Trees, Trees, Trees

More tree terms

- Note: subtree can start at any node
  - There’s a subtree rooted at C!
  - Subtrees follow same rules as trees
- A tree’s height is the largest number of nodes from root to leaf
  - Height of the tree on the right is 4 (A->C->D->F)
- Balanced binary tree
  - For all nodes, the height of the left and right subtree differ by at most 1
  - This tree is not balanced!
- Full tree
  - No missing nodes
  - For all nodes, height of left and right subtree are equal
Classes used in building binary trees

- As with linked lists, two classes in binary trees
  - TreeNode: an individual node in the tree
  - BinaryTree: a subtree rooted at a particular TreeNode
- TreeNodes support the usual operations
  - TreeNode (Object newItem)
  - TreeNode (Object newItem, TreeNode lt, TreeNode rt)
  - Object getItem()
  - void setItem (Object newItem)
  - TreeNode getLeft/Right()
  - TreeNode setLeft/Right (TreeNode left)
  - Note: this implementation doesn’t have “up” pointers in each node that point to the node’s parent
- These operations are straightforward
  - Similar to operations in linked lists

Methods to build binary trees

- Constructors
  - BinaryTree(): creates an empty tree
  - BinaryTree(Object rootItem): creates a tree with a root
  - BinaryTree(Object root, BinaryTree lt, BinaryTree rt): creates a tree with a root and left & right subtrees
- Attach things to the tree (root must already exist)
  - attachLeft/Right (Object newItem): attach an object to the left or right of the root
  - attachLeft/RightSubtree (BinaryTree tree): attach an entire tree to the root
  - attachLeft() could be done by creating a new subtree and attaching it with attachLeftSubtree()…
- Exceptions thrown for
  - Non-existent root
  - Trying to attach something on top of an existing subtree
Methods to take trees apart

- Often, necessary to take a tree apart
  - Make it better (more balanced)
  - Delete an item
- This can be done with
  - BinaryTree detachLeft/RightSubtree (): detaches a subtree from the root, and returns it
  - Left or right node set to null
- Informational methods
  - Object getRootItem ()
  - void setRootItem (Object newItem)
  - boolean isEmpty()
- How can we use these methods to build up a tree?

Create a simple tree

```java
bt = BinaryTree ("A");
b.t.attachLeft ("B");
ct = BinaryTree ("C");
ct.attachLeft ("D");
bt.attachRightSubtree (ct);
```
Iterators

- Provide a general way of traversing a tree
  - Can’t use internal types like TreeNode!
  - Instead, use an iterator
- An iterator is a class whose purpose is to allow other structures to be “read” in order
  - Example (sort of): Tokenizer
- An iterator supports a set of methods
  - Constructor: specifies the data structure to iterate over
  - hasNext(): true if there is another object to iterate to
  - next(): returns the next object in the traversal

Traversing a binary tree

- Trees can be traversed in three orders
  - Pre-order: L, R, root
    - B, F, D, E, C, A
  - In-order: L, root, R
    - B, A, F, D, C, E
  - Post-order: root, L, R
    - A, B, C, D, F, E
- Order chosen depends on
  - What the tree is being used for
  - What the traversal is supposed to accomplish
- Traversal is done recursively!
  - Treat L, R as trees in their own right
  - Recursively visit them