CS472 Module 4 Part B - The Searching Problem and Balanced Trees

Athens State University

March 7, 2016

Outline

Contents

1 The Searching Problem

The Searching Problem

Problem: Given a (multi)set S of keys and a search key K, find an occurrence of K in S, if any exists

- Searching must be considered in context of
  - File size (internal or external)
  - Dynamics of data
- Dictionary operations
  - Find
  - Insert
  - Delete

The Searching Problem: A Taxonomy of Searching Algorithms

- List searching
  - Sequential search
  - Binary search
  - Interpolation search
- Tree searching
  - Binary search tree

  - Binary balanced trees: AVL trees, red-black trees
  - Multiway balanced trees: 2-3 trees, 2-3-4 trees, B-trees

- Hashing
  - Open hashing
  - Closed hashing

2 Binary Search Trees

Binary Search Trees: A review

- Arrange keys in a binary tree with binary search tree property
- *Binary search tree*: A binary tree s.t. for a given node K, all nodes "less than" K are in the node's left most descendants and all nodes "greater than" K are in the node's right-most descendants.

Binary Search Trees: A review

- Operations on binary search trees
  - Searching: you know how to do it
  - Insertion: search for key, insert at leaf where search terminates
  - Deletion
    * Delete key at leaf
    * Delete key at node with single child
    * Delete key at node with two children
- Efficiency depends upon height of tree

$$log_2(n) \leq h \leq n - 1$$

  with height average to about $3 * log_2(n)$
  - Worst case efficiency: $\Theta(n)$
  - Average case efficiency: $\Theta(log(n))$

Binary Search Tree: It's not all a bed of roses

- Recall: the height h of a binary tree with n nodes must satisfy the relationship

$$n \leq 2^{h+1} - 1$$

- From this we know that the minimum height of a tree of n nodes is equal to $log_2(n)$ rounded down
- But, it is far too common that end up with a tree with height n
  - Consider for a tree with 1,000,000 nodes, the minimum height is $log_2(1,000,000)$ rounded down, which is 19

3 Balanced Trees

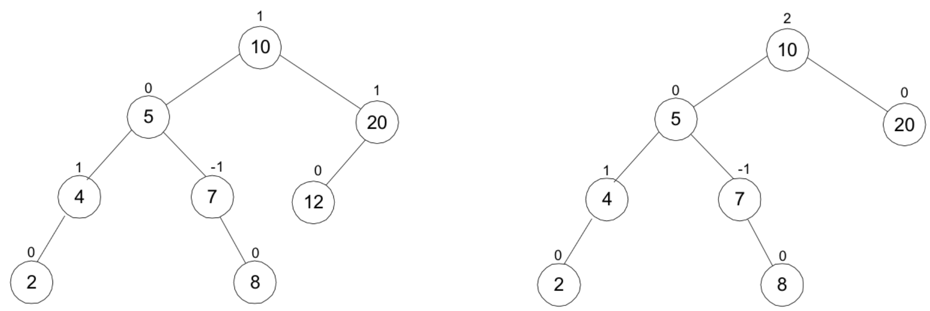
Balanced Trees

- We need a data structure that keeps the good qualities of the binary search tree but doesn't suffer as bad from a worst case
- Self-balancing trees do this by automatically transforming itself to keep the height of the tree as small as possible as items are inserted and deleted
- These trees do this by applying transformations at key times

Balanced Trees: AVL Trees

- *AVL Tree*: A binary search tree that, for each node, maintains the tree so that the difference in height of the node's subtrees is either -1, 0, and 1.
  - The height of an empty tree is defined as -1

Balanced Trees: AVL Trees: Example



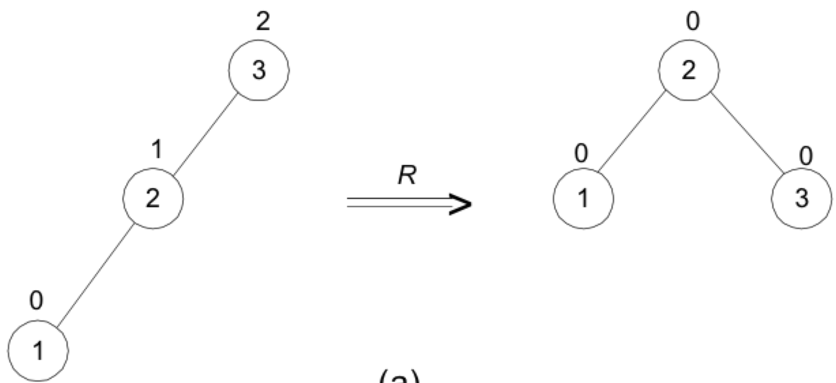
(a) (b)

Balanced Trees: AVL Trees: Rotations

If a key insertion violates the balance requirement at some node, then subtree that is rooted at that node is transformed back into an AVL tree by applying one or more *rotations*

- The rotation will be done to the subtree rooted at an "unbalanced" node closet to the new leaf

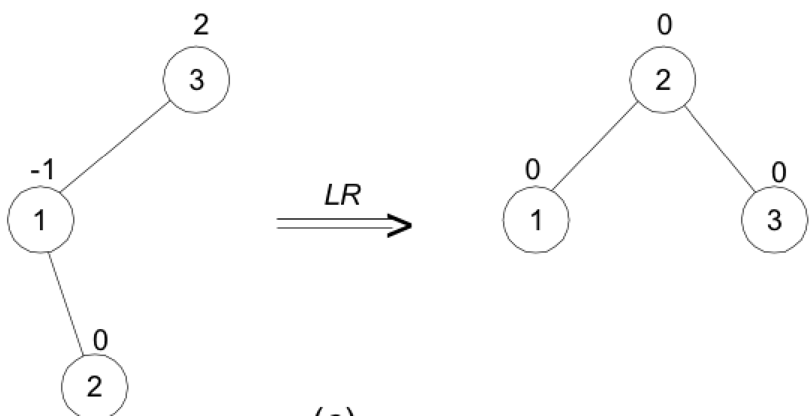
Balanced Trees: AVL Trees: Rotations



(a)

Single *R*-rotation

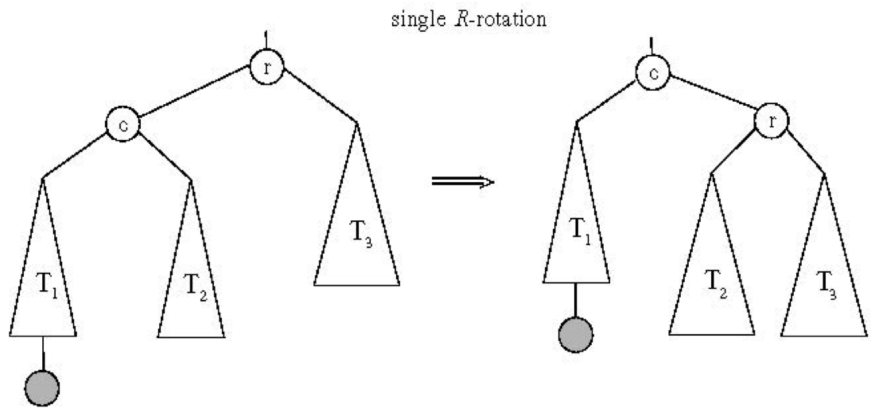
Balanced Trees: AVL Trees: Rotations



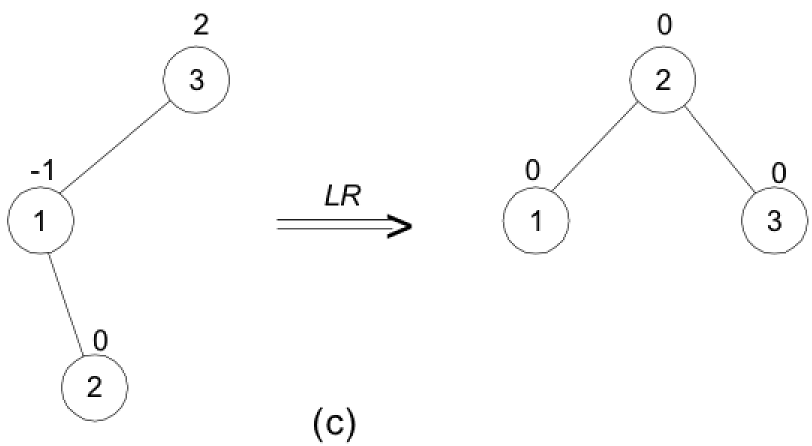
(c)

Double *LR*-rotation

Balanced Trees: AVL Trees: Rotations: General Case - Single R

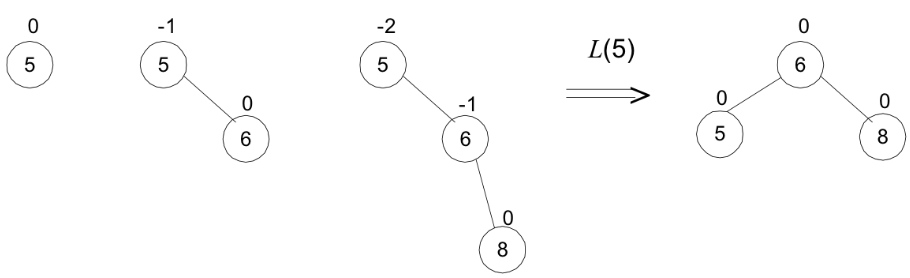


Balanced Trees: AVL Trees: Rotations: General Case - Double LR

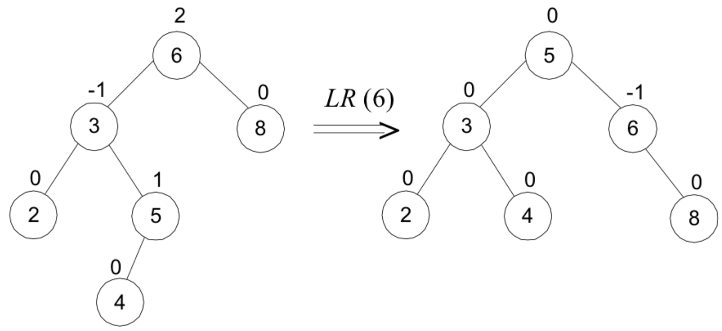


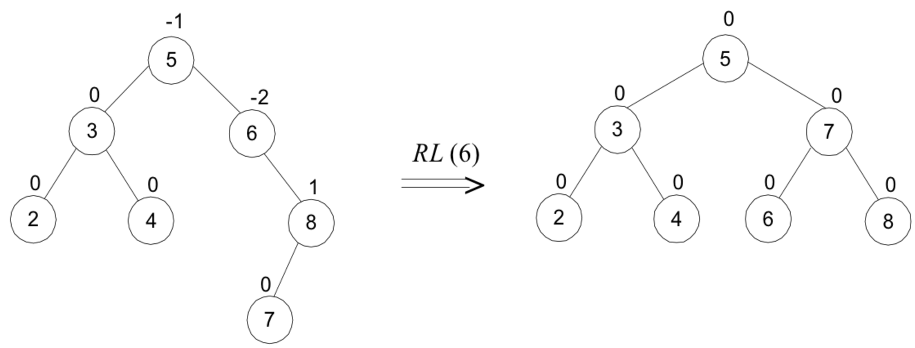
Double *LR*-rotation

Balanced Trees: AVL Trees: Example - 5,6,8,3,2,4,7



Balanced Trees: AVL Trees: Example - 5,6,8,3,2,4,7





p

Balanced Trees: AVL Trees: Analysis

- Height

$$h \leq 1.404 * log_2(n + 2) - 1.3277$$

- Search and insertion are $O(logn)$
- Deletion, while more complicated, is also $O(logn)$
- Disadvantages
  - Frequent rotations
  - Complexity
- A similar idea: *red-black trees* - height of subtrees is allowed to differ by up to a factor of 2

4 Key Points

Key Points

- Nature of the searching problem
- Binary Search Trees as solution to the searching problem
  - What is good and bad about BSTs?
- Self-balancing trees
  - AVL Trees