Trees

- Non-linear data structure
  - not just predecessor and successor relations
  - ancestor, descendant, parent, child relations.
- Natural orginizing structure for hierarchy
- Many examples can be found.
- Unix file structure, process structure, etc.

Trees

- A tree T is a set of nodes defined by the following recursive definition:
  - There is a special node r called **root**
  - A partition of remaining nodes into k disjoint subsets $T_1, T_2, \ldots, T_k$ of T each of which is a tree and referred to as **subtrees** of T.
  - The roots $r_1, r_2, \ldots, r_k$ of subtrees $T_1, T_2, \ldots, T_k$ respectively are all **children** of the node r.
- Except for root r every node v of a tree T has an immediate predecessor called its **parent**.

Trees

- Tree is ordered if there is an ordering among the children of all nodes.
- Two children of same parent called **siblings**
- A node which is not a parent of any node is called a **leaf** node.
- Nodes other than leaf nodes are called **internal** nodes.
- A node v is an **ancestor** (**descendant**) of a node u, provided either
  - $v = u$, or
  - v is an ancestor (descendant) of the parent of u

Binary trees

- Each internal node of a **binary** tree can have at most two children
- In a **strictly binary** tree each internal node has exactly two children
- The two children of an internal node in a binary tree are distinguished as **left** and **right** child respectively.
- the subtrees rooted at left child and the right child are referred to as **left subtree** and the **right subtree** respectively.
- The root of a tree is at **level** 0
- All the children of an internal node at level i are at level $i + 1$

Binary trees

- The **height** of a tree is equal to the maximum of the levels of its leaf nodes.
- **Depth** of a node is synonymous to its **level**.
- More precisely, **Depth** of any node is equal to its distance from the root.
- Depth of the root equal to 0.
- The **height** of a node is equal to the maximum depth of its leaf descendants.

Binary trees

- A **full** binary tree is a strictly binary tree in which all leaves occur only at the bottom most level.
- A **complete** binary tree is a binary tree in which
  - all leaves occur either in the bottom most level or one level up,
  - if all leaf nodes at bottom-most level are deleted from a complete binary tree then a full binary tree is produced,
  - all the internal nodes at one level up the bottom most level appear to left of all the leaves at that level.
- A full binary tree is also a complete binary tree.

Properties of binary trees

- A strictly binary tree of height h has following properties
  - The number of leaf nodes at most 2^h and at least $h+1$
  - The number of internal nodes at most 2^h-1 and at least h
  - The total number nodes is at most $2^{h+1}-1$ and at least $2h+1$
  - $\log(n+1)-1 \leq h \leq (n-1)/2$.
- The number of leaf nodes in a strictly binary tree is equal to one more than the number of internal nodes.
  1. repeated application of *contraction* operation one external node + its parent
  2. inductive proof.

Tree Methods

- Stores elements at position.
- Positions are nodes or places where elements are held.
- Neighbouring positions define parent-child relations.
- `element()`: returns an object at this position.
- **Accessor Methods**: accept and return positions.
  - `root()`
  - `parent(v)`
  - `children(v)`
- **Query Methods**: accept and return positions.
  - `isInternal(v)`
  - `isLeaf(v)`
  - `isRoot(v)`

Tree Methods

- **Constructor methods**: create new empty trees
- **Update methods**: accepts trees (and elements) and return modified trees or elements.
  - `replaceElement(v, e)`: returns object at position v.

Generic Methods

- **size()**: returns number of nodes.
- **elements()**: returns an iterator of Objects in tree.
- **positions()**: returns an iterator of all positions (nodes) in tree.
- **swapElements(v,w)**: swaps contents of v and w positions.
- **replaceElements(v,e)**: replaces contents of v by object e.

An Iterator

- Iterator is an object that provides an item-wise retrieval starting with `first item` by calling `next()` method.
- Other methods of an iterator are: `hasNext()` and `remove()`.
- `java.util` package provides some built-in function for managing a collection: set, sequence, vector or trees. These have iterators with generic methods as appropriate for the data structure.