CSC 403 Sections 901,902,910,911 Spring 2015
Red-black Trees

• Motivation
  o Implementing 2-3 trees is painful
  o Total of 6 cases for insertion (with 12 cases for deletion)
  o Simplification: represent a 2-3 tree as a red-black tree

• Advantages of red-black trees over 2-3 trees
  o reduce code complexity
  o minimize or eliminate space overhead
  o single top-down pass

• Definition of a red-black tree
  o A red-black tree is a binary search tree whose nodes are colored either red or black
  o Each 2-3 tree has a corresponding unique representation of a red-black tree
  o 2-nodes are the same as in 2-3 trees
  o 3-nodes are represented as a parent node colored black and its left child colored red
  o Examples:

![Diagram of a red-black tree with 3-nodes and 2-nodes](image_url)
**Alternate definition**

A red-black tree is a binary search tree in which

1. Nodes are colored either red or black
2. All red nodes are left children
3. The tree has perfect *black balance*, meaning that every path from the root to a null link contains the same number of black nodes
4. recall the definition of a path -- a path from $x$ to $y$ a sequence of nodes $(x, n_1, n_2, ..., y)$ such that there is an edge between consecutive nodes in the path
5. No path contains consecutive red nodes

**Note:** red-black trees are **not** perfectly balanced

**Representation of 4-nodes in red-black trees**

- Two possible red-black trees correspond to 4-nodes:
o Both are transformed to the same subtree:

![Tree Diagram]

o Transformations are done with 3 operations:
  - Left (counterclockwise) rotate
  - Right (clockwise) rotate
  - Color flip

- **Red-black rotations**
  o To maintain the above properties, portions of red-black trees must sometimes be rotated in order to preserve the properties listed above
  o Two kinds of rotations: clockwise and counterclockwise
  o Clockwise rotation:

```java
Node rotateRight(Node h)
{
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.color = h.color;
    h.color = RED;
    x.N = h.N;
    h.N = 1 + size(h.left) + size(h.right);
    return x;
}
```

![Clockwise Rotation Diagram]
Counterclockwise rotation:

- Algorithm for insertion
  - To add to an empty tree, assign a new node to be the root. Its color is black.
  - For a non-empty tree, add a new node to the tree just as with a binary search tree (always add a leaf). Color the new node red.
  - If in the path from the root to the new node there is a black node \( n \) with two red children, then flip their colors (i.e., \( n \) becomes red, and its children become black)
    - However, \( n \) stays black if it is the root
  - If the new node is still red and is a right child, perform counterclockwise rotation on the subtree rooted at the new node's parent
  - If the path from the root to the new node's (or its left child) now ends with 2 red nodes, perform a clockwise rotation of the subtree rooted at the new node's grandparent
    - It's the new node's left child if a counterclockwise rotation was performed at insertion
  - These rotations and recolorings may require additional adjustments further up the tree
Examples

- Sedgewick, p. 440

- Starting with an empty tree, insert 5, 3, 7, 2, 4, 11, 1, 0, 12, 6, 9, 8, 10, 13, 14
Add 5, 3, 7, 2, 4

Add 1, 0

Add 12, 6

Add 9

Add 8, 10, 13, 14
• **Implementation**

The implementation in the text is for a **symbol table**. Symbol tables are like Python dictionaries, or Java maps. Each entry in a symbol table consists of a *key* and a *value*. In the text example, the keys are letters (S E A R C H E X A M P L E) and the values are the last position of each letter. For example, the value under the key 'R' is 3, since the 'R' is at position 3 in the string. Similarly, the value under the key 'E' is 12, since the letter E last appears in position 12 in the example.

In the text implementation, the keys are used to determine where a Node belongs in the tree. The values are retrieved by using the `get` method.

The version which we will discuss in class is a red-black implementation of a Set. Sets have values but not keys, and behave similarly to Python or Java sets.