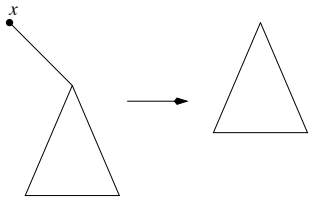
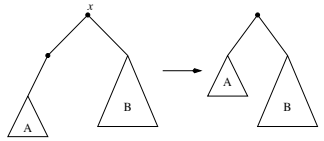
Balanced Binary Search

CS 62
Spring 2015
Kim Bruce & America Chambers

Remove node

- Remove topmost node.
- Easy cases:
  - no left subtree, or no right subtree -- easy, they are new tree

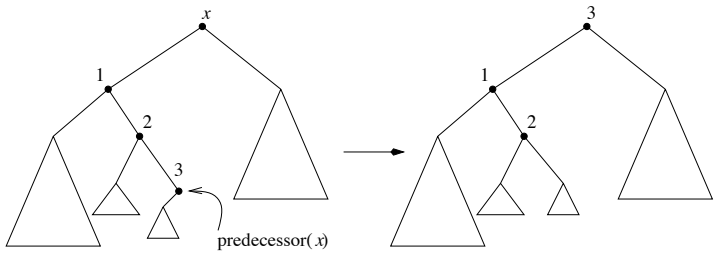




  - left child has no right subtree

General Case

- Left Child has a right subtree:



Remove method

- Locate element to be deleted
- RemoveTop of node rooted at element
- Hook up resulting tree as child of elt's parent.
- O(h), where h is height of tree.
  - O(h) to find,
  - Could be another O(h) to find predecessor
  - Constant to patch back together.

Complexity

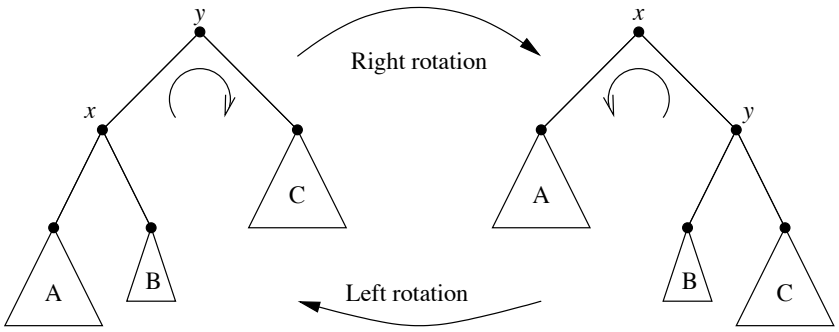
- Runtimes depend on the height of the tree.
- To achieve a O(log n) runtime we need to keep the tree “balanced”.

Random BST

- If values are inserted in random order, then the expected height is O(log n)
- This gives us a new O(n log n) sorting algorithm!
  1. Insert values into BST in random order
  2. Read values from BST in-order

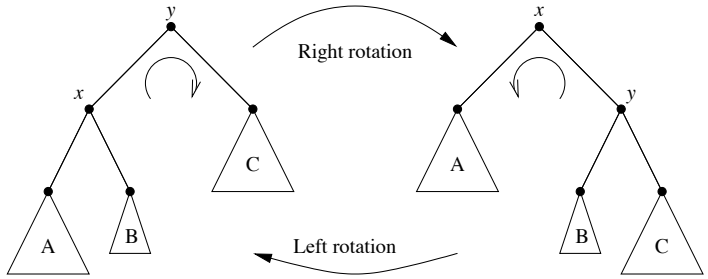
Tree Rotations

- Change the structure of the BST while preserving the ordering properties.



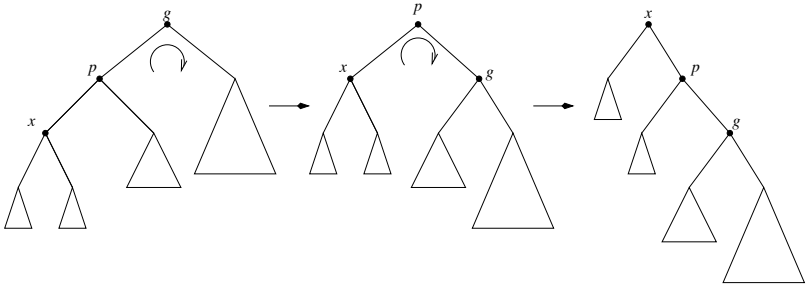
Rotating Trees

- Rotate x to root, while maintain BST structure
  - All nodes in subtree A go up one level, all in C go down one level, all in B stay same.
  - See code in BinaryTree



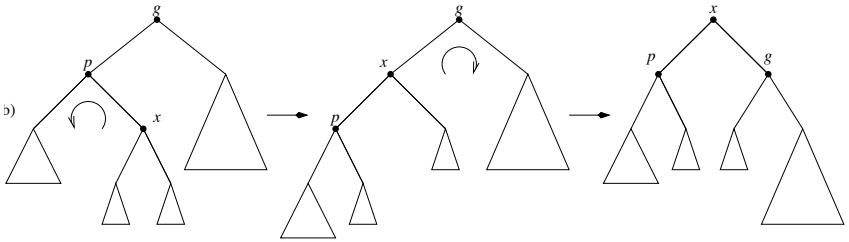
Shifting elements toward root

- Move x up two levels w/ two rotations
- If x is left child of a left child ...



Shifting elements toward root

- If x is a right child of a left child.



Symmetric if interchange left and right

Splay Tree

- Idea behind splay tree.
  - Every time find, get, add: or remove an element x, move it to the root by a series of rotations.
  - Other elements rotate out of way while maintaining order.
- Splay means to spread outwards
- The height of a splay tree is on average O(log n)

Self Balancing Trees

- AVL Trees (easy to code)
- Red-Black trees (best in practice)
- Treaps (easy to code, hard to prove height)
- Splay (mostly of theoretical interest)
- many others…