A Literature Review on Various Balanced Binary Search Techniques

Ms.Reshma Jadhav[1], Ms.N. S. Soni[2], Mrs. D. V. Kirange[3]
[1,2,3]Computer Department, DYPCOE, Akurdi

***Abstract*—Binary Search Tree (BST) is a Basic Data Structure in Computer Science; because of its efficient time complexity for performing various operations such as insertion, deletion and searching a particular value in BST. But in the worst case of BST or unbalanced BST is not efficient for any of the operations on BST it is just like a linked list. In this paper we will compare various techniques to construct a balanced BST.**

***Keywords*— BST, AVL, RBT, LL, RR, LR, RL.**

I. INTRODUCTION

Binary Search Tree is an Advanced Data Structure used to manage data storing in such a way that Time complexity to perform various operations such as add record, delete record will be O(log n). To achieve this TC BST having some properties.

1. Every Node having at most 2 children's (2 pointers left and right)
2. If child node value less than root node it will be placed to left side of root node
3. If child node value Greater than root node it will be placed to right side of root node [2]

The Worst case occurs when data is in sorted form like in ascending or descending then it will create either right biased or left biased BST and in this case TC to perform various operations on BST will be O (n). In this case the BST needs to be balanced.[1]

In Present Literature Survey multiple Balancing methods for BST were proposed and these methods guarantee all Basic operations search, insert, delete with TC (log n). Each method has its own advantages and disadvantages as well different application areas

This paper introduces some balancing methods and their comparisons as well as their applications this will help software developers to choose most efficient BST method.

II. EXISTING BST BALANCING METHODS

A. AVL Tree

AVL tree named after inventors Adelson Vleskii and Landis.in 1962. In avl tree BF (balance Factor) is most important parameter. So the node structure for AVL tree is

Left pointer	BF	Data	Right pointer

Fig. 2.1 AVL tree node structure

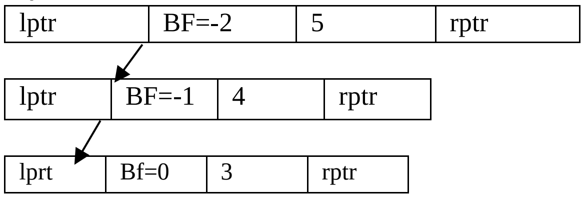
BF of each node can be calculated as

BF(X) =Height (right subtree(X)) – Height (left subtree(X))

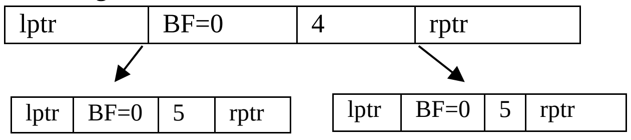
A binary tree is an AVL tree if B(X) ϵ {-1, 0, 1}

In worst case we need to rebalance the tree and achieve BF property of AVL tree and this rebalancing done through rotations. And these rotations divided into 2 parts

a) Single rotation: this rotation is used when tree is left skewed or right skewed, single rotations are of two types RR(right rotation) and LL (left rotation) for key values 5,4,3 tree will be as shown below [3]

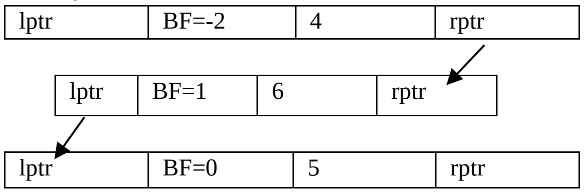


In above fig BF of node X is -2 not in range and it is left biased tree apply rotation and balance tree after balancing

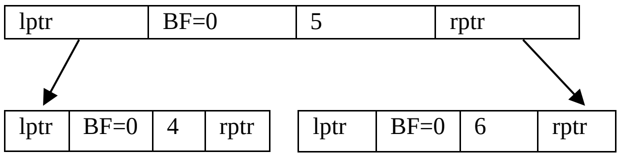


Above fig shows tree after balancing the BF of root node is 0 which is valid *b)* Double rotation: this rotation is used when there is unbalancing in left- right or right-left subtree these rotations are left right (LR) rotation and right left (RL) rotations

For keys 4, 6,5 The tree will be as shown below



BF for node x is -2 which is not in range and this thee is unbalanced tree due to right left insertion of data so apply (RL) rotation first we apply right rotation and then apply left rotation this is called as double rotation.



Above fig shows tree after (RL) balancing and the BF now changes.

Operations on AVL tree:

1. Insertion
2. Deletion
3. Searching

Insertion rules for AVL tree:

Time Complexity analysis of AVL tree:

1. Insertion: AVL also follows BST properties so insertion takes O (log n) time in worst case one node must be rebalanced and rebalancing by rotation takes O (1) time by rotation at each node. So overall TC of insertion is O (log n).
2. Deletion: AVL also follows BST properties so deletion takes O (log n) time in worst case one node must be rebalanced and rebalancing by rotation takes O (1) time by rotation at each node. So overall TC of deletion is O (log n).
3. Searching: AVL tree is balanced BST so for both average and worst case TC is O (log n)[4]

B. Red Black Tree(RB Tree)

The red-black tree was invented by Rudolf Bayer in 1972. RB tree is a balanced BST where each node having color either red or black. In BST worst case we apply some RB tree properties to make a BST Balanced BST with TC O (log n) the properties are as follows

1. It should be a binary search tree.
2. Every node having color either red or black.
3. Root node is always black.
4. The children of red colored node must be black.
5. The number of black colored nodes should be same for all paths.
6. Every new node must be inserted with red color.
7. Every leaf must be of black color.

Operations on red-black tree:

1. Insertion
2. Deletion
3. Searching

Insertion rules in RBT:

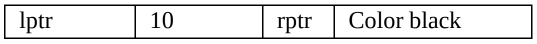
1. Check weather tree is empty or not if it is empty create a new node with color black
2. If tree is not empty create a leaf node with color red.
3. If parent of new node is black then exit
4. If parent of new node is not black or red then we have to check parents sibling of new node

a) If sibling is black then we perform rotation and re coloring.

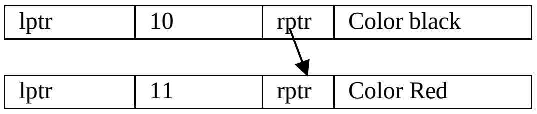
b) If sibling is red then we recolor the node and check for its grandparent if grandparent is root node or not if root node then exit else we will recolor and recheck the node.

Example: Insert following keys into RBT
10,11,12,15

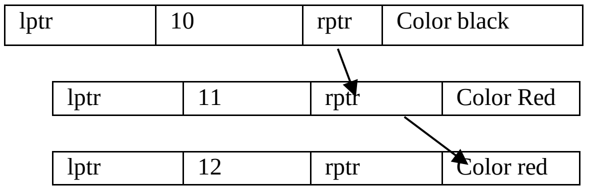
Step 1: insert 10 with color Black (insertion R1)



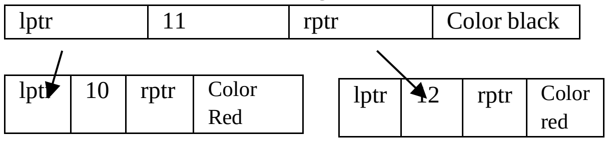
Step2: insert 11 with color red (insertion R2)



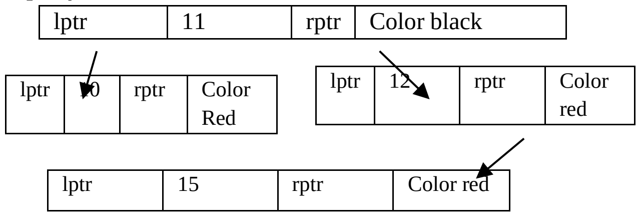
Step 3 insert 12 with color red (Insertion R2) but violets RBT Property no 4 so according to insertion R4a rule rotate and recolor



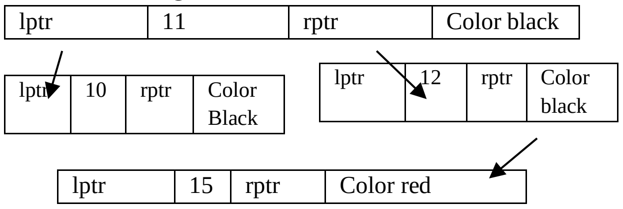
After Rotation and Recoloring



Step 4 insert 15 with color red (insertion R2) but violates RBT property 4 so according to insertion property 4b recolor the node



After Recoloring



Above is final RBT

Time Complexity analysis of RBT

1. Time complexity for inserting a node into a Red-Black Tree is O(log n).and Rotation takes O(1)
2. Deletion: RBT also follows BST properties so deletion takes O (log n) time in worst case one node must be rebalanced and rebalancing by rotation takes O (1) time by rotation at each node. So overall TC of deletion is O (log n).
3. Searching: RBT tree is balanced BST so for both average and worst case TC is O (log n)[1]

C. Splay Tree

Is also one of self balancing tree. Splay tree contain 6 types of rotations

- Zig rotation: Similar as right rotation
- Zag rotation: Similar as Left rotation
- Zig-Zig rotation: Similar as two right rotations
- Zag-Zag rotation: Similar as two left rotations
- Zig-Zag rotation: Similar as right left rotation
- Zag-Zig rotation: Similar as two left rotations

Time Complexity analysis of splay tree:

1. Time complexity for inserting a node into a splay Tree is O(log n).and Rotation takes O(1)
2. Deletion: splay tree deletion takes O (log n) time in worst case
3. Searching: Splay tree searching takes for both average and worst case TC is O (log n)[1]

III. APPLICATIONS OF VARIOUS BALANCED TREES

Sr. No	Tree Name	Application
1	AVL Tree	network routing, text editor, file system etc.
2	RB Tree	data structure, memory allocator, file system etc.
3	Splay Tree	geospatial data , online algorithms etc.

REFERENCE

[1] Fahd Mustapha Meguellati, , Djamel Eddine, “A Survey on Balanced Binary Search Trees methods,”)*ICSAT 2021*

[2] Dr.R.Chinnaiyan, Abhishek Kumar , “Construction of Estimated Level Based Balanced Binary Search Tree “, ICECA 2017

[3] Siddharth Nair, Simran Singh Oberoi, Shubham Sharma, “AVL TREE AND ITS OPERATIONS”, IJIRT 2014

[4] Lalit Kumar, Gourav Agghi, Nishant Malik, Ajay Anand,” A BRIEF STUDY OF BALANCING OF AVL TREE” IJIRT 2014