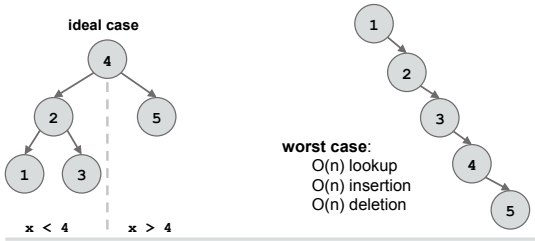
Recitation 9

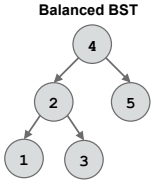
Tree Rotations and AVL Trees

Review: Binary Search Tree (BST)



worst case:
O(n) lookup
O(n) insertion
O(n) deletion

Make BSTs balanced!



worst case:
O(log n) lookup
O(log n) insertion
O(log n) deletion

If a BST becomes unbalanced, we can **rebalance** it in **O(log n).**

Review: definition of Height

```
public static int getHeight(TreeNode t) {
      if (t == null)
            return -1;
      return 1 + Math.max(getHeight(t.left),

      getHeight(t.right));
}
```

length of the longest path from a node to a leaf

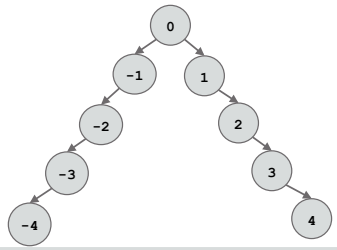
Definition of Balanced

```
public static boolean isBalanced(TreeNode t) {
      return t == null ||
                  Math.abs(getHeight(t.left) -
                   getHeight(t.right)) <= 1 &&
                  isBalanced(t.left) &&
                  isBalanced(t.right);
}
```

A tree is balanced if each of its subtrees is balanced and their heights differ by at most 1

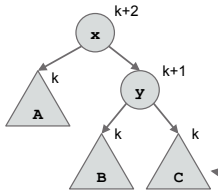
isBalanced: Recursion needed!

All subtrees need to be balanced!



Tree Rotations

Notation

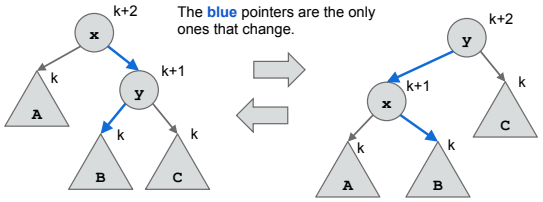


Inorder traversal:
`A x B y C`

Recall that the BST inorder traversal gives sorted order.

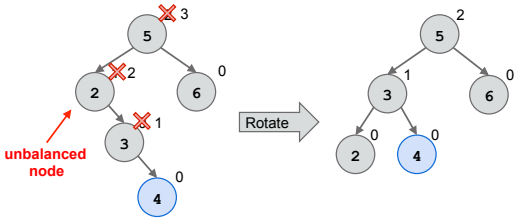
A subtree of height k

Rotations: Used to balance a BST

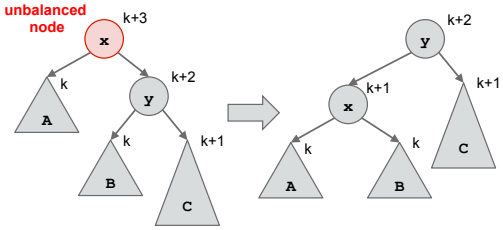


Inorder traversals are the same

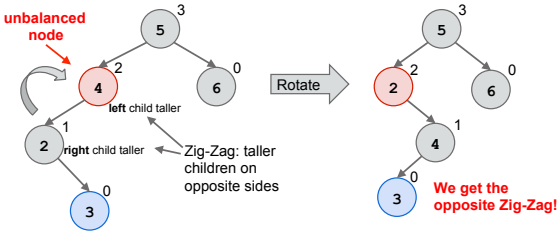
Rotations example



Rebalancing

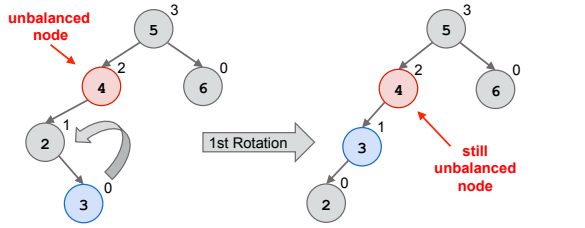


Problem: Rotating a Zig-Zag!

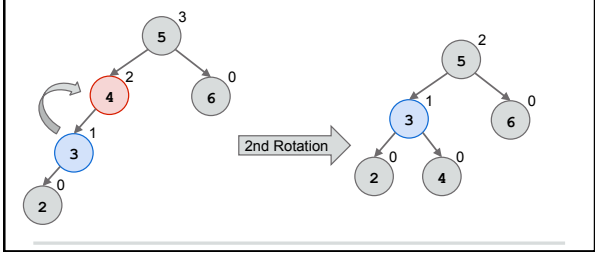


Zig-Zag: taller children on opposite sides

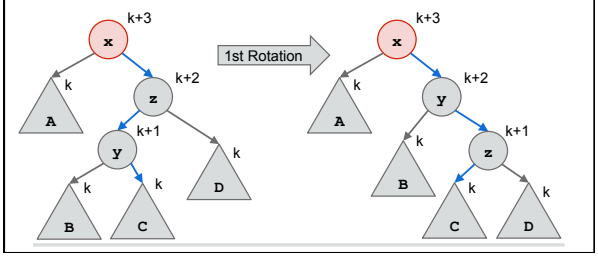
Double rotate



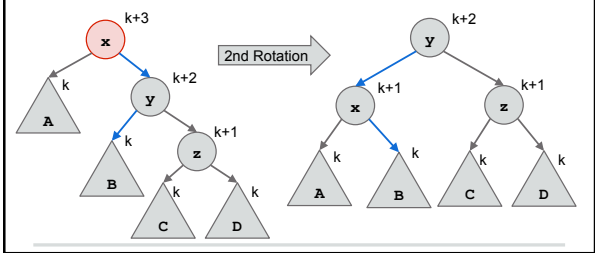
Double rotate



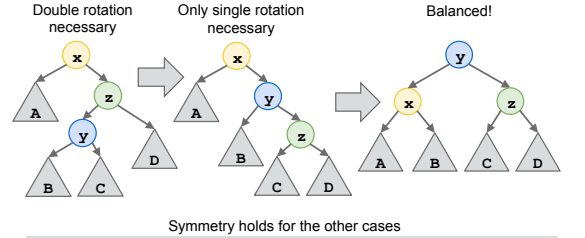
Rebalancing with double rotate



Rebalancing with double rotate

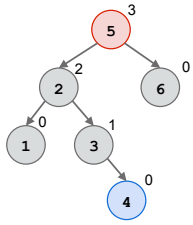


Summary of Rotations

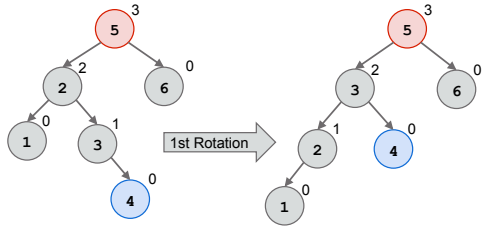


Symmetry holds for the other cases

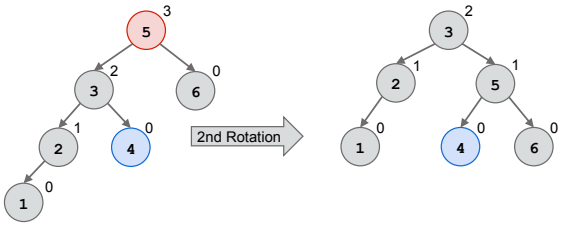
Question: What is the resulting tree?



Question: What is the resulting tree?



Question: What is the resulting tree?



AVL Trees

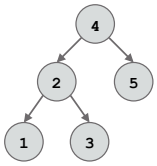
AVL Trees

Named after its two Soviet inventors, Georgy Adelson-Velsky and E. M. Landis, who described AVL tres in a paper in 1962.

First invention of *self-balancing BSTs*.
Later: red-black trees, splay trees, and others

AVL Tree

AVL Tree: self-balancing BST



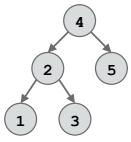
AVL invariant:
the height difference between its left and right children is at most 1

Lookup works the same as a normal BST lookup

worst case:
O(log n) lookup
O(log n) insertion
O(log n) deletion

Inserting an element

`insert(E elem)`



https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

Insert like a normal BST and if the AVL invariant is broken, do a single or double rotation to fix it

Localizing the problem:
1. Imbalance will occur only on the path from the root to the newly inserted node
2. Rebalancing should occur at the **deepest** node
3. Must search for possible imbalance all the way up to root

Why use AVL Trees?

If HashSets have a lookup of expected O(1), why use BSTs with an expected lookup time of O(log n)?

Depends on the problem:

1. Binary Search Trees are great at keeping elements in **sorted order.**
2. Key Ranges: How many words in the set start with k and end in z?
3. `findPredecessor(E elem)` and `findSuccessor(E elem)`
   - O(log n) for AVL Tree, expected case O(n) for HashSet
4. Better worst case lookup and insertion times

Prelim Information

1. Tree Rotations will *not* be tested on Prelim 2
2. You don't need to be able to write Tree Rotations code but can find it online if interested