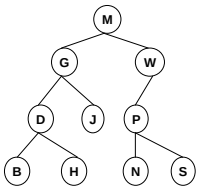
Tree Terminology

- M is the *root* of this *tree*
- G is the *root* of the left *subtree* of M
- B, H, J, N, and S are *leaves*
- N is the *left child* of P; S is the *right child*
- P is the *parent* of N
- M and G are *ancestors* of D
- P, N, and S are *descendents* of W
- A collection of several trees is called a *forest*



Searching in a BST

```
class BST {
    private Object key, value;
    private BST left,right;

    public Object get (Object k) {
        if (k.equals(key)) return value;
        if (k.before(key)) {
            if (left == null) return null;
            else return left.get(k);
        } else {
            if (right == null) return null;
            else return right.get(k);
}}}
```

- Note that before() is not part of Java, but equals() is
- Can also do a simple iterative version which would probably be somewhat more efficient

Deleting from a BST

Cases:

- Delete a leaf
  - easy
- Delete a node with just one child
  - delete and replace with child
- Delete a node with two children
  - delete node's successor
  - write successor's data into node

- How do we find the successor?
- The successor always has at most one child. Why?
- Would work just as well using predecessor instead of successor

BST Performance

- Time for put(), get(), update(), remove() is O(*h*) where *h* is the height of the tree
- How bad can *h* be?
- Operations are fast if tree is *balanced*

- How balanced is a random tree?
  - If items are inserted in random order then the expected height of a BST is O(log *n*) where *n* is the number of items
- If deletion is allowed
  - Tree is no longer random
  - Tree is likely to become unbalanced

Analysis of Random BST

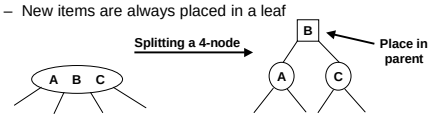
- Only the number of items and their order is important
  - Can restrict our attention to BSTs containing items {1,…, *n*}
- We assume that each item is equally likely to appear as the root
- Define H(*n*) ≡ expected height of BST of size *n*
- If item *i* is the root then expected height is
  1 + max { H(i-1), H(n-i) }
  We average this over all possible *i*
- Can solve the resulting recurrence (by induction)
  H(n) = O(log n)

Why use a BST instead of a Hashtable?

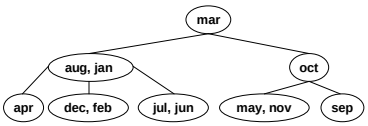
- If we use a *balanced* BST scheme then we achieve guaranteed *worst-case* time bounds
- There are some operations that can be efficient on BSTs, but very inefficient on Hashtables
  - getMin, getMax
  - select(k) // find the *k*-th element
  - report-elements-in-order
- Note that balanced BST schemes can be difficult to implement
  - But there are lots of reliable codes for these schemes available on the Web
  - Java 1.2 includes a balanced BST scheme among its standard packages

234-Trees

- Nodes have 2, 3, or 4 children (1, 2, or 3 data items)
- All leaves are at the same level
- Basic rule for insertion: We hate 4-nodes
  - Split a 4-node whenever you find one while coming *down* the tree
  - Note: this requires that parent is not a 4-node
  - New items are always placed in a leaf



Example 234-Tree: Months



- The months were inserted in calendar order; alphabetical order is used within the tree

Deleting from a 234-Tree

- Deleting is more complicated than inserting
  - For delete, we hate 2-nodes
  - As in BSTs, cannot delete from a nonleaf
  - Use same BST trick: delete successor/predecessor and recopy its data
- To destroy a 2-node
  - First, attempt to steal from a sibling to make a 3-node (must pass item through parent to ensure order remains valid)
  - If that doesn't work then sibling is also a 2-node; combine with sibling to steal an item from the parent to make a 4-node
  - Note: parent is never a 2-node, so can always steal from parent if necessary

234-Tree Analysis

- Time for insert or get is proportional to tree's height
- Let *N* be the number of nodes, *n* be the number of items, and *h* be the height
- Define *h* so that a tree consisting of a single node is height 0
- It's easy to see $1+2+4+\ldots+2^h \le N \le 1+4+16+\ldots+4^h$
- It's also easy to see $N \le n \le 3N$
- Using the above, we have $n \ge 1+2+4+\ldots+2^h = 2^{h+1}-1$
- Rewriting, we have $h \le \log(n+1) - 1$ or $h = O(\log n)$
- Thus, Dictionary operations on 234-trees take time O(log n) in the worst case