Red-Black Trees
Structure property:
   - Every node is “colored” either red or black.
   - The root is black.
   - If a node is red, its children must be black. (A leaf can be red.)
   - Every path from a node to a null reference must contain

Notes:
• Uses the standard rotations, plus some coloring operations, to maintain structure.
• Worst case find, insert, delete: O(log n)
• Has nice top-down, non-recursive implementation.
• Java uses top-down red-black trees.

Announcements (5/5/08)
• Project 2B due Wednesday night.
• Midterms will be graded this week, returned to you in section.
• Homework 4 due Friday, beginning of class.
Treaps

Order property:
• Each node has a randomly assigned priority value, in addition to its key value.
• Tree has both BST and heap order!

Orange = low priority value, Yellow = high priority value

Notes:
• Insert:
  – insert as leaf
  – assign random priority
  – percolate up with rotations.
• Delete:
  – set priority to infinity
  – percolate down with rotations
  – once it’s a leaf, delete.
• Average case insert, delete, find: $O(\log n)$.

There are many more…

There are many more sorted Dictionary ADTs:
• AA trees
• Scapegoat trees
• Skip lists
• …

Experimental evaluation

Pfaff [1] tried out 20 different BSTs on several common systems applications.

Findings:
• Random inputs:
  – BSTs perform best (low overhead)
• Mostly random inputs, occasional ordering:
  – Red-Black trees best
• Ordered inputs, random finds
  – AVL trees best
• Ordered inputs, ordered or clustered finds:
  – Splay trees best (though not best for interactive situations)
Special queries

The BSTs we’ve discussed were only required to support the Dictionary ADT: find, insert, delete.

But, other operations that leverage the sorted data can also be efficient on BSTs:
- findMax
- findMin
- findMedian
- findRange (i.e., keys within a certain range)
- printSorted

Bibliography