**Balanced Search Trees**
- rotations
- AVL trees
- 2-3 trees

**Binary Search Tree (Best Case)**

Tree is balanced.
Depths of all nodes are $O(\log n)$.

How to insert a key e.g. 49?

**Binary Search Tree (Worst Case)**

If keys are inserted in order then the BST behaves as a linked list and worst case searching for a key is $O(n)$.

We would like our BSTs to be "balanced". There are several ways to define "balanced".
The main goal is to keep the depths of all nodes to be $O(\log n)$.

**Suppose we have**

- $k_1, k_2$ are keys
- $A, B, C$ are sub trees (of unspecified shape)

$A < k_1 < B < k_2 < C$

All keys in A are less than key $k_1$. $k_1$ is less than all keys in $B$, which are less than $k_2$. $k_2$ is less than all keys in $C$. 

---

**Background**
- comp 250 binary search trees

**Resources for this lecture**
- rotations, AVL trees
  (slides only)
- 2-3 trees - Sedgewick 1

https://class.coursera.org/algs4partI-003/lecture/49
Rotation

right rotation

A < k₁ < B < k₂ < C

right rotation

rotateRight( root ){
    newRoot = root.left
    root.left = newRoot.right
    newRoot.right = root
    return newRoot
}

rotateLeft( root ){
    ... 
    ... 
}

Exercise:
What if $k_2$ is not the root of the tree is, what if $k_2$ has a parent?

Exercise: rotate Right ($k_2$) maintains the BST property for the whole tree. Why?

Balanced search trees

- AVL trees
- red-black trees
  used in Java's TreeMap class
  If you are interested, see
  https://class.coursera.org/algs4partI-003/lecture/50

- 2-3 trees

Recall from COMP 250:
the height of a node in a binary tree is the length of longest path from that node to a leaf.

AVL tree
[Adelson-Velskii, Landis 1962]

Balance Condition:
For any node in an AVL tree, the height of its left subtree differs by at most 1 from the height of its right subtree.

Examples: valid AVL or not?

- yes
- no
Because of time constraints, we will cover only insertion into AVL trees, but we ignore deletion.

(Recall from COMP 250 that deletion from a BST is trickier than insertion. This is the case for AVL trees too. See wikipedia for details if you are interested.)

http://en.wikipedia.org/wiki/AVL_tree

Suppose we have an AVL tree and we do an insertion that causes the heights of the left and right subtrees of some node to differ by 2. How can we rebalance the tree?

There are four ways (two pairs of ways) that the imbalance could have occurred, namely the insertion was:
- to the left subtree of the left child (outside)
- to the right subtree of the right child (outside)
- to the right subtree of the left child (inside)
- to the left subtree of the right child (inside)

**Answer:** rotate Left($k_i$)

**Question:** How to rebalance $k_i$?

**BEFORE**

**AFTER** (balanced)

---

c.g. Insertion was in tree C and extended C's height from $h$ to $h+1$, creating imbalance at subtree rooted at $k_i$ (but not $k_2$).
```

2-3 trees
(balanced search trees
that don't use rotations)

Exercise: How to rebalance $k_1$?

Each node of a 2-3 tree either has
1 or 2 keys
(called a "2 node" or "3 node", respectively, i.e. number of children)

Internal

Leaf

Searching (find) in a 2-3 tree uses same idea as BST.

Insertion of a key always occurs into a leaf node

https://class.coursera.org/algs4partI-003/lecture/49
```
The problem comes when the leaf node already is a 3-node (i.e. two keys present already) e.g. `insert(3)`

Split 4-node into two 2-nodes and move middle key (X) to parent.

Now we are balanced again!

Summary: Insertion in 2-3 tree

- two phases:
  - downward (search for leaf node where key is inserted)
  - upward (split if necessary)

Next lecture:
- hash tables
- Prepare by reviewing my COMP 250 notes on this topic.
  (or M. Blanchette's slides)