International Journal of Current Trends in Engineering & Research
e-ISSN 2455–1392 Volume 1 Issue 2, December 2015 pp. 8-11
http://www.ijcter.com

Study about AVL Tree & Operations

Viraj Mehta[1], Kapil Prajapati[2]

[1,2]Indira Gandhi National Open University

Abstract—AVL tree is the first dynamic tree in data structure which minimizes its height during insertion and deletion operations. This is because searching time is directly proportional to the height of binary search tree (BST). When insertion operation is performed it may result into increasing the height of the tree and when deletion is performed it may result into decreasing the height. To make the BST a height balance tree (AVL tree) creators of the AVL tree proposed various rotations. This paper shows BST and its operation and AVL.

Keyword— AVL, BST

I. INTRODUCTION

A binary search tree is a binary tree with a special property called the BST-property, which is given as follows:

For all nodes x and y, if y belongs to the left subtree of x, then the key at y is less than the key at x, and if y belongs to the right subtree of x, then the key at y is greater than the key at x.

We will assume that the keys of a BST are pairwise distinct. Each node has the following attributes:

- p, left, and right, which are pointers to the parent, the left child, and the right child, respectively, and
- key, which is key stored at the node.

Example:

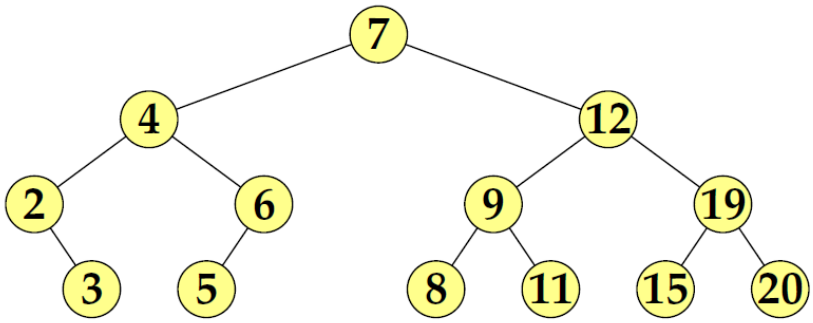

Figure 1: BST

Traversal of the Nodes in a BST By \traversal" we mean visiting all the nodes in a graph. Traversal strategies can be specied by the ordering of the three objects to visit: the current node, the left subtree, and the right subtree. We assume the the left subtree always comes before the right subtree. Then there are three strategies.

1. Inorder. The ordering is: the left subtree,the current node, the right subtree.

2. Preorder. The ordering is: the current node, the left subtree, the right subtree.

3. Postorder. The ordering is: the left subtree, the right subtree, the current node.

The AVL tree is named after its two Soviet inventors, Georgy Adelson-Velsky and E.M. Landis, who published it in their 1962 paper "An algorithm for the organization of information".[1] It take O(log n) time for the basic operations. We know that every AVL tree is a BST (Binary Search Tree) while every BST is not an AVL tree.[2] AVL tree is a selfbalancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

II. OPERATIONS ON AVL TREE

2.1 Searching Searching in an AVL tree is done as in any binary search tree. The special thing about AVL trees is that the number of comparisons required, i.e. The AVL tree's height, is guaranteed never to exceed log (n).

2.2 Traversal Once a node has been found in a balanced tree, the next or previous nodes can be explored in amortized constant time. Some instances of exploring these "nearby" nodes require traversing up to log(n) links (particularly when moving from the rightmost leaf of the root's left subtree to the root or from the root to the leftmost leaf of the root's right subtree; in the example AVL tree, moving from node 14 to the next but one node 19 takes 4 steps). However, exploring all n nodes of the tree in this manner would use each link exactly twice: one traversal to enter the subtree rooted at that node, another to leave that node's subtree after having explored it. And since there are n−1 links in any tree, the amortized cost is found to be 2×(n−1)/n, or approximately 2. After inserting a node, it is necessary to check each of the node's ancestors for consistency with the rules of AVL ("retracing"). The balance factor is calculated as follows: balanceFactor = height(left subtree) -height(right subtree). Since with a single insertion the height of an AVL subtree cannot increase by more than one, the temporary balance factor of a node will be in the range from −2 to +2. For each node checked, if the balance factor remains in the range from −1 to +1 then only corrections of the balance factor, but no rotations are necessary. However, if balance factor becomes less than −1 or greater than +1, the subtree rooted at this node is unbalanced.[1]

Rotations in Insertion Operation

In case of insertion, we have following rotations.

2.2.1 LL Rotation When a node X is inserted in the left sub tree of left sub tree of node N.
2.2.2 RR Rotation When a node X is inserted in the right sub tree of right sub tree of node N. [2] Pictorial description of how rotations rebalance an AVL tree. The numbered circles represent the nodes being rebalanced. The lettered triangles represent subtrees which are themselves balanced AVL trees. A blue number next to a node denotes possible balance factors (those in parentheses occurring only in case of deletion). **Figure 2**

2.3 Deletion Let node X be the node with the value we need to delete, and let node Y be a node in the tree we need to find to take node X's place, and let node Z be the actual node we take out of the tree. Deleting a node with two children from a binary search tree using the in-order predecessor (rightmost node in the left subtree, labelled 6).

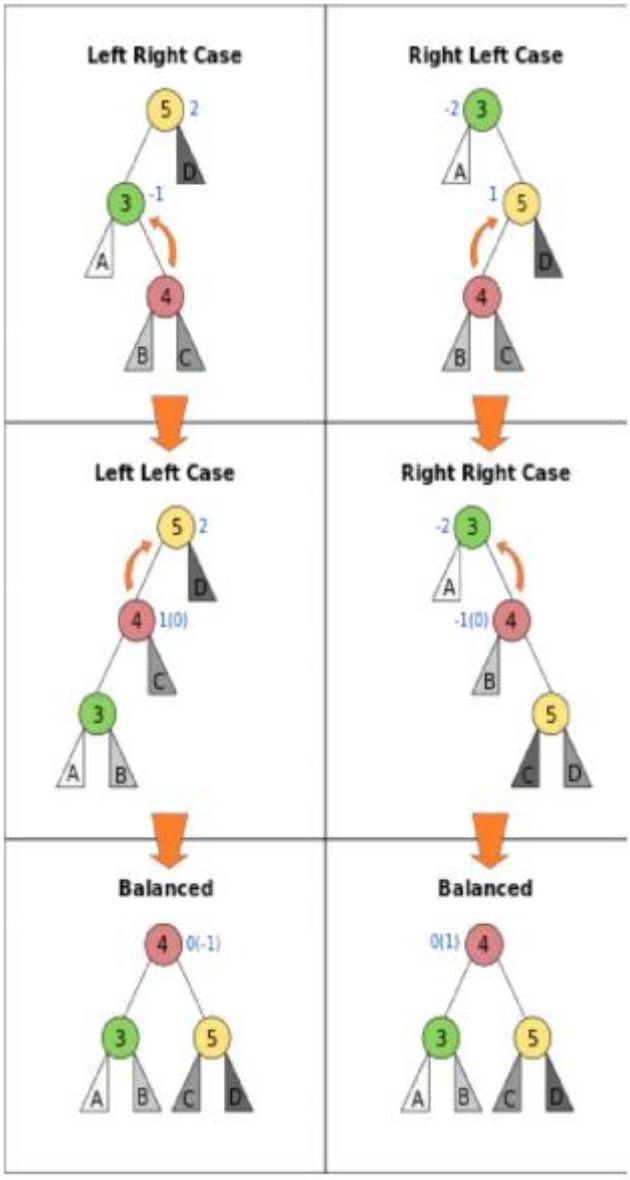

Figure 2 AVL Tree

Steps to consider when deleting a node in an AVL tree are the following:

1. If node X is a leaf or has only one child, skip to step 5 with Z: =X.
2. Otherwise, determine node Y by finding the largest node in node X's left subtree (the inorder predecessor of X − it does not have a right child) or the smallest in its right subtree (the in-order successor of X − it does not have a left child).
3. Exchange all the child and parent links of node X with those of node Y. In this step, the in-order sequence between nodes X and Y is temporarily disturbed, but the tree structure doesn't change.
4. Choose node Z to be all the child and parentlinks of old node Y = those of new node X.
5. If node Z has a subtree (which then is a leaf) attach it to Z's parent.
6. If node Z was the root (its parent is null), update root.
7. Delete node Z.

8. Retrace the path back up the tree (starting with node Z's parent) to the root, adjusting the balance factors as needed. [1]

Rotations in Deletion Operation In case of deletion, we have following rotations. 2.3.1 Let the deletion is being performed into right sub tree then we have three types of rotations R(0) rotation ,R(-1) rotation and R(+1) rotation. 2.3.2 Let the deletion is being performed into left sub tree then we have three types of rotations L(0) rotation ,L(-1) rotation and L(+1) rotation.[2] Here the authors of both the reviewed papers have demonstrated the searching, insertion and deletion operations on an AVL tree.

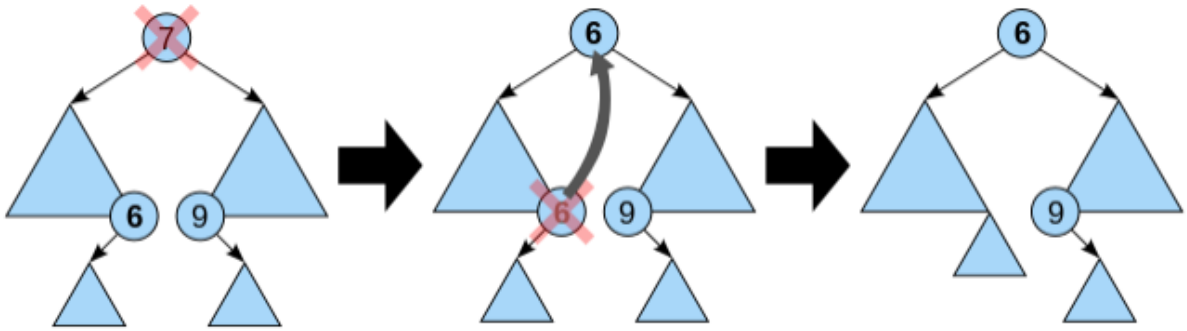

Figure 3 Deletion operation

III. CONCLUSION

Creators of the AVL tree proposed the various rotation schemes for balancing the BST for both insertion and deletion. To implement these rotations a care has to be taken that whether the new node will be in the left of the left subtree, left of the right subtree, right of the left subtree or right of the right subtree. Similar approach has to be applied in case of deletion also. In this paper we reviewed all these operations on an AVL tree.

REFERENCES

[1] Rajeev R. Kumar Tripathi. *Balancing of AVL Tree Using Virtual Node*. International Journal of Computer Applications Volume 7– No.14, October 2010.
[2] Siddharth Nair, Simran Singh Oberoi, Shubham Sharma. *Avl tree and its operations.* 2014 IJIRT | Volume 1 Issue 6
[3] https://www.cs.rochester.edu/~gildea/csc282/slides/C12-bst.pdf

