Page :
Date : / /

Assignment 8

Title: AVL Tree.

Problem: A Dictionary stores keywords f its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data stored in ascending/descending order. Also find how many maximum comparisons may require for finding any keyword. Use height balance ~~t~~ tree and find the complexity for finding a keyword.

Objective: To understand construction of AVL tree and its rotation techniques.

Outcome: At this end of this assignment, student will be able to construct an AVL tree and perform rotation.

Theory:
AVL Tree (Self Balancing Tree) is a binary search tree in which the difference of heights of left and right subtrees of any node is less than or equal to 1. The technique of balancing the height of binary tree was developed by Adelso, Velskii and Landis and hence given the short form as AVL tree.
An AVL tree can be defined as follows:
Let T be a~~n~~ non-empty binary tree with

Scanned with CamScanner

Page :
Date : / /

T_L and T_R as its left and right subtrees.
The tree is height balanced if:

i) T_L and T_R are height balanced.
ii) $h_L - h_R <= 1$, where h_L, h_R are the heights of T_L and T_R.

The Balance factor of a node in a binary tree can have value 1, -1, 0 depending on whether the height of its left subtree is greater than or less than or equal to the height of the right subtree.

Advantages of AVL Tree:
- Since AVL trees are height balance trees, operations like insertion and deletion have low time complexity.

Balance factor = height (left subtree) - height (right subtree)

AVL Rotations:
1) To balance itself, an AVL tree may perform the following four kinds of rotations:
i) Left Rotation   iii) Left-Right Rotation.
ii) Right Rotation   iv) Right-Left Rotation.

The first two rotations are single rotations and the next two rotations are double rotatio

Scanned with CamScanner

Page :
Date : / /

~~The first two rotations are su~~

Representation of AVL Trees :

```
Struct AVLNode
{
    int data;
    struct AVLNode *left, *right;
    int balfactor;
};
```

Algorithm :

i) Insertion :

a) First, insert a new element into the tree using BST's insertion logic.

b) After inserting the elements you have to check the balance factor of each node.

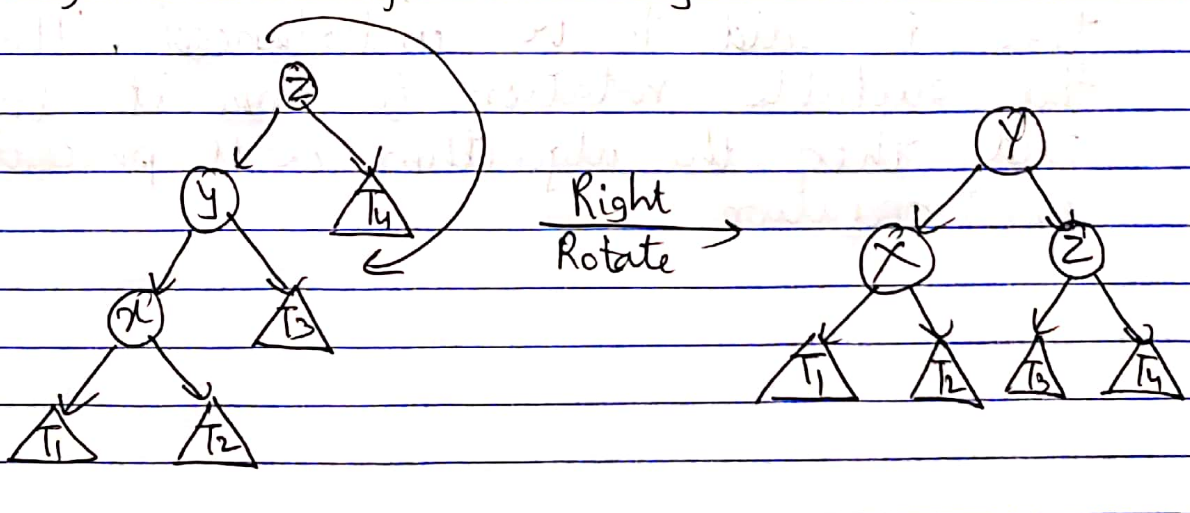
c) When balance factor of every node will be found like 0,-1, or 1 then algorithm will proceed for the next operation.

d) When the balance factor of any node comes other than the above three values than the tree is said to be imbalanced. Then perform the suitable rotation to make it balanced and then the algorithm will proceed for the next operation.

Page :
Date : / /

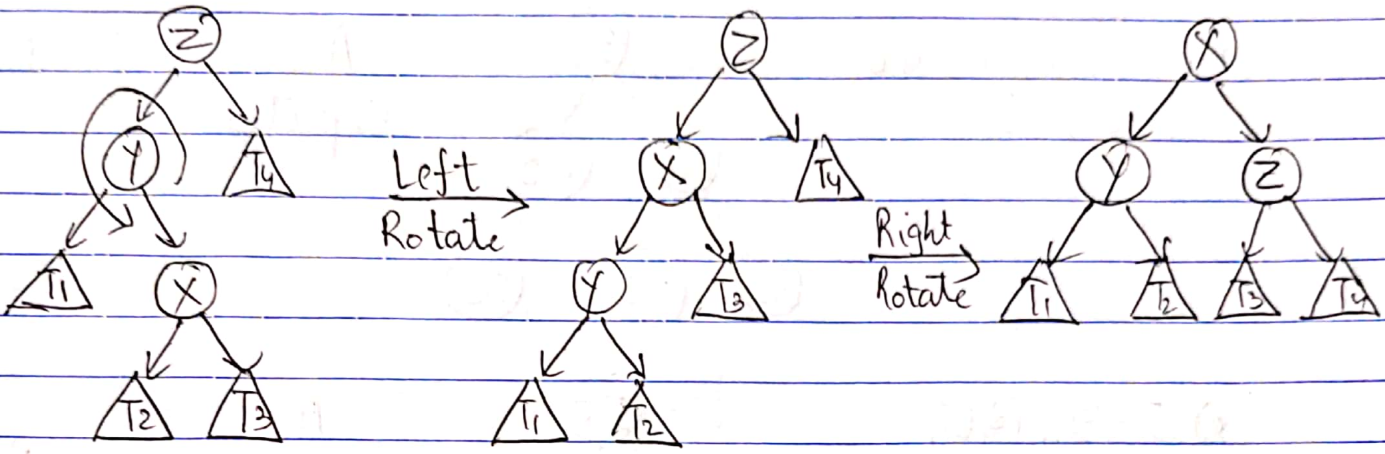
2) Delete
a) Firstly, find that node where k is stored.
b) Secondly delete those contents of the node
~~(Suppose the node is~~
c) Deleting a node in an AVL tree can be reduced by deleting a leaf. There are three possible cases:
→ When x has no children then delete x.
→ When x has one child, let 'x' be child of x
→ Notice x' cannot have a child, since subtrees of T can differ in height by at most one.
→ then replace the contents of x with 'x'.
→ then delete x' (a leaf)
d) When x has 2 children.
→ then find x's successor z (which has no left child)
→ then replace x's content with z's contents and delete z

Rotations :
1) Left - Left case: x is the left child of y & y is the left child of z

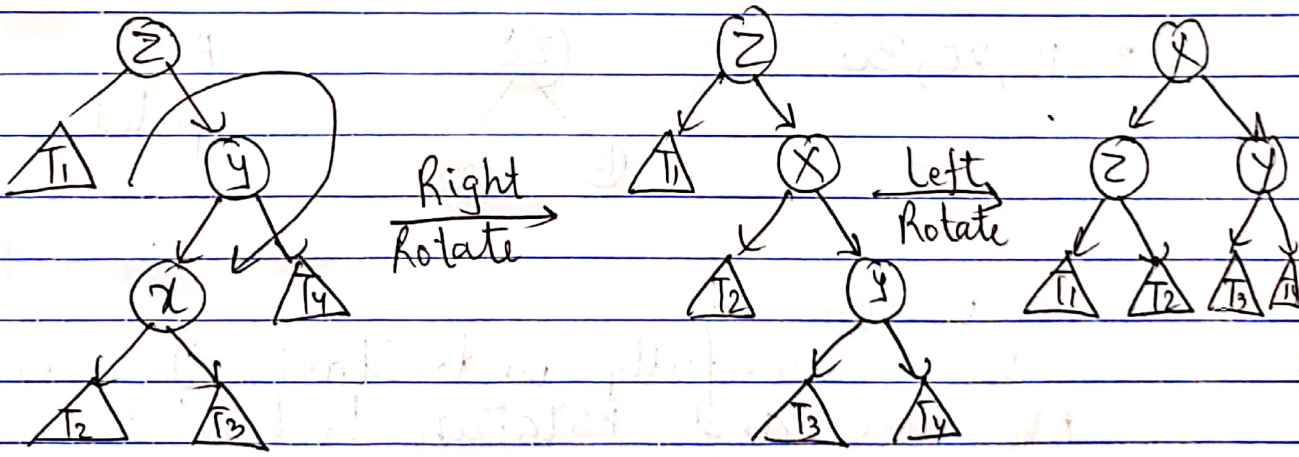


Scanned with CamScanner

2) Left-Right case : x is the right child of y & y is the left child of z.

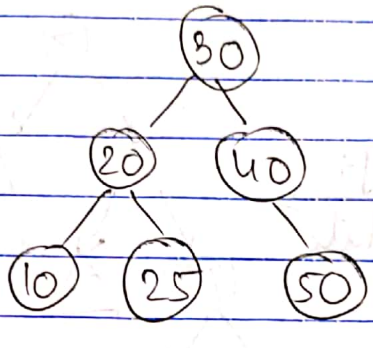
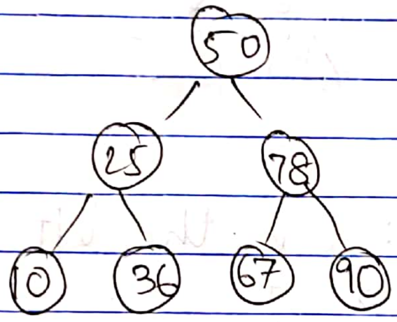
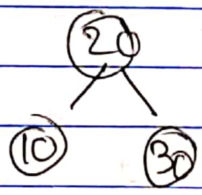


3) Right-left Case : x is the left child of y & y is the right child of z.



Page :
Date : / /

Test Case :

Test Case	Expected	Outcome	Result.
1) 10, 20, 30, 40, 50, 25		As Expected	Pass
2) 50, 36, 78, 67, 25, 10, 90		As expected	Pass
3) 10, 20, 30		As expected	Pass

Conclusion:
We successfully understood the concept of AVL tree and rotation techniques and completed the assignment.

Scanned with CamScanner