Balanced Search Trees Worksheet

The Trees

2-3-4 Trees

Idea: We stuff more than one value into a node, so we can keep our tree short.
Like a binary search tree, but each node can have **2, 3, or 4** children. This means each node can store 1-3 values!

`Insert`: To insert a value n,

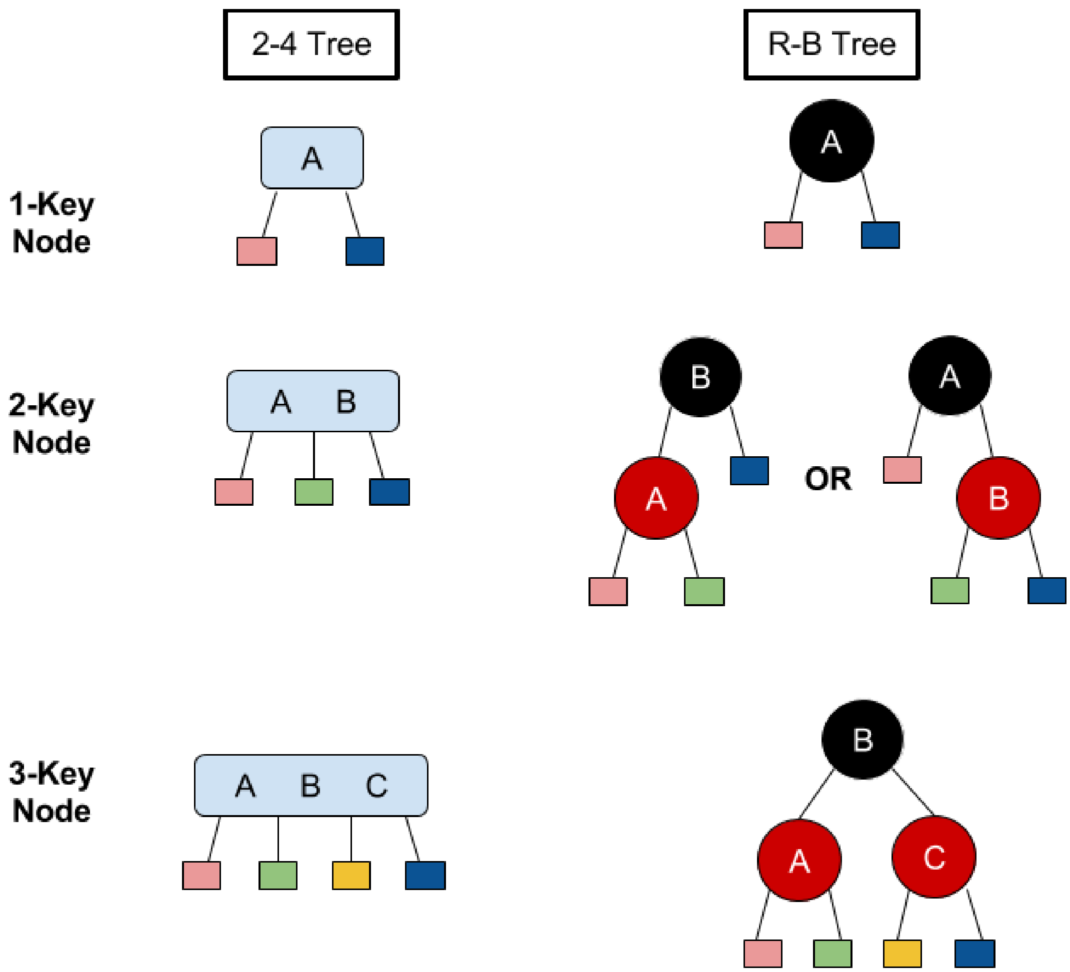
1. Traverse down the tree from root to leaf to find the correct place to put n
2. If you ever encounter a node with 3 items as you traverse down the tree, kick up the middle item up.
3. Once you are at a leaf, insert n in its correct spot in the leaf node.

Exercise: `insert(5)`

Red-Black Trees

Idea: 2-3-4 trees are hard to implement, so we make a type of tree that can represent a 2-3-4 tree, but is also much easier to code.
RB trees come directly from 2-3-4 trees! They were created as a way to represent 2-3-4 trees so that it could be easy to code. A red-black tree is a regular binary search tree. However, we ”color” some of the nodes red, which will help with our balancing operations.

The conversion process:



Exercise: convert the 2-3-4 tree on the first page (before `insert(5)`) into a RB tree

Splay Trees

Idea: We have a binary search tree, but any time we do some `get`, `put`, or `remove` operation on some node n, we **splay** the node n.
Splaying a node n means we do a series of rotations to make n the root of the tree. Although this isn't the same kind of balancing that 2-3-4 trees do, it allows for a special feature: **locality of reference**.

`splayNode`: To splay some node n, we must do as many rotations on n needed until it becomes the root. For any node n, we consider the position of its parent node p, and its grandparent node g to figure out what kind of rotation we need to do. Here are three cases you'll run into when you rotate n:

1. Zig
   (a) Situation:
   (b) Operation:
2. Zig-Zag
   (a) Situation:
   (b) Operation:
3. Zig-Zig
   (a) Situation:
   (b) Operation:

1 Practice Exam Problems

1. Given a 2-3-4 tree containing N keys, how would you obtain the keys in sorted order in worst case $O(N)$ time? We dont need actual pseudo code; an clear description will do

2. If a 2-3-4 tree has depth h (that is, the leaves are at distance h from the root), what is the maximum number of comparisons done in the corresponding red-black tree to find whether a certain key is present in the tree?

Solutions for #1 and #2 available on Fall 2016 website:
https://inst.eecs.berkeley.edu/ cs61b/fa16/materials/disc/discussion11sol.pdf