Sorting

- Examples used doubles or Strings
- Work with any class with ordering operator
  
  ```java
  interface Comparable<T>
  int compareTo(T other)
  ```

  - returns negative if self < other,
  - 0 if equal,
  - positive if self > other

Classes with ordering

- Classes with ordering written as:
  - class C implements Comparable<C>
  - Means must have method
    ```java
    public int compareTo(C other) {...}
    ```

- Collections class contains
  - public static <T extends Comparable<T>>
    ```java
    void sort(List<T> list)
    ```

Ordered Association

- Earlier talked about:
  ```java
  public class Association<K,V> {
    protected K theKey; // key of the key-value pair
    protected V theValue; // value of key-value pair
  }
  ```

- Now want associations where can order by key
ComparableAssociation

public class ComparableAssociation<K extends Comparable<K>, V>
        extends Association<K, V>
        implements Comparable<ComparableAssociation<K, V>> {
    public ComparableAssociation(K key, V value) {
        super(key, value);
    }
    public int compareTo(ComparableAssociation<K, V> that) {
        return this.getKey().compareTo(that.getKey());
    }
    ...
}

Now can use in sort!

Comparators

- java.util.Comparator interface in Java:

    public interface Comparator<T> {
        // returns negative if o1 < o2,
        // 0 if o1 == o2,
        // positive if o1 > o2
        // in whatever ordering is being supported by object.
        int compare(T o1, T o2);
    }

Way of Comparing Strings

public class TrimComparator implements Comparator<String> {
    // pre: s1 and s2 are strings
    // post: returns negative, zero, or positive depending on relation
    //        between trimmed parameters.
    public int compare(String s1, String s2) {
        String s1trim = s1.trim();
        String s2trim = s2.trim();
        return s1trim.compareTo(s2trim);
    }
}

Using Comparators

- Classes supporting sort or other operations using comparisons generally have two versions:
- From Collections class:
  - static <T extends Comparable<T>> void sort(List<T> list)
  - static <T> void sort(List<T> list, Comparator<T> c)
  - Actual types a bit more general (and complex).
Ordered Structures

• See text for discussion of operations on ordered structures
  • E.g., find, add, etc.

Trees

Definition

• Def: A tree is either
  • empty or
  • consists of a node, called the root node, together with a collection of trees, called its subtrees. These trees are disjoint from each other and the root.

MoreDefs

• An edge connects a node to its subtrees.
• The roots of the subtrees of a node are said to be the successors or descendants of the node.
• Nodes without successors are called leaves. The others are called interior nodes.
• All nodes except root have unique predecessor.
• A collection of trees is called a forest.
Example: Binary Search Tree

K, C, A, N, B, V, F, U, D, H, M

Expression Tree

\[ [A*(B-C)]+(D/E) \]

Family Tree Terminology

- *Parent* node is directly above *child* node: K is parent to C, N.
- Sibling node has same parent: A, F
- K is ancestor of B
- B is descendant of K

More Terminology

- Simple path is series of distinct nodes s.t. there is edge between successive nodes.
- Path length = # edges in path
- Height of node = length of longest path to a leaf
- Height of tree = height of root
- Depth of node is length of path from root to that node
- Degree of node is # of children

Node plus all descendants gives subtree
More Terminology

- Level of node defined recursively:
  - Root is at level 0
  - Level of any other node is one greater than level of parent
- Level of node is also length of path from root to the node.

Binary tree has all nodes of degree $\leq 2$.

Counting

- Lemma: If $T$ is a binary tree then at level $k$, $T$ has $\leq 2^k$ nodes.

- Theorem: If $T$ has height $h$, then # nodes in $T \leq 2^{h+1} - 1$.
- Equivalently, if $T$ has $n$ nodes then $n - 1 \geq h \geq \log(n+1) - 1$