Algorithms & Complexity 2/13/2017 – 2/17/2017

0145-344-001

Note Taker: Jai Punjwani

ANNOUNCEMENTS

Topic: Searching & Trees

PowerPoint: <http://home.adelphi.edu/~siegfried/cs344/344l5.pdf>

Linear Search/Sequential Search: Traverse through array linearly until we find element (***O***(n)).

When our data set is SORTED, our search efficiency can improve by using a **binary search.** This is a divide and conquer algorithm in which we check whether our element is less than, greater than, or equal to the middle element of our array. If it is lesser than the middle element, we consider only the left half of our array, and if it is greater than the middle element, we consider only the right half. This is repeated recursively until we narrow our scope down to one element.

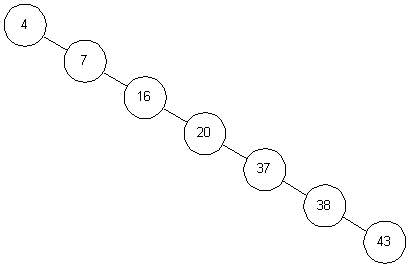
**Data Structures Used for Searching**

1. Lists/Arrays (sequential search, binary search, interpolation search)
2. Trees (Binary Search Tree, Binary Balanced Trees: AVL trees, red-black trees; Multiway balanced trees (B-trees, 2-3 trees)
3. Hash Tables

**Trees**

1. Binary Search Tree (BST): binary tree in which elements less than a node go to the left, and elements greater than a node go to the right.  
   Average Complexity for search: ***O***(log n)

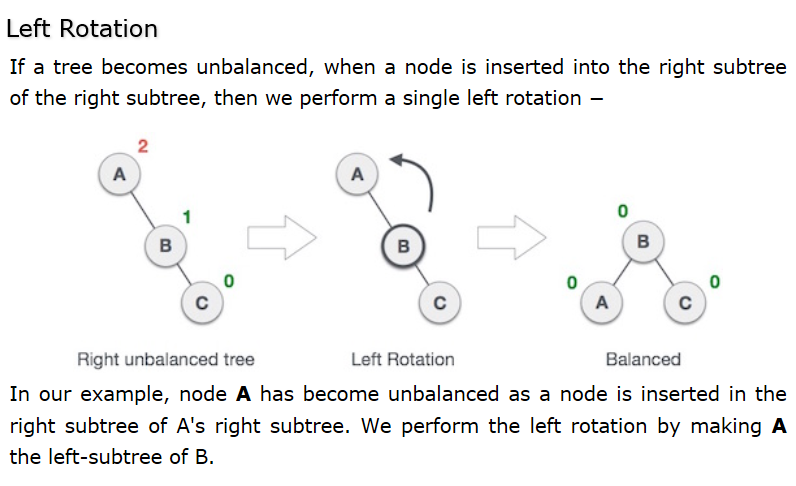
**BUT WHAT IF OUR TREE IS NOT BALANCED?**

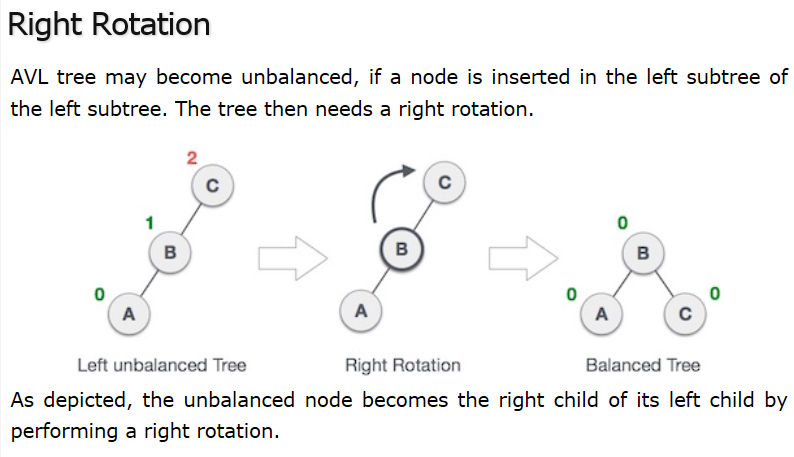


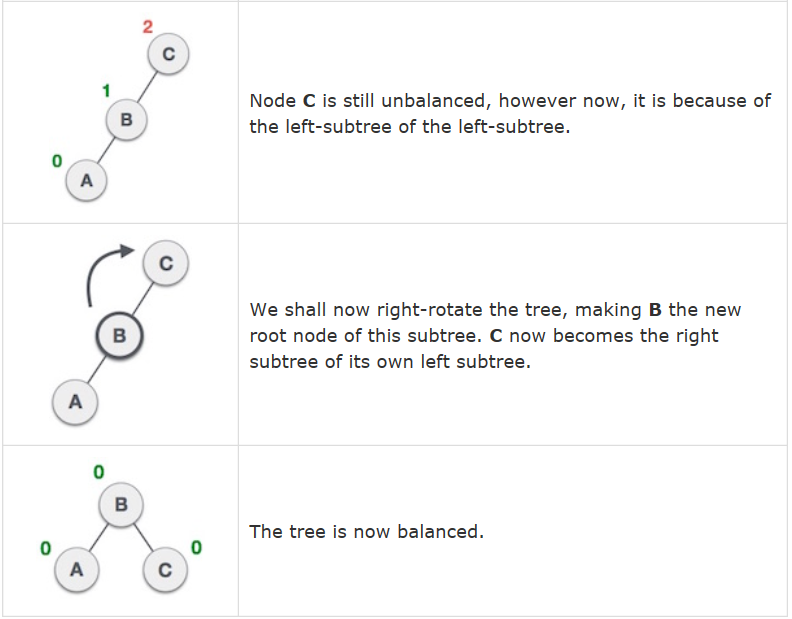
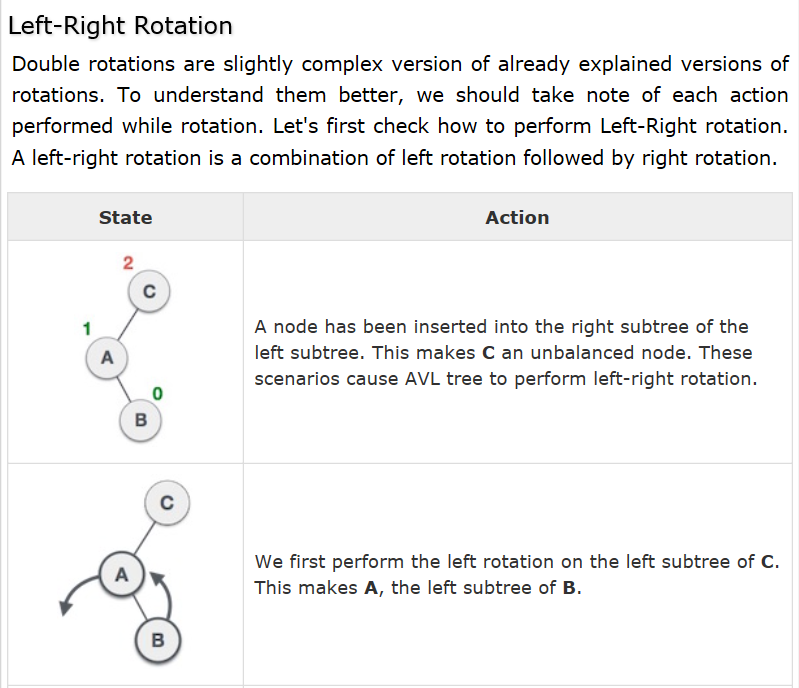
**Then searching becomes more of a sequential search. Enter balanced trees:**

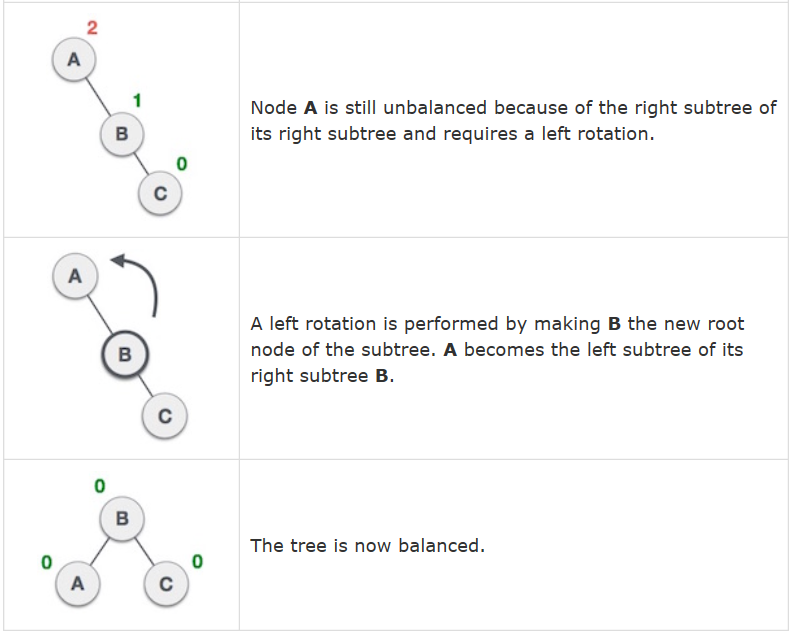
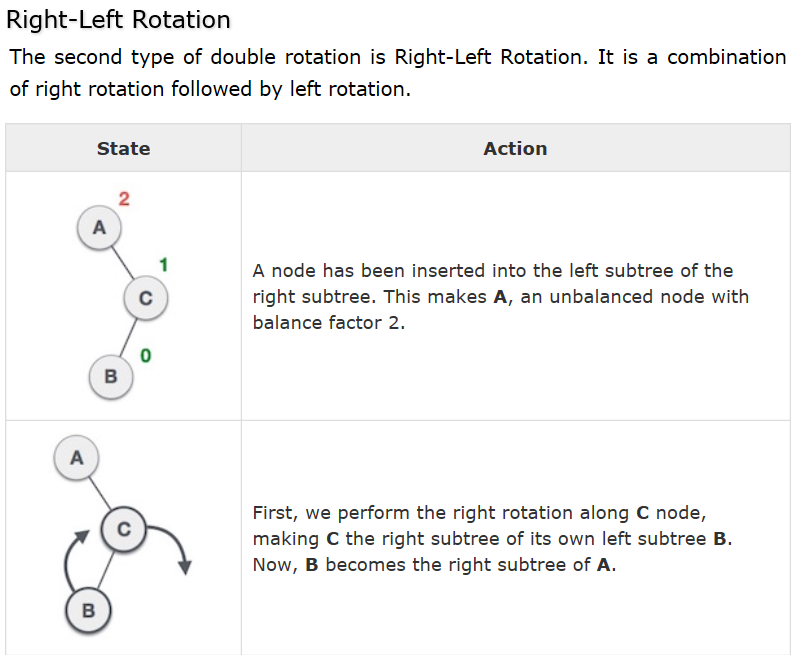
1. Binary AVL Tree – binary tree that balances itself on each insertion (if necessary) to ensure that the depths of any node’s left and right sub-trees do not differ by more than 1. In order to do this, we perform rotations. There are 4 general cases of rotations (below).

Complexity: ***O***(log n) for insertion and search









(These diagrams were taken from a tutorial found at: <https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm>)

1. Multiway Balanced B-Trees – B-trees are ordered just like BSTs. B-trees of an order n have nodes that can hold up to n-1 elements, and n children. Once a node is filled with n elements, the node is split into 3 nodes – one containing the middle element, one containing the left side, and one containing the right side. The middle element is promoted to the parent node.   
     
   **Note that a 2-3 tree is a B-tree of order 3.**

Below are some **useful resources** on B-trees and AVL Trees: <http://courses.washington.edu/css502/zander/Notes/06AVL-Btree.pdf>

Tools to help you create B-trees and AVL trees:

<https://www.cs.usfca.edu/~galles/visualization/BTree.html>

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

**Hashing** - a method of storing and retrieving data easily by transforming a key into an index (or an address) to retrieve its value. A **hash function** takes in data and computes its index. Hash functions are one-way (meaning that if we have an input we can generate its output BUT NOT vice versa). Good hash functions will yield a time complexity of ***O***(1) for insertion and retrieval.   
  
One common issue in hash functions is **collision,** that is when more than one input gets mapped to the same hash value. In the case that this occurs, a method called **linear probing** is used in which the key is simply placed at the next available slot in the hash table or array. Searching for a key collides with another hash value would still be easily done, provided that our hash function is a good one!