ISSN: 2321-7782 (Online)
Volume 2, Issue 12, December 2014
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

A New Way of Inserting and Deleting the Node To and From the AVL SEARCH Tree

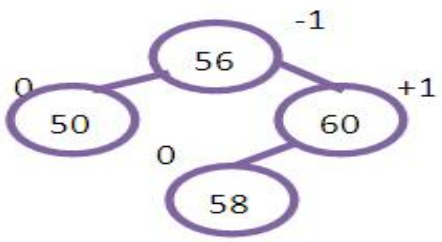
Goutam Mondal
Computer Science,
Belsingha Sikshayatan
West Bengal , India

Abstract: ***In the paper, it is tried to establish a new way to insert in and to delete the elements form the AVL Search tree. For this purpose, the concept of heap sort as well as binary search tree is used partially. The article shows the way, how easily maintain the balance factor after inserting in and deleting the elements from the AVL Search tree. The article also uses so few operations compare to the traditional insertion and deletion of AVL Search tree.***

I. INTRODUCTION

An AVL Search tree is a binary search tree which is an AVL tree i.e. each node of which does not contain other than the balance factor 0, -1 and +1. . The figure given below shows the representation of an AVL Search tree. The number against each node represents its balance factor.

Searching an AVL Search tree for an element is exactly similar to the method used in a binary search tree.



II. INSERTING IN AN AVL SEARCH TREE

Inserting an element into an AVL search tree in its first phase is similar to that of the one used in a binary search tree. However, if after insertion of the element, the balance factor of any node in the tree is affected so as to render the binary search tree unbalanced, we can take the help of heap sort partially.

To perform, it is necessary to identify a specific node A whose balance factor is neither 0, 1, -1 and which is the nearest ancestor to the inserted node on the path from the inserted node to the root. This implies that all nodes on the path from the inserted node to A will have their balance factors to be either 0, 1,-1. To restore balance two types of operations are performed.

LEFT INSERTION:

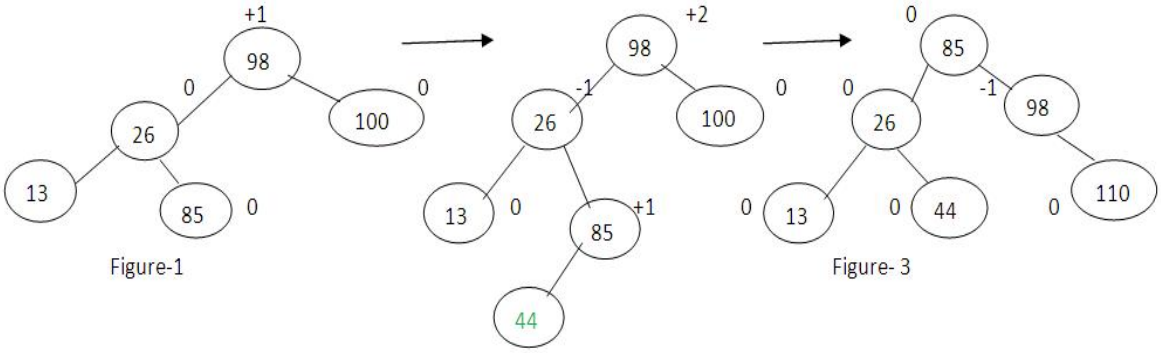
If the inserted node is in the left side of A, then we have to choose the greatest value node from the inserted node to the previous node of A and have to place it in place of A, placing A as its right child and left children of A except the greatest valued node as its left children in binary search tree manner.

RIGHT INERTION:

If the inserted node is in the right side of A, then we have to choose the least valued node from the inserted node to the previous node of A and have to place it in place of A, placing A as its left child and right children of A as its right children in binary search tree manner.

After promoting the above manner, if it is seen that one or more node have also unbalanced factor, then it has to take the nearest ancestor from the leaf node and have to perform the same operations either 1 or 2 as necessary.

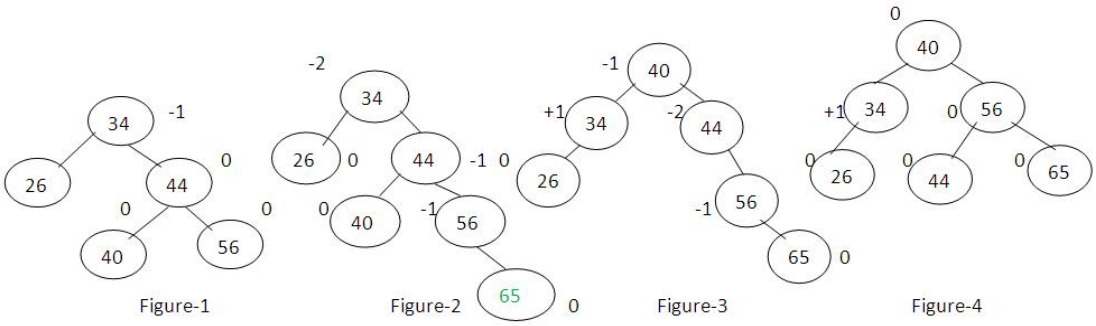
Example-1



In the figure- 1 the tree is balanced but when the node 44 is inserted , it becomes unbalanced that is shown in figure-2.

Since 44 has been inserted in the left side of the unbalanced node 98, so the greatest valued node from the left sub tree of 98 i.e. 85 is selected and placed in place of 98 keeping 98 as its right child and the left sub tree of 98 (except 44) as its left sub tree that is shown in figure-3.

Example-2



The tree shown in figure-1 is balanced but when the node 65 is inserted, it becomes unbalanced where 34 is the unbalanced node shown in figure-2.

Since 65 is in the right side of the unbalanced node 34, so the least valued node 40 is selected from the right sub tree of 34 and placed in place of 34 keeping 34 as its left child and the right sub tree of 34 as its (40) right sub tree shown in figure-3. But the tree in figure-3 is also unbalanced where 44 is the unbalanced node. So , from the leaf node 65 to previous of 56, since the least valued node is 56, hence 56 is placed in place of 44 keeping 44 as its(56) left child and 65 as its right child.

III. DELETION IN AN AVL SEARCH TREE

The deletion of an element in an AVL search tree proceeds as same procedures for deletion of an element in a binary search tree. However, in the event of an imbalance due to deletion, one or more operations need to be applied to balance the AVL tree .

On deletion of a node D from the AVL tree, let A be the closest ancestor node on the path from D to the root node with a balance factor +2 or -2. To restore balance the two types of operations are performed.

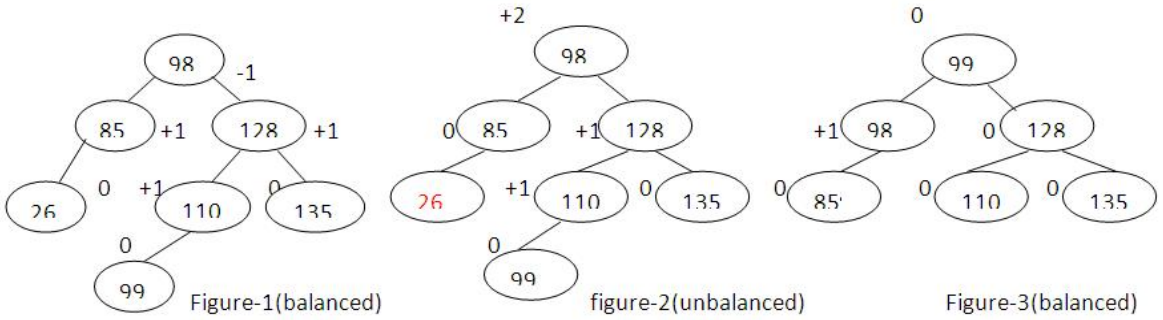
1. *Left Deletion*

If the node that is to be deleted, is in the left sub tree of the closest ancestor node a containing imbalance factor, then it has to be chosen the least valued node from the right sub tree of A and is to be placed in place of A keeping A as its left child right sub tree of A as its right sub tree in binary search tree manner. After this, if it is also seen that one or more nodes have imbalance factor, than identify the nearest imbalance ancestor A from leaf and select the smallest or greatest valued node from leaf to previous of A if it is in right or left sub tree of the imbalance factor respectively and is placed in place of A follow the same procedure of binary search tree and so on.

2. *Right Deletion*

If the deleted node is in the right sub tree of the closest ancestor node A containing imbalance factor, then it has to be chosen the greatest valued node from the left sub tree of A and is to be placed in place of A keeping A as right child and taking left sub tree except the node with greatest value as left sub tree in binary search tree manner. After this, if it is also seen that one or more nodes have imbalance factor, then identify the nearest imbalance ancestor A from leaf and select smallest or greatest valued node from leaf to previous of A if it is in right or left sub tree of the imbalance factor respectively and is placed in place of A and follow the same procedure of binary search tree and so on.

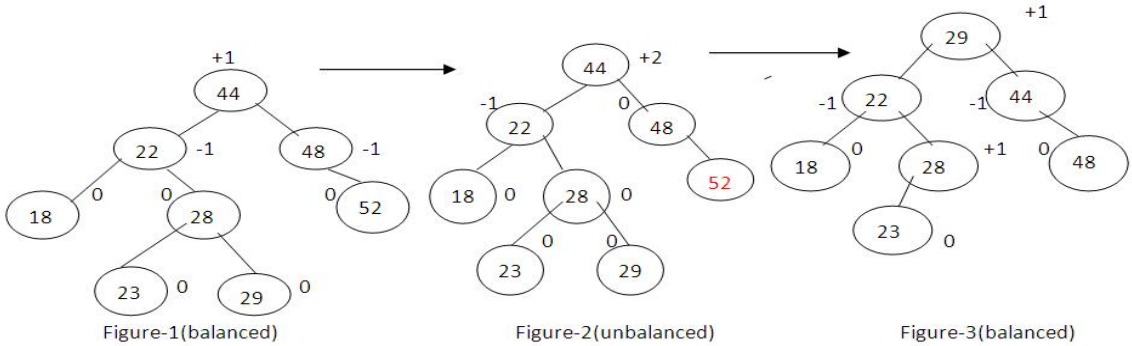
Example-1



In the figure-1 the tree is balanced but when the 26 is deleted from it, then it becomes unbalanced shown in figure-2(red colored node is the deleted node) with unbalanced node 98 containing factor +2.

Since the deleted node 26 was in the left sub tree of unbalanced factor node 98, so the least valued node 99 is selected from the right sub tree of 98 and is placed in place of 98 keeping 98 as its left child and right sub tree of 98 (except 99) as its right sub tree. Now the new tree is balanced which is shown in figure-3.

Example-2



In the figure-1 the tree is balanced but when the node 52 is deleted from it, then it becomes unbalanced shown in figure-2(red colored node is the deleted node) with unbalanced node 44 containing factor +2.

Since the deleted node was in the right sub tree of unbalanced node 44, so the greatest valued node 29 is selected from the left sub tree of 44 and placed in place of 44 keeping 44 as its right child and left sub tree of 44 (except 29) as its left sub tree. Now the new tree is balanced which is shown in figure-3.

ACKNOWLEDGEMENT

Author wishes to acknowledge Mrs Nilima Mondal for her continuous inspiration for developing and maintaining the Pair Sorting Method article.

References

1. S. Lipschutz and G.A.V. Pai , “Data Structures”, T. M. H. Education Pvt. Ltd.

AUTHOR(S) PROFILE

GOUTAM MONDAL, received B.TECH in Computer Science from Govt. College of Engg. & Textile Technology, Serampore under West Bengal University of Technology in 2006 And M.Sc in Computer Science from Vinayaka Mission University in 2013. Now engaged as an Assistant Teacher in Belsingha Sikshayatan,W.B,India.

