What is a Tree?

Non-linear data structure
- Hierarchical arrangement of data

Has components named after natural trees
- root
- branches
- leaves

Drawn with root at the top

Components of a Tree

Node: stores a data element
Parent: single node that directly precedes a node
  - all nodes have 1 parent except root (has 0)
Child: one or more nodes that directly follow a node
Ancestor: any node which precedes a node
  - itself, its parent, or an ancestor of its parent
Descendent: any node which follows a node
  - itself, its child, or a descendent of its child

More Tree Terminology

Leaf (external) node: node with no children
Internal node: non-leaf node
Siblings: nodes which share same parent
Subtree: a node and all its descendents
  - ignoring the node’s parent, this is itself a tree
Ordered tree: tree with defined order of children
  - enables ordered traversal
Binary tree: ordered tree with up to two children per node

Examples of Trees

Directory tree
- Organizes directories and files hierarchically
- Directories are internal nodes, files are leaf nodes (usually)
Class hierarchy
- Object is root, other classes are descendents
Decision tree
- Binary tree
  - Path taken determined by boolean expression
Expression tree
- Operators are internal nodes, variables and constants are leaf nodes

Comparison of Tree and List

<table>
<thead>
<tr>
<th></th>
<th>List</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>head</td>
<td>root</td>
</tr>
<tr>
<td># before</td>
<td>1 (prev)</td>
<td>1 (parent)</td>
</tr>
<tr>
<td># after</td>
<td>1 (next)</td>
<td>&gt;= 1 (children)</td>
</tr>
</tbody>
</table>
Tree methods

- `root()`: returns root
- `parent(v)`: returns parent of v
- `children(v)`: returns iterator of children of v
- `size()`: returns number of nodes
- `elements()`: returns iterator of all elements
- `positions()`: returns iterator of all positions/nodes
- `swapElement(v, w)`: swaps elements at two nodes
- `replaceElement(v, e)`: replaces element of a node

Tree query utility methods

- `isInternal(v)`: test if node is internal
- `isExternal(v)`: test if node is external
- `isRoot(v)`: test if node is root

Depth

- Depth of v is numbers of ancestors (excluding v)
  - depth of root is 0
  - depth of node is depth of parent plus 1

  ```java
  public static int depth(Tree T, Node v) {
    if (T.isRoot(v)) return 0;
    else return 1 + depth(T, T.parent(v));
  }
  // running time? O(d(v))
  ```

Height

- Height of v is maximum path length of subtree
  - Height of leaf node is 0
  - Height of internal node is maximum height of children + 1
    - Height of a tree is height of root or maximum depth of a leaf

  ```java
  public static int height1(Tree T) {
    int h=0;
    PositionIterator it = T.positions();
    while (it.hasNext()) {
      Position v = it.nextPosition();
      if (T.isExternal(v))
        h = Math.max(h, depth(T, v));
    }
    return h; // running time? O(n^2)
  }

  public static int height2(Tree T, Position v) {
    if (T.isExternal(v)) return 0
    else {
      int h=0;
      PositionIterator children = T.children(v);
      while (children.hasNext()) {
        Position v = children.nextPosition();
        h = Math.max(h, height2(T, children.nextPosition()));
      }
      return 1 + h; }
    // running time? O(n)
  }
  ```
Traversal

Ordered way of visiting all nodes of tree
Converts hierarchy into a linear sequence

Preorder Traversal

Visit node, then visit children

```java
public void preorder(Tree T, Position v) {
    visit(v);
    PositionIterator children = T.children(v);
    while (children.hasNext()) {
        preorder(children.nextPosition());
    }
}
```

Preorder Example

Preorder Example

Preorder Example

Preorder Example
Preorder Example

Preorder Example

Preorder Example

Preorder Example

Preorder Example

Preorder Example
Preorder Example

Postorder Traversal

Visit children, then visit node

public void postorder(Tree T, Position v) {
    PositionIterator children = T.children(v);
    while (children.hasNext())
        postorder(children.nextPosition());
    visit(v);
}

Postorder Example
Postorder Example
Postorder Example

In-class Exercises...

(Paper and pencil recommended)
**Postorder Letter Scramble**

```
T
e  a  l
```

**Binary Tree**

Each node has no more than 2 children

- Proper binary tree: each node has either 0 or 2 children

**Binary Tree ADT**

- `leftChild(v)`: returns left child of `v`
- `rightChild(v)`: returns right child of `v`
- `sibling(v)`: returns sibling of `v`

**Binary Tree Traversal**

- **Preorder**: node, left, right
- **Postorder**: left, right, node
- **Inorder**: left, node, right

**Euler Tour Traversal**

Generalizes preorder, inorder, and postorder

Visit each internal node 3 times

```java
public void eulerTour(Tree T, position v) {
    visitPre(T, v);
    if (T.hasLeft(v))
        eulerTour(T, T.leftChild(v));
    visitIn(T, v);
    if (T.hasRight(v))
        eulerTour(T, T.rightChild(v));
    visitPost(T, v); return;
}
```
Template Method Pattern

Implement template or skeleton method for high-level algorithm

Extend the template’s class to override lower-level methods

EulerTour example
  * Override methods for visitPre, visitIn, and visitPost

Useful Binary Tree Definitions

Level $d$: All nodes in a binary tree at depth $d$
  * Maximum of $2^d$ nodes in level $d$

Complete binary tree: tree of height $h$ with $2^h$ leaf nodes
  * $2^h-1$ internal nodes
  * $2^{h+1}-1$ total nodes

Binary Tree Properties

(proper) Binary tree $T$ of height $h$
  * $h+1 \leq$ external nodes $\leq 2^h$
  * $h \leq$ internal nodes $\leq 2^h-1$
  * $2h+1 \leq$ total nodes $\leq 2^{h+1}-1$
  * $\log(n+1)-1 \leq h \leq (n-1)/2$
  * external nodes = internal nodes + 1

Implementing Binary Tree

Linked
  * Each node references left, right, and parent as well as element

Vector-based
  * Number nodes in level order
  * Store nodes at rank according to number
    * Storage allocated for entire complete tree