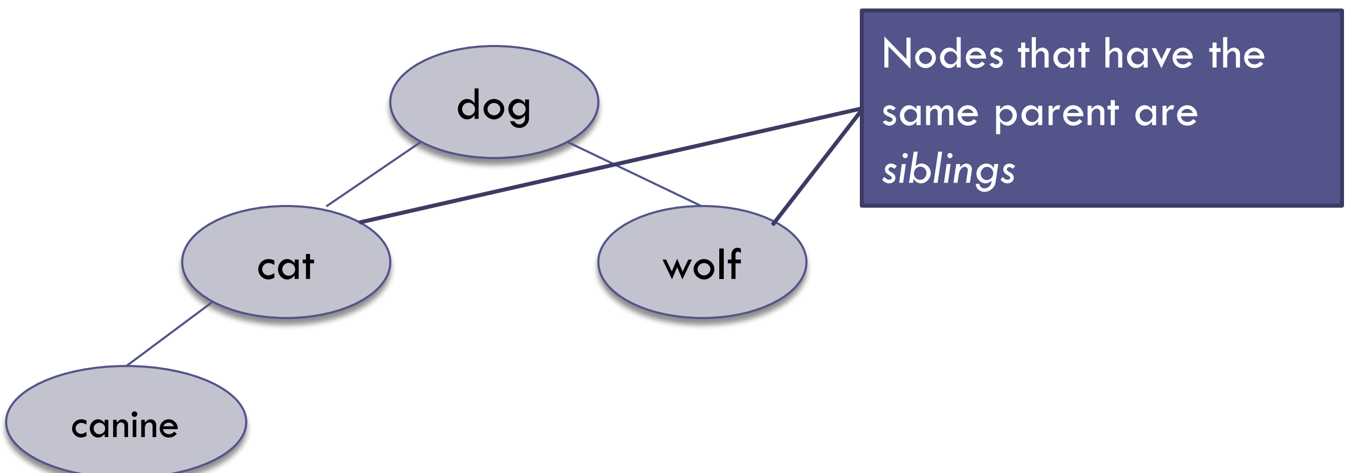
A tree consists of a collection of elements or nodes, with each node linked to its successors

Each node in a tree has exactly one parent except for the root node, which has no parent



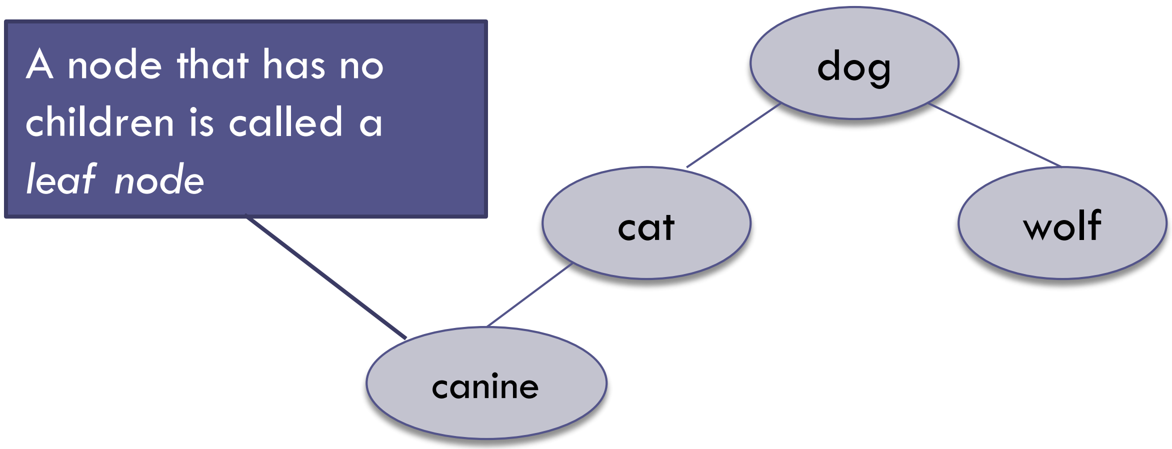
Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors



Tree Terminology (cont.)

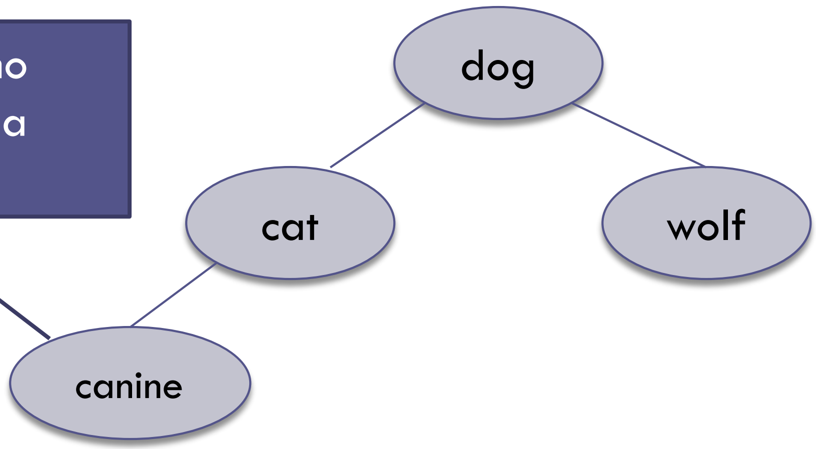
A tree consists of a collection of elements or nodes, with each node linked to its successors



Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors

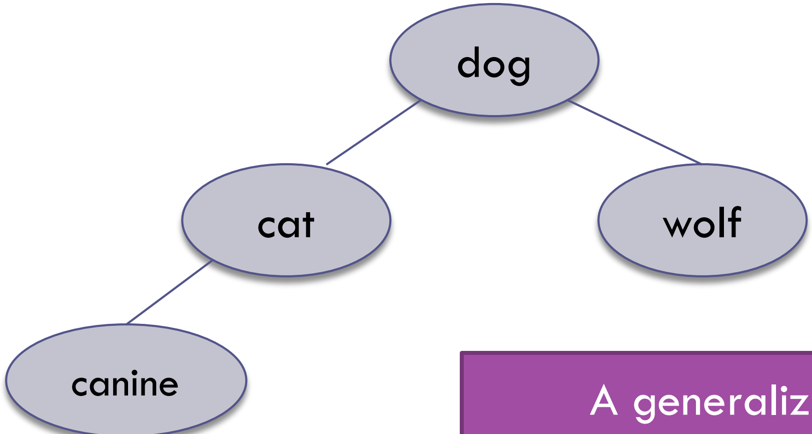
A node that has no children is called a *leaf node*



Leaf nodes also are known as *external nodes*, and nonleaf nodes are known as *internal nodes*

Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors

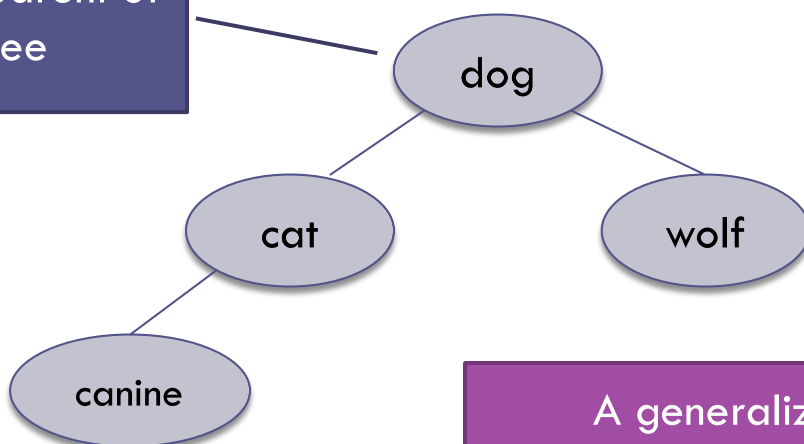


A generalization of the parent-child relationship is the *ancestor-descendant relationship*

Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors

dog is the parent of cat in this tree

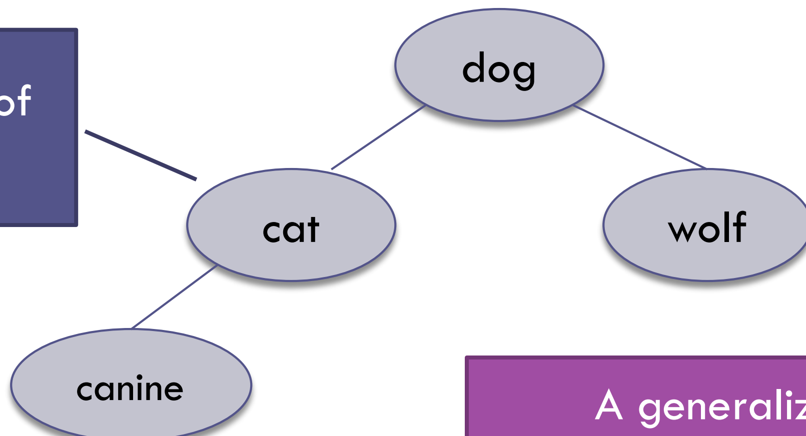


A generalization of the parent-child relationship is the *ancestor-descendant relationship*

Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors

cat is the parent of canine in this tree

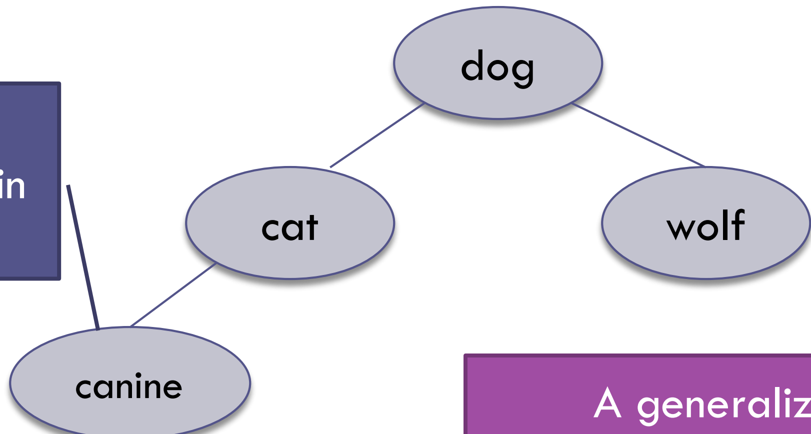


A generalization of the parent-child relationship is the *ancestor-descendant relationship*

Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors

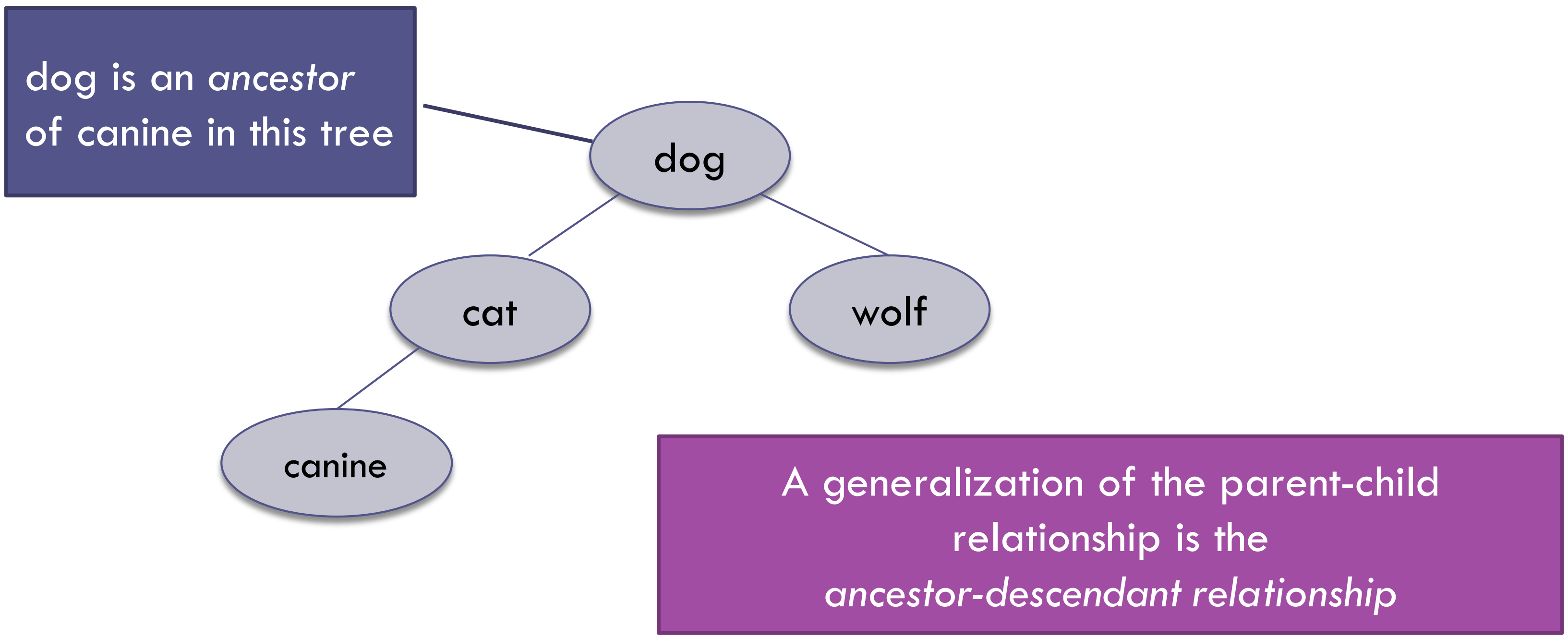
canine is a *descendant* of cat in this tree



A generalization of the parent-child relationship is the *ancestor-descendant relationship*

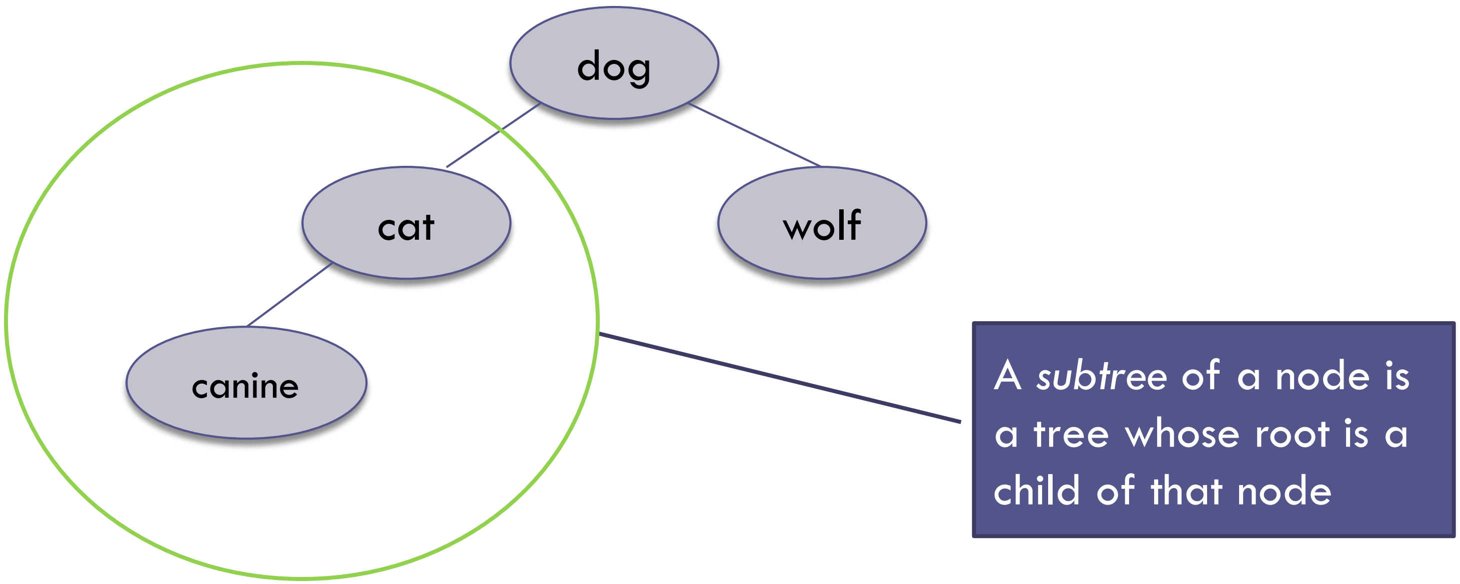
Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors



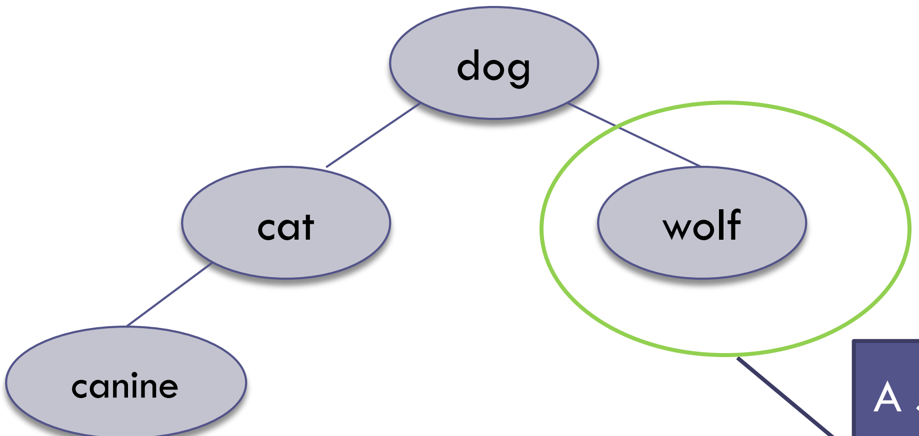
Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors



Tree Terminology (cont.)

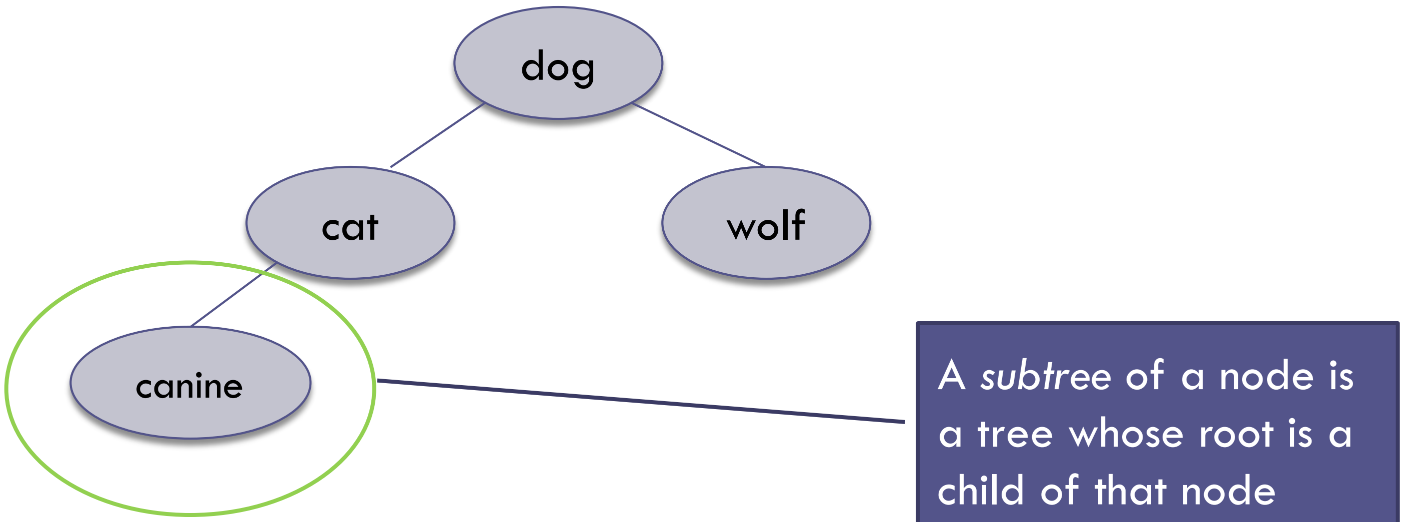
A tree consists of a collection of elements or nodes, with each node linked to its successors



A *subtree* of a node is a tree whose root is a child of that node

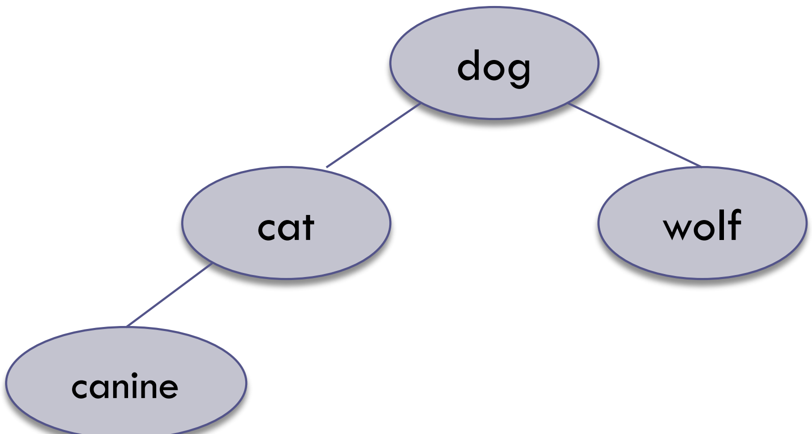
Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors



Tree Terminology (cont.)

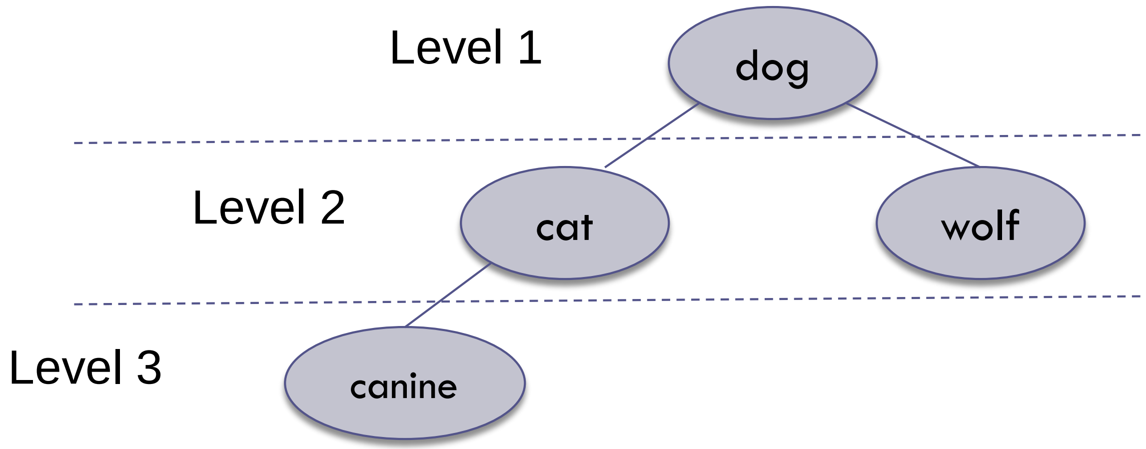
A tree consists of a collection of elements or nodes, with each node linked to its successors



The *level of a node* is determined by its distance from the root

Tree Terminology (cont.)

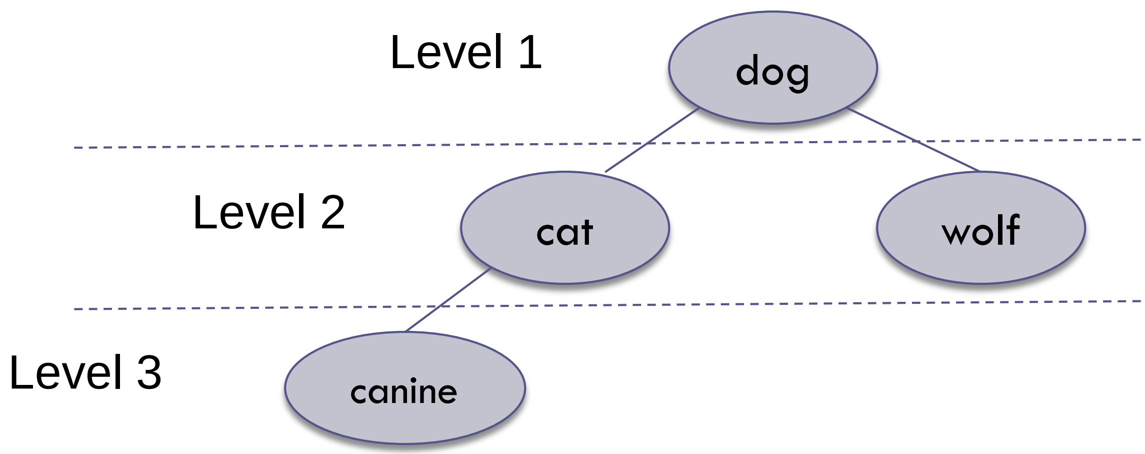
A tree consists of a collection of elements or nodes, with each node linked to its successors



The *level of a node* is its distance from the root plus 1

Tree Terminology (cont.)

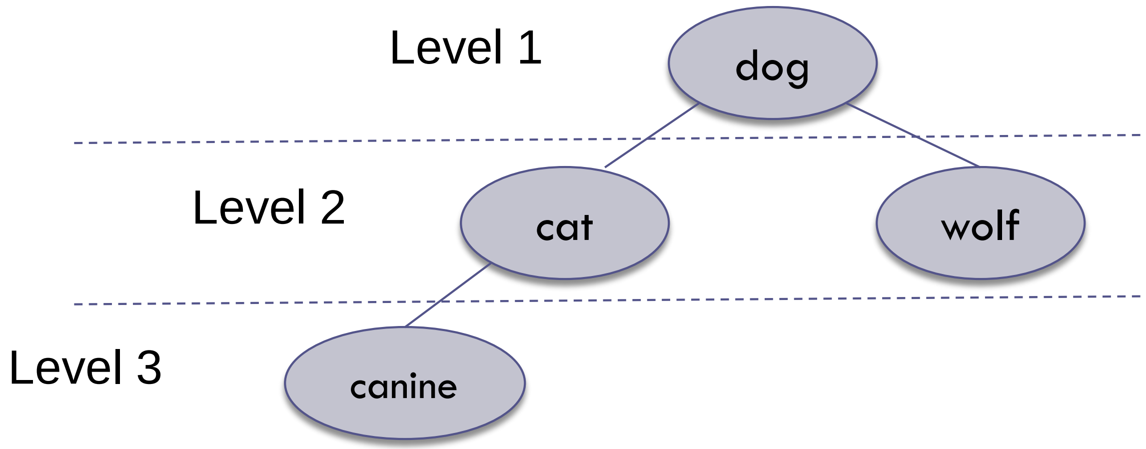
A tree consists of a collection of elements or nodes, with each node linked to its successors



The *level of a node* is defined recursively

Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors



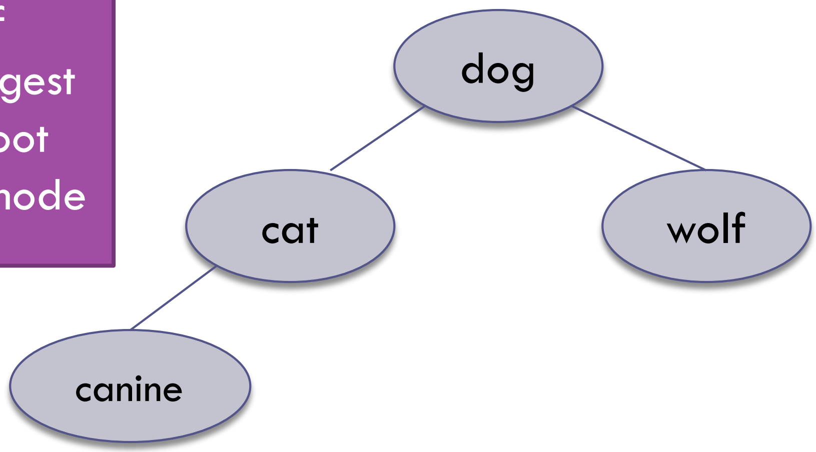
The *level of a node* is defined recursively

- If node *n* is the root of tree T, its level is 1
- If node *n* is not the root of tree T, its level is 1 + the level of its parent

Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors

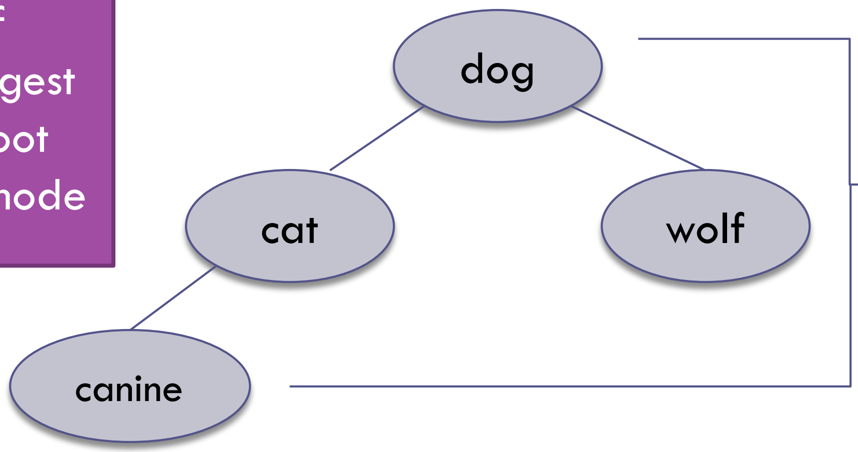
The ***height*** *of a tree* is the number of nodes in the longest path from the root node to a leaf node



Tree Terminology (cont.)

A tree consists of a collection of elements or nodes, with each node linked to its successors

The *height of a tree* is the number of nodes in the longest path from the root node to a leaf node



The height of this tree is 3

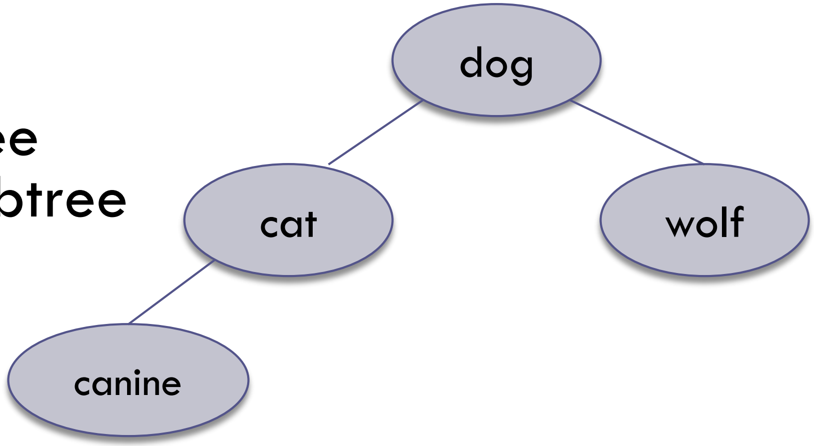
Binary Trees

- In a binary tree, each node has two subtrees
- A set of nodes T is a **binary tree** if either of the following is true
  - T is empty
  - Its root node has two subtrees, T_L and T_R, such that T_L and T_R are binary trees

  (T_L = left subtree; T_R = right subtree)

Binary Search Tree

- Binary search trees
  - All elements in the left subtree precede those in the right subtree
- A formal definition:



A set of nodes T is a binary search tree if either of the following is true:

  - T is empty
  - If T is not empty, its root node has two subtrees, T_L and T_R, such that T_L and T_R are binary search trees and the value in the root node of T is greater than all values in T_L and is less than all values in T_R

Binary Search Tree (cont.)

- A binary search tree never has to be sorted because its elements always satisfy the required order relationships
- When new elements are inserted (or removed) properly, the binary search tree maintains its order
- In contrast, a sorted array must be expanded whenever new elements are added, and compacted whenever elements are removed—expanding and contracting are both O(n)

Binary Search Tree (cont.)

- When searching a BST, each probe has the potential to eliminate half the elements in the tree, so searching can be O(log n)
- In the worst case, searching is O(n)

Recursive Algorithm for Searching a Binary Tree

1. `if` the tree is empty
2. return null *(target is not found)*

   `else if` the target matches the root node's data

3. return the data stored at the root node

   `else if` the target is less than the root node's data

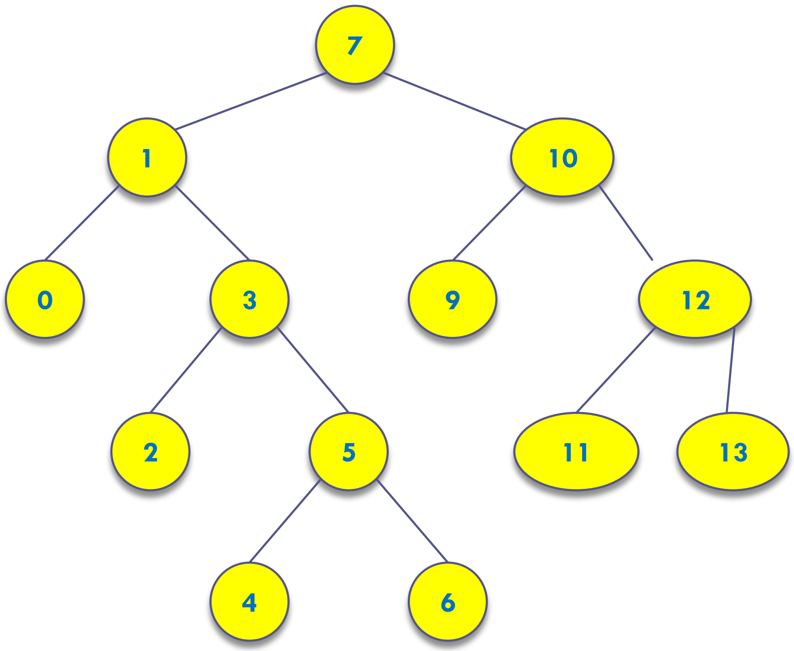
4. return the result of searching the left subtree of the root

   `else`

5. return the result of searching the right subtree of the root

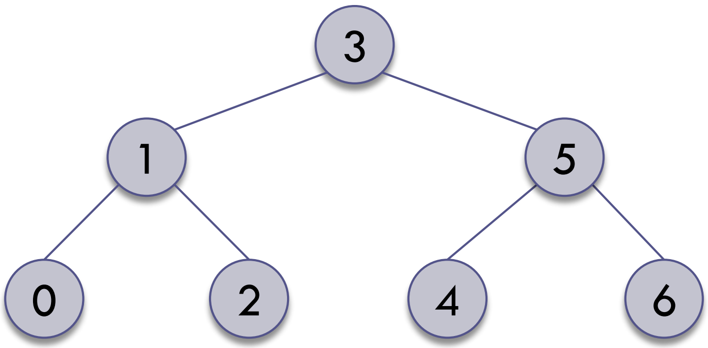
Full, Perfect, and Complete Binary Trees

- A full binary tree is a binary tree where all nodes have either 2 children or 0 children (the leaf nodes)



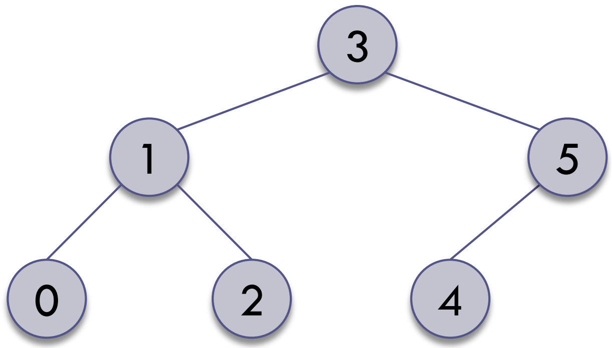
Full, Perfect, and Complete Binary Trees (cont.)

- A *perfect binary tree* is a full binary tree of height n with exactly $2^n - 1$ nodes
- In this case, $n = 3$ and $2^n - 1 = 7$



Full, Perfect, and Complete Binary Trees (cont.)

- A *complete binary tree* is a perfect binary tree through level $n-1$ with some extra leaf nodes at level n (the tree height), all toward the left



Tree Traversals

Section 6.2

Tree Traversals

- Often we want to determine the nodes of a tree and their relationship
  - We can do this by walking through the tree in a prescribed order and visiting the nodes as they are encountered
  - This process is called ***tree traversal***
- Three common kinds of tree traversal
  - Preorder
  - Inorder
  - Postorder

Tree Traversals (cont.)

- Preorder: visit root node, traverse T_L, traverse T_R
- Inorder: traverse T_L, visit root node, traverse T_R
- Postorder: traverse T_L, traverse T_R, visit root node

Algorithm for Preorder Traversal

1. `if` the tree is empty
2. Return.
   `else`
3. Visit the root.
4. Preorder traverse the left subtree.
5. Preorder traverse the right subtree.

Algorithm for Inorder Traversal

1. `if` the tree is empty
2. Return.
   `else`
3. Inorder traverse the left subtree.
4. Visit the root.
5. Inorder traverse the right subtree.

Algorithm for Postorder Traversal

1. `if` the tree is empty
2. Return.
   `else`
3. Postorder traverse the left subtree.
4. Postorder traverse the right subtree.
5. Visit the root.

Example

- Preorder traversal

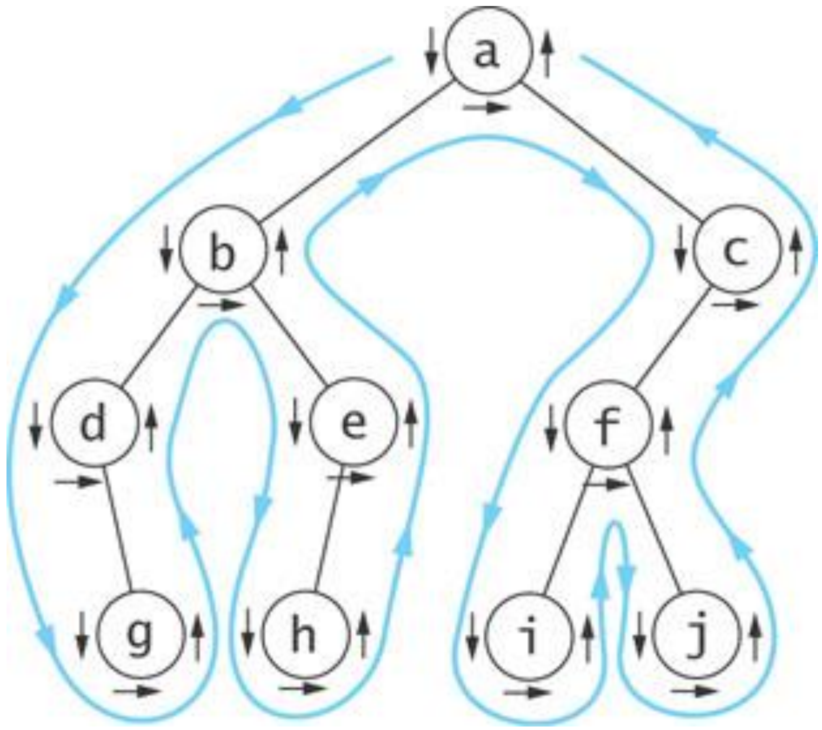
a b d g e h c f i j

- Inorder traversal

d g b h e a i f j c

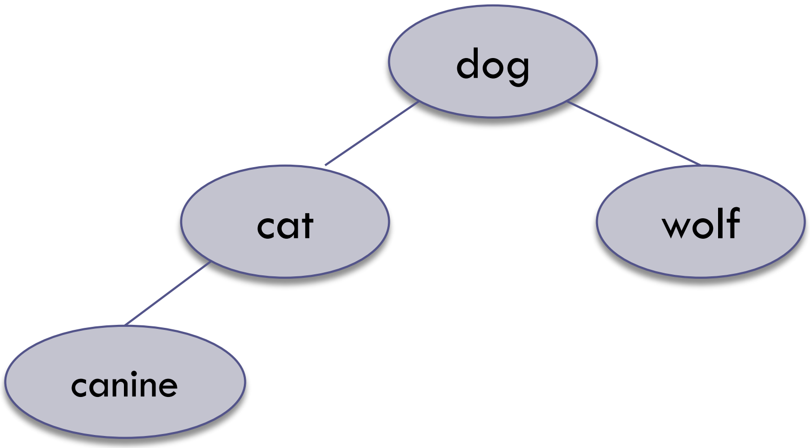
- Postorder traversal

g d h e b i j f c a



Traversals of Binary Search Trees and Expression Trees

- An inorder traversal of a binary search tree results in the nodes being visited in sequence by increasing data value

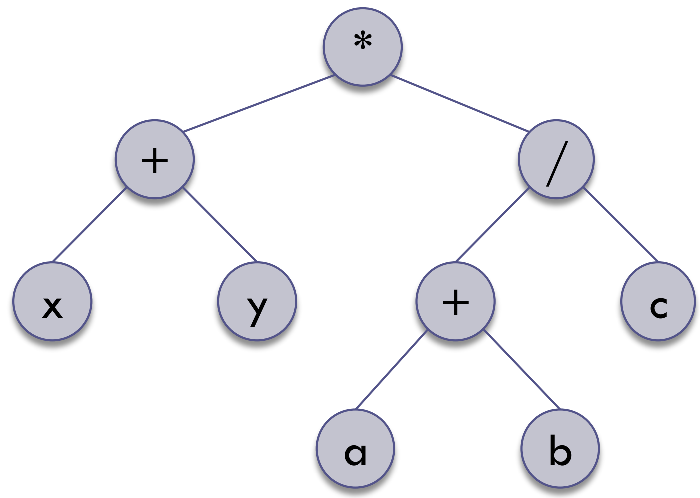


canine, cat, dog, wolf

Traversals of Binary Search Trees and Expression Trees (cont.)

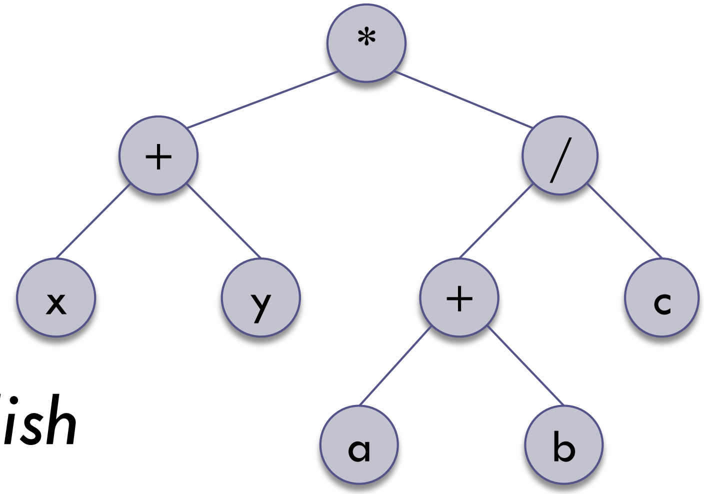
- An inorder traversal of this expression tree results in the sequence
  x + y * a + b / c
- If we insert parentheses where they belong, we get the infix form:

```
(x + y) * ((a + b) / c)
```



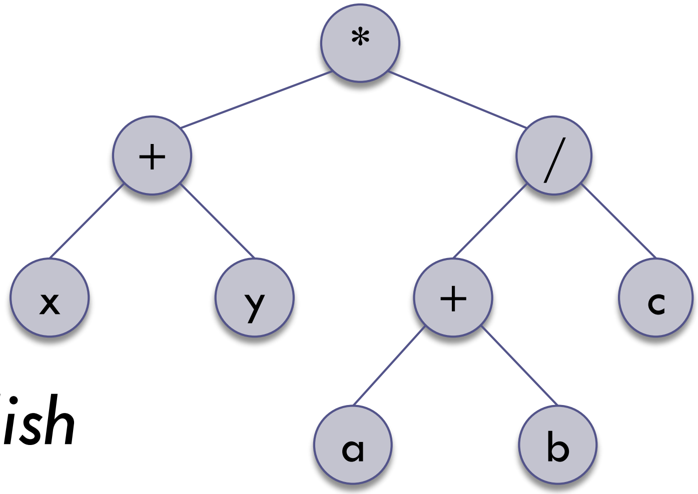
Traversals of Binary Search Trees and Expression Trees (cont.)

- A postorder traversal of this expression tree results in the sequence
  x y + a b + c / *
- This is the *postfix* or *reverse polish* form of the expression
- Operators follow operands



Traversals of Binary Search Trees and Expression Trees (cont.)

- A preorder traversal of this expression tree results in the sequence
  * + x y / + a b c
- This is the *prefix* or *forward polish* form of the expression
- Operators precede operands



Examples

- Create a Binary Tree of Students
- All three traversals

Examples (cont'd)

- Create a Binary Tree of Students
- Search for student: ("John", "Havana“) and print out his GPA
- Print out all students with a last name starting with the letter N or after
- Delete student ("John", "Dolce")
- Delete all students with a GPA < 2.5
- Change the root to ("Alex", “Carlson”)