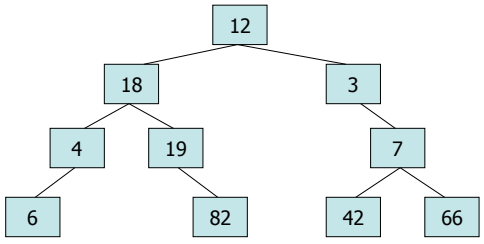
Introducing Trees

"I think that I shall never see,
a data structure as lovely as a tree...."

Trees

- Consist of a finite set of "nodes"
- If the set of nodes is not empty, there is one special node called the "root"
- Each node (with the exception of the root) has exactly one "parent"
  - The root does not have a parent
- The root is the "ancestor" of all nodes

A binary tree of integers



Basic tree terminology

- root
- parent, grandparent
- child, grandchild
- sibling
- leaf ("exterior node")
- interior node
- sub-tree
- ancestor
- descendant
- depth of a node
  - root node depth == 0
- height of a tree

Binary trees

- Trees in which any node has at most two children (a left child and a right child)
  - The sub-tree with the left child of the root as *its* root is called the left sub-tree
  - The sub-tree with the right child of the root as *its* root is called the right sub-tree
- A full binary tree
  - Every leaf is at exactly the same depth, and every interior node has exactly two children

Questions:

- How many nodes are in a full binary tree of height 2? of height 4?
- How many nodes are there in a full ternary (3 children/ node) tree of height 3?
- How many nodes does a full binary tree of depth *n* have?

More types of trees exist

- Complete binary tree:
  - A full binary tree where the rightmost leaves may be missing
- Also worth noting that although binary trees are very common, we can also have ternary trees, k-nary trees, and so on.

Representing trees with arrays

- Complete binary trees can be represented by arrays
  - Put the root at index 0
  - The nodes at level (depth) 1 from left to right occupy the next 2 indices
  - The nodes at level (depth) 2 from left to right occupy the next 4 indices
  - And so on....

Challenge #1

- What is the layout of the binary tree stored in the following array?

  8, 7, 4, 32, 7, 12, 10, 14, 12, 13, 15

- If a node is at index *i*, what is the index of its parent?
- What is the index of its left child (if one exists)?
- What is the index of its right child (if one exists)?
- The above representation isn't suitable for all binary trees. Why not?

More about binary trees

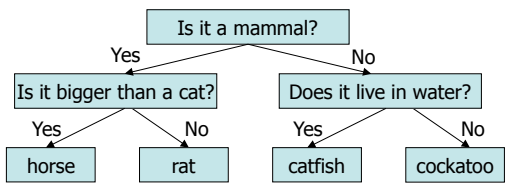
Some examples, some code, and more terminology....

Binary Taxonomy Trees

- Taxonomy:
  - The orderly classification of plants and animals according to their presumed natural relationships
- A binary taxonomy tree:
  - Allows you to identify a plant/animal from its characteristics
  - Yes/no questions are used to guide you through the tree

Example

- Tree for horse/rat/catfish/cockatoo



Challenge #2

- Create a binary taxonomy tree with eight animals.
  - Is the tree full?
  - Is the tree complete?

Recursive definition of a binary tree

- A binary tree is either empty, or else it consists of data, a left sub-tree, and a right sub-tree
- Example:

```
public class BinaryTree<E> {
    private E data;
    private BinaryTree<E> left;
    private BinaryTree<E> right;
    . . . .
}
```

Building a simple binary tree

```
public class BinaryTree<E> {
    private E data;
    private BinaryTree<E> left;
    private BinaryTree<E> right;

    public BinaryTree( E val )
    {
        data = val;
        left = right = null;
    }

    // accessors and modifiers for data and
    // left/right sub-trees go here
}
```

Challenge #3: Tree methods

- Instance methods
  - Add a method `isLeaf()` to BinaryTree that returns true iff this tree is a leaf.
- Static methods:
  - Add a method `size(BinaryTree root)` to BinaryTree that returns the size of the tree starting at the node root.
  - Add a static method `height(BinaryTree root)` to BinaryTree that returns the height of the tree, given the root node.

Tree traversals

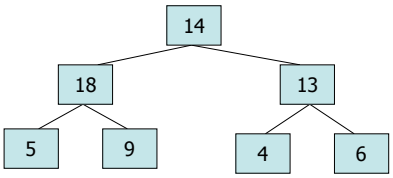
- In a tree traversal, we visit each of the tree's nodes precisely once
  - This is typically to **do** something at each node, such as print out data, modify it, etc.
- There are several common traversals:
  - breadth-first
  - in-order
  - post-order
  - pre-order

Breadth-first traversals

- In a breadth-first traversal, we visit:
  - the root node
  - then the nodes at level 1 from left to right
  - then the nodes at level 2 from left to right
  - then the nodes at level 3 from left to right
  - ....

A breadth-first traversal

- What would a breadth-first traversal to print out this tree's data look like?



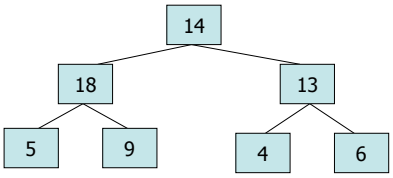
Answer: 14, 18, 13, 5, 9, 4, 6

In-order traversals

- In an in-order traversal, we:
  - Do an in-order traversal of the left sub-tree
  - Visit the root
  - Do an in-order traversal of the right sub-tree

An in-order traversal

- What would a in-order traversal to print out this tree's data look like?



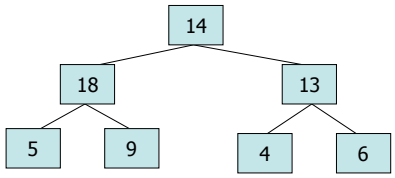
Answer: 5, 18, 9, 14, 4, 13, 6

Pre-order traversals

- In a pre-order traversal, we:
  - Visit the root
  - Do a pre-order traversal of the left sub-tree
  - Do a pre-order traversal of the right sub-tree

A pre-order traversal

- What would a pre-order traversal to print out this tree's data look like?



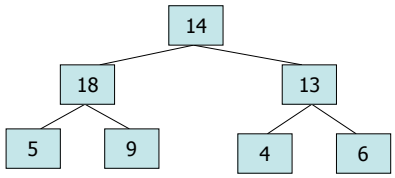
Answer: 14, 18, 5, 9, 13, 4, 6

Post-order traversals

- In a post-order traversal, we:
  - Do a post-order traversal of the left sub-tree
  - Do a post-order traversal of the right sub-tree
  - Visit the root

A post-order traversal

- What would a post-order traversal to print out this tree's data look like?



Answer: 5, 9, 18, 4, 6, 13, 14

Challenge #4

- Add the following instance methods to the BinaryTree class:
  - `inOrderPrint()`
  - `preOrderPrint()`
  - `postOrderPrint()`

Binary search trees

- These assume that the data in a binary tree can be compared in some pre-defined way
- Data is stored in a BST as follows:
  - Every element in a given node's left sub-tree is *less than or equal to* the value stored in that node
  - Every element in a given node's right sub-tree is *greater than* the value stored in that node

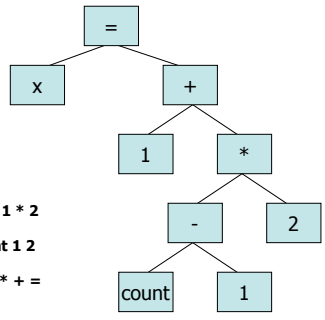
Binary search tree examples

- Building a BST
- Finding data in a BST
- Unbalanced binary trees

Expression trees

- Logical and mathematical expressions can be represented in trees
  - Each literal value or variable is a leaf node
  - Each operation is an interior node, with its value being the result of the operation on the left and right sub-trees
- Examples:
  - x = 1 + (count - 1) * 2
  - (!a | b) && !(c && d)
- Alternative mathematical expression syntaxes
  - Prefix, or "Polish notation"
  - Postfix, or "Reverse Polish notation" (RPN)

Expression tree example



Infix: **x = 1 + count – 1 * 2**

Prefix: **= x + 1 * - count 1 2**

Postfix: **x 1 count 1 – 2 * + =**

Huffman encoding

- The idea behind this is simple:
  - Give commonly occurring things short codes
  - Give longer codes to things that occur less frequently
- This approach minimizes the average number of bytes used to store text (or other data)
- A short tutorial on Huffman encoding is available at http://tinyurl.com/butq9

Uses for Huffman encoding

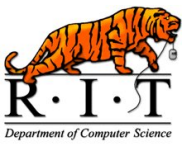
- MP3's
  - Last step before formatting a Level 3 stream is Huffman encoding
  - This is done for compression
- ZIP file compression

Some lingering issues

- We want the tree to be as balanced (i.e., as close to full) as possible, for efficiency, but:
  - Data may not be added in an order that automatically results in a balanced tree
  - Deletions leave holes in the trees
  - Updating "search key" values mean that the tree must be reorganized
- Solutions exist to these issues, but they're beyond the scope of this lecture.
  - Rest assured, however, that adding/deleting data (while retaining a reasonably full tree) can be done in O(log n) time....

When to use what for storage (part 2)

- Binary trees are good when:
  - You need frequent capacity changes in the data structure
  - High-speed manipulation (especially searches) is important
  - You're concerned with providing sorted access to data
  - You're unwilling to risk the "hiccups" involved with rehashing (coming up....)



Any questions?