Red-Black Trees

• Motivation:
  – The good: 2-3 and 2-3-4 trees produce balanced trees of height bounded by log(n)
  – The bad: without extreme efforts in OO design 2-3 and 2-3-4 trees typically waste a lot of memory on unused items and children references.
  • 2-nodes are really 4-nodes with 2/3 of memory for that node being unused.
• Red-Black trees are conceptually identical to 2-3-4 trees.

Here is a 4-node

```
  10  20  35
   α   β   γ   δ
```

“4-node” Represented as 2-nodes in a Red-Black Tree

```
  20
   10   35
       α   β   γ   δ
```
• All nodes in a Red-Black tree are 2-nodes. Just like BSTs.
• 2-nodes are simply represented as they would be in a BST.
• 3-nodes and 4-nodes are represented by separate 2-nodes (one node for each item) and these nodes are “bound” to each other using “red” links
• Examples of 3-nodes…

Either way is fine.
Implementation

- Implementation of red “links”
- “links” or children references do not have the ability to contain information about their “color”
- The book would like to represent 4-nodes like the example given here:
- But that isn’t practical.
• Instead we represent red “links” by coloring the node the link references red.
• This can be done by simply adding a boolean to the BST Node class:

```java
class Node {
  Object stored;
  Node left, right;
  boolean coloredRed;
}
```
• If you take any 2-3-4 tree you can illustrate it as a Red-Black Tree…
• Figure 12-20 from the book as a 2-3-4 tree
• Same tree stored as a Red-Black tree.

• There are 15 other valid variations.
• Why is it conceptually the same tree?
• We could then treat a Red-Black tree exactly like a 2-3-4 tree. We can tell if a node is really a 4-node by checking if both its left and right children are colored red.
• A node is a 3-node if its not a 4-node and has one red child.
During insertion 4-nodes get “split” by simply recoloring the nodes...

Simply splitting a 4-node works to correct the root because the root node can simply be recolored black.
Splitting 4-nodes and adopting the middle.

- Non-root nodes need to have the middle item “pushed” up to and included with the parent: (We can’t just recolor the top node black.)
• Works great for 2-nodes as was shown.
• Also good for “easy” 3-nodes:
• Some three nodes are harder:

\[
\begin{align*}
32 & \quad 33 & \quad 34 \\
50 & \quad 60 \\
\alpha & \\
32 & \quad 33 & \quad 34 \\
60 & \\
\beta & \\
32 & \quad 34 \\
\alpha & \\
32 & \quad 34 & \\
\beta & \\
32 & \quad 34 & \\
\alpha & \\
32 & \quad 34 & \\
\beta & \\
33 & \quad 50 & \quad 60 \\
\alpha & \quad \beta & \\
32 & \quad 34 & \\
\end{align*}
\]

What is that?

It's sort of this (heights are wrong)

But that's not a truly valid 4-node. (Red nodes cannot have red children.)
• Adding an item to a 2-node makes it a three node…

• It’s the same as adding a node to a BST tree. The node is added as a leaf.

• We start its color as red though.
• Adding a value to a 3-node to make it a 4-node isn’t quite a simple.

\[
\begin{array}{c}
32 & 33 \\
\hline
34
\end{array}
\]

= 33

\[
\begin{array}{c}
32 \\
\hline
34
\end{array}
\]

Looks Easy!
• But it's not always that easy…

= 

• What the heck kinda node is that?
• What we need to do is rotate this subtree.
• Right rotations:
• Here’s a possible configuration that wants to be a four node but can’t be right rotated to correction in one step:

Left rotation around 31  
Right rotation around 32
• Red-Black tree as a more formal definition (Warning: this is not in the text):

• We can abstract the idea of Red-Black trees away from conceptually thinking of them as 2-3-4 trees.

• This is done making a formal set of rules for the creation and maintenance of Red-Black Trees.
The Rules

• Every node is either red or black
• Every null reference is considered to be a link to a black node
• Red nodes must have ONLY black children
• Every simply path from the root to any leaf contains the same number of black nodes as any other such simple path.
Operations

• The balancing of Red-Black trees is done by maintaining the rules through the use of Left and right rotations and recoloring.

\[ \text{Right-Rotate}(y) \]

\[ \text{Left-Rotate}(x) \]
• Introduction to Algorithms (Cormen, Leiserson & Rivest) present a much more detailed analysis of Red-Black trees from this perspective.

• It is sufficient to implement a Red-Black tree by treating it exactly the same as a 2-3-4 tree taking into account the colors of node in determining whether you are about to step into a 4-node (for insertion) or a 2-node (for deletion).

• You also need to make sure that no red node has red children.