International Journal of Computer Applications (0975 – 8887)
Volume 7– No.14, October 2010

Balancing of AVL Tree Using Virtual Node

Rajeev R. Kumar Tripathi
SSSIST
Sehore (M.P) India

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) [1-9].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 proposes the balancing of the AVL tree using the concept of virtual node. This virtual node is a hypothetical node which is inserted into the inorder traversal of the BST and by doing the inorder traversal (left, root, right) we make a BST. Ultimately this virtual node is deleted to get an AVL tree.

Keywords

AVL, BST, Insertion, Deletion, Virtual Node.

1. INTRODUCTION

We know that binary searching is more efficient than linear searching [1-9]. But binary searching was applicable in contiguous memory location only. Performance factor of binary search inspired the people and they began to implement the binary search into distributed memory allocation by stating a problem "Can we have a nonlinear data structure in which we can perform binary search?"Solution to this problem is a comparison tree or a binary search tree (BST). This data structure enables us to search an item with average number of comparison O($\log_2 n$) while linear search requires O(n) comparisons, where n is the number of items[1]. Searching time in BST is directly proportional to the height of the tree. When insertion operation is performed into a BST, it may result into increasing the height of the tree. As height increased during the insertion operation, more searching time required. In case of deletion of a node height of BST may decrease and hence in the same proportion time requirement reduces. To make the BST a height balanced tree Adelson, Velskii and Landis proposed the concept of AVL tree.
We know that every AVL tree is a BST (Binary Search Tree) while every BST is not an AVL tree[1]. To make the BST a height balanced tree creator of AVL proposed various rotations in case of insertion and deletion. In case of insertion, we have following rotations.

1.1 Rotations in Insertion Operation

In case of insertion, we have following rotations.

1.1.1 *LL Rotation*

When a node X is inserted in the left sub tree of left sub tree of node N.

1.1.2 *RR Rotation*

When a node X is inserted in the right sub tree of right sub tree of node N. LL rotation is shown by fig(1)&(2).

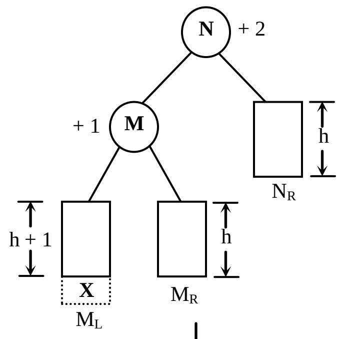

Fig 1

LL Rotation

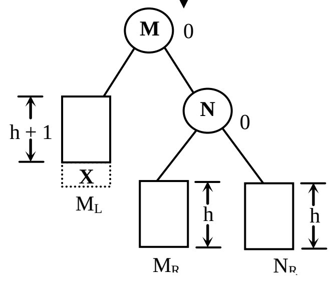

Fig 2

1.1.3 *LR Rotation*

When a node X is inserted in the right sub tree of left sub tree of node N.

1.1.4 *RL Rotation*

When a node X is inserted in the left sub tree of right sub tree of node N.
LR rotation is shown by fig(3)&(4).

LL rotation and RR rotation are mirror image of each other. Similarly LR rotation and RL rotation are mirror image of each other.

1.2 Rotations in Deletion Operation

In case of deletion, we have following rotations.

1.2.1 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.

1.2.2 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.

The L rotations are the mirror image of the R rotations. Using these rotations insertion and deletion from the BST requires $O(\log_2 n)$ times where n is the number of nodes in the BST[2].
We know that inorder traversal of BST always gives the result into ascending order.

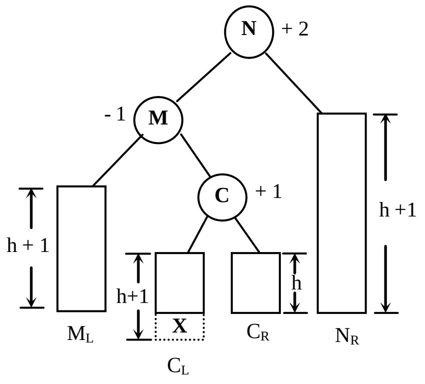

Fig 3 LR Rotation

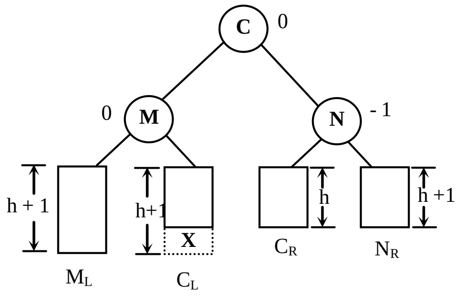

Fig 4

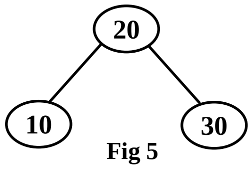

Fig 5

The fig (5) represents the BST. Its inorder traversal is 10, 20, 30.In inorder traversal we first traverse the left child, root and right child. For any given BST we have only two **cases (1)** Either the BST contains total number of nodes odd or **(2)** BST contains total number of nodes even.

2. PROPOSED VIRTUAL NODE CONCEPT IN AVL TREE

Virtual node is the hypothetical node whose value is greater than the values of nodes in left subtree but less than the values of nodes in right subtree.If we get only three nodes into the traversal, there is no need to insert a virtual node. The mid node will be the root and the predecessor of the mid will be the left child and successor of the mid will be the right child.

2.1 Case (1)

Let us consider the following BST in fig (6). Inorder traversal of fig (6) is 10, 20, 30, 40, 50.Now we will divide this set (output of inorder traversal) into three subsets left subtree nodes (LSN), root node(RN)and right subtree nodes(RSN) as:

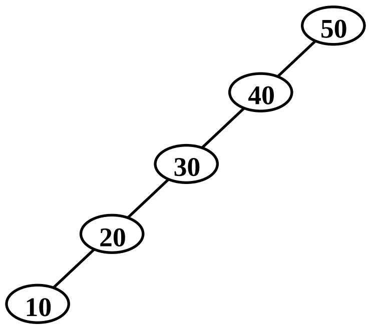

Fig 6

(1) The $(m/2)^{th}$ index node will be the root.
(2) Nodes from index 1 to (m/2)-1 will be in left subtree nodes.
(3) Nodes from index (m/2) +1to n will be in right subtree nodes.

10, 20	30	40, 50
LSN	**RN**	**RSN**

We will consider the left subtree nodes as a single node and it will be the left node of the root 30, and right subtree nodes as a single node and it will be the right node of the tree.
Now by using this assumption we will construct the BST by following the inorder traversal. And we will get the tree as in fig (7).

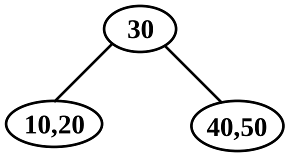

Fig 7

Let us consider the left child having the node10, 20. Now we will insert the virtual node **x** at $(m/2 +1)^{th}$ index .Where 10<**x**<20.Now we have the inorder sequence as 10,**x**,20.Now we will use this sequence as

10	**x**	20
LSN	**RN**	**RSN**

Now we will construct the BST using the inorder traversal and we will get the tree as in fig(8).

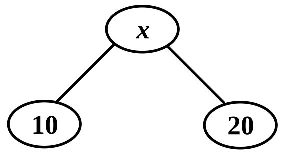

Fig 8

Let us consider the right child having the node 40, 50. Now we will insert the virtual node **y** at $(m/2 +1)^{th}$ index .Where 40<**y**<50.Now we have the inorder sequence as 40,**y**,50.Now we will use this sequence as

40	**y**	50
LSN	**RN**	**RSN**

Now we will construct the BST using the inorder traversal and we will get the tree as in fig (9).

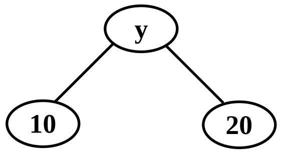

Fig 9

Now we will expand the tree shown in fig (7) by using the trees given in fig (8) & (9), we will have the tree as in fig (9)

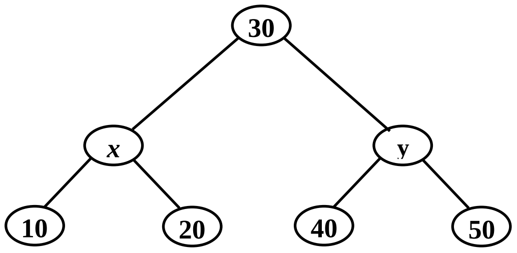

Fig 10

Now we have two virtual nodes **x**, **y**. After deletions of these virtual nodes we have an AVL tree like the fig(11).

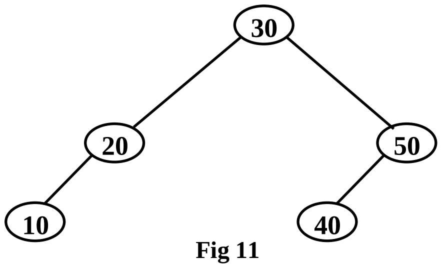

Fig 11

2.2 Case (2)

Let us consider the BST in fig(12).

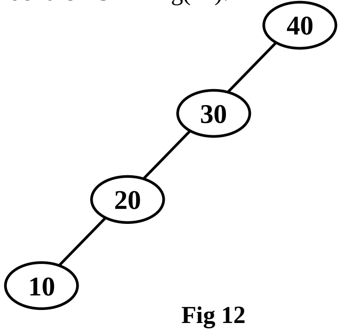

Fig 12

Inorder traversal of fig (12) is 10,20,30,40. Now we will divide this set (output of inorder traversal) into three subsets left subtree nodes, root and right subtree nodes as:

(1) Insert a virtual node **x** at ((m/2) +1)th index and it will be the root.

(2) Nodes from index 1 to ((m/2)-1)th will be in left subtree nodes.

(3) Nodes from index ((m/2) +1)th to m will be in right subtree nodes.

After performing the above operation we will get the sequence as: 10, 20, **x**, 30, 40.Where x is a virtual node as 20<**x**<30.

Now we will use this sequence as

10, 20	**x**	30, 40
LSN	**RN**	**RSN**

We will consider the left subtree nodes as a single node and it will be the left node of the root **x**,and right subtree nodes as a single node and it will be the right node of the tree. Now by using this assumption we will construct the BST by following the inorder traversal. And we will get the tree as in fig (13).

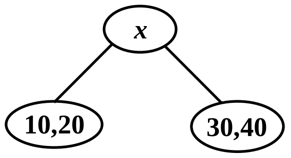

Fig 13

Now consider the left child nodes 10, 20, it is even in number. Now insert a virtual node **a** into it according to the formula **(**m/2+1**)**th we have the inorder sequence as 10,**a**,20 .Where **a** is a virtual node as 10<**a**<20.Now we will use this sequence as

10	**a**	20
LSN	**RN**	**RSN**

Now we will construct the BST using the inorder traversal and we will get the tree as in fig (14).

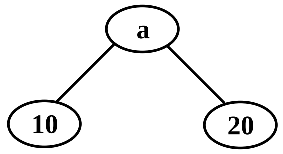

Fig 14

Now consider the right child nodes 30, 40, it is even in number. Now insert a virtual node **b** into **it** at (m/2+1)th index. We have the inorder sequence as 30, **b, 40** where **b** is a virtual node as 30<**b**<40.Now we will use this sequence as

30	**b**	40
LSN	**RN**	**RSN**

Now we will construct the BST using the inorder traversal and we will get the tree as fig (15).

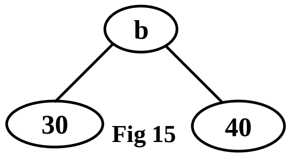

Fig 15

Now we will expand the tree shown in fig (13) by using the trees given in fig (14) & (15), we will have the tree as in fig (16).

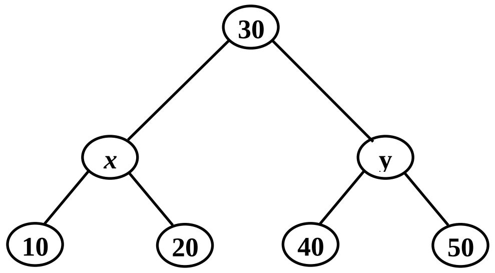

Fig 16

Now we have three virtual nodes **a**, **x** and **b**. Now deleting these virtual nodes from fig (16) we have an AVL tree like the fig(17).

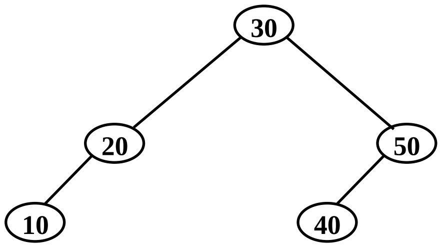

Fig 17

2.3 Deletion

Let we want to delete the node 40 from the AVL tree shown in fig (12). After deleting 40 we have the BST as shown in fig(18).

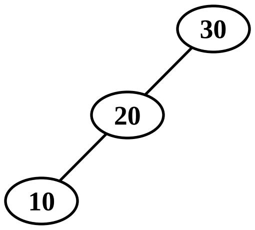

Fig 18

As this BST has 3 nodes, by using the concept of virtual node 20 will be the root and node 10 & 30 will be the left and right child.

3. ALGORITHM

1-Traverse the BST into inorder and count the total number of nodes m.
2-if m==3
3-The mid node of the inorder traversal will be the root node, its predecessor will be the left node and successor will be the right node.
4-if m is even then
5-Introduce virtual node into the list at (m/2 +1)th index.
6-Make this virtual node root. Keep the first m/2 nodes of original list into left sub tree and m/2+1 to m nodes into right sub tree.
7-Repeat step 5th and 6th until each node contains only one key.
8- Delete the virtual nodes.
9- Else (if m is odd)
10-Make the (m/2)th index node as root node.
11-put the first m/2 nodes into left sub tree and (m/2 +1)th to m index nodes into right sub tree. That is left sub tree and right sub tree has even number of nodes.
12-Repeat step 5,6,7.
13-Delete the virtual nodes.
14-Stop

This algorithm will be used into both the cases i.e. insertion as well as in deletion.

4. ANALYSIS

In this algorithm original list of inorder traversal is being partitioned as –

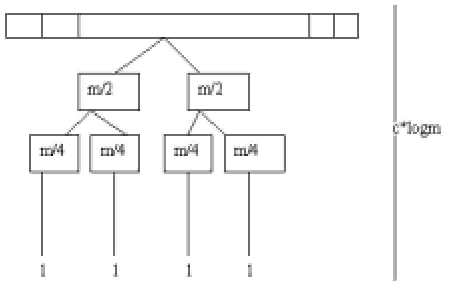

Fig 19

Where m is total number of nodes into the BST and c is the partitioning cost. Let m is even then total number of virtual nodes is equal to (m/2 +1) and if m is odd then total number of virtual nodes are m/2.Let the introducing cost of virtual node be a then total cost of virtual node insertion is a* (m/2).Let deletion cost be b then total cost of deletion of these nodes will be b*(m/2). Then total cost T will be:

$T=c*\log_2 m + (a+b)\ m/2$
$\equiv \Theta(\log_2 m)$

5. 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 a new concept of virtual node is introduced, due to which it becomes easy to balance a BST, in both the cases of insertion and deletion. By analyzing algorithm of both methods i.e. rotations and virtual node, it is found that the complexity is same but proposed one is easy to implement than the other one.

6. REFERENCES

[1] R.R.K.Tripathi, Concepts of Data Structure Using C, Katson Publication, 1st edition.

[2] Ellis Horowitz & Sartaj Sahni, Fundamentals of Data Structures, Galgotia Book Source, 1st edition.

[3] Trembley & Sorenson, An Introduction to Data Structures with Applications, TMH, 2nd edition.

[4] Robert L. Kruse, Data Structure and Program Design, PHI, 3rd edition.

[5] Yedidyah Langsam, Moshe J. Augenstein and Aaron M. Tenenbaum. Data Structures Using C and C++, PHI, 2nd edition.

[6] Seymour Lipschutz, Data Structures, McGraw-Hill,1st edition.

[7] Adam Drozdek, Data Structures & Algorithms in Java, Vikas Publication, 1st edition.

[8] Bhagat Singh and Thomas L. Naps, Introduction to Data Structures, Galgotia Book Source, 1st edition.

[9] Marry E.S. Loomis, Data Management & File Structures, PHI, 2nd edition.