AVL Trees

Introduction to AVL Tree

- AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.
- Improved efficiency in search operation as compared to BST

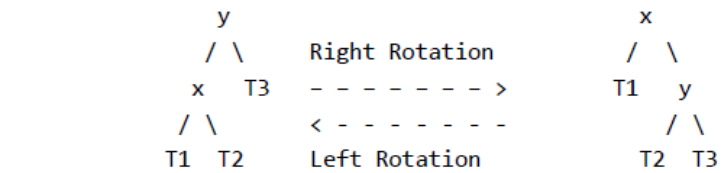
AVL tree Insertion

- To make sure that the given tree remains AVL after every insertion, we must augment the standard BST insert operation to perform some re-balancing.
- Two basic operations that can be performed to re-balance a BST without violating the BST property (keys(left) < key(root) < keys(right)).
  - Left Rotation
  - Right Rotation

3

AVL tree Insertion…

T1, T2 and T3 are subtrees of the tree rooted with y (on left side) or x (on right side)



Keys in both of the above trees follow the following order

keys(T1) < key(x) < keys(T2) < key(y) < keys(T3)

So BST property is not violated anywhere.

4

AVL tree Insertion…

Let the newly inserted node be w

1) Perform standard BST insert for w.

2) Starting from w, travel up and find the first unbalanced node.

Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.

3) Re-balance the tree by performing appropriate rotations on the subtree rooted with z.

5

AVL tree Insertion…

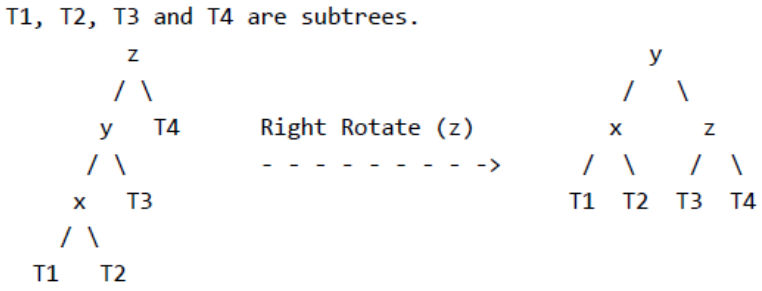
- There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways:
  a) y is left child of z and x is left child of y (Left Left Case)
  b) y is left child of z and x is right child of y (Left Right Case)
  c) y is right child of z and x is right child of y (Right Right Case)
  d) y is right child of z and x is left child of y (Right Left Case)

6

AVL tree Insertion…

- Operations to be performed

a) Left Left Case

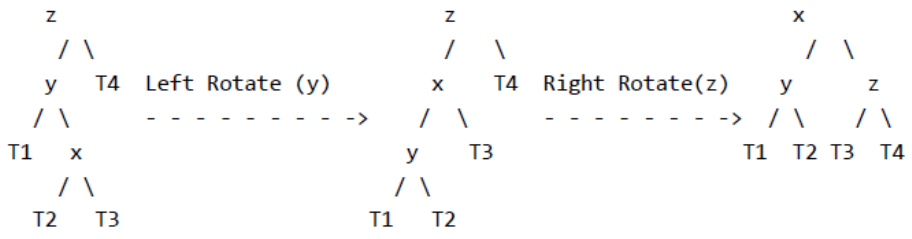


7

AVL tree Insertion…

- Operations to be performed

a) Left Right Case

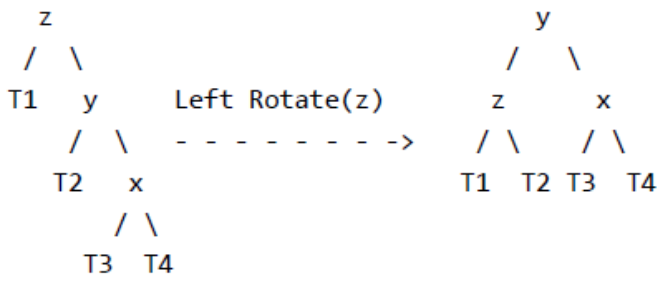


8

AVL tree Insertion…

- Operations to be performed

a) Right Right Case

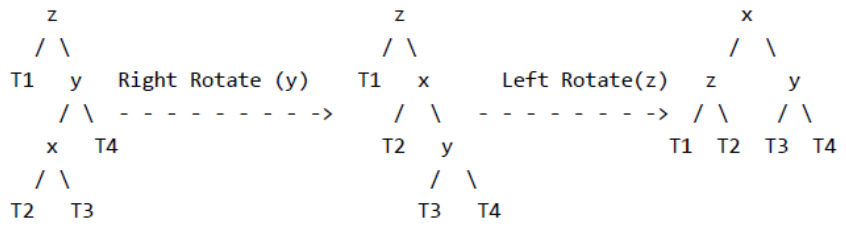


9

AVL tree Insertion…

- Operations to be performed

a) Right Left Case



10

AVL tree Deletion

- Let w be the node to be deleted
  1) Perform standard BST delete for w.
  2) Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the larger height child of z, and x be the larger height child of y.
  3) Re-balance the tree by performing appropriate rotations on the subtree rooted with z.

11

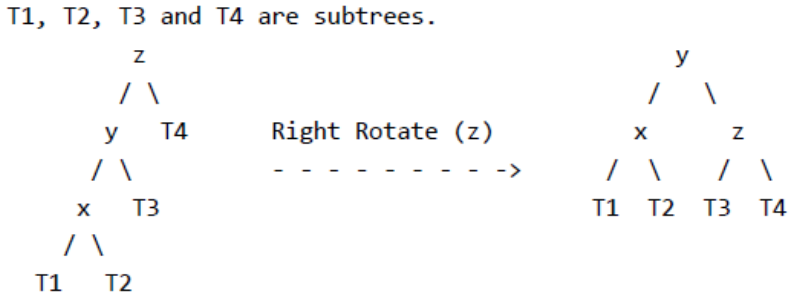
AVL tree Deletion…

- There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:
  a) y is left child of z and x is left child of y (Left Left Case)
  b) y is left child of z and x is right child of y (Left Right Case)
  c) y is right child of z and x is right child of y (Right Right Case)
  d) y is right child of z and x is left child of y (Right Left Case)

12

AVL tree Deletion…

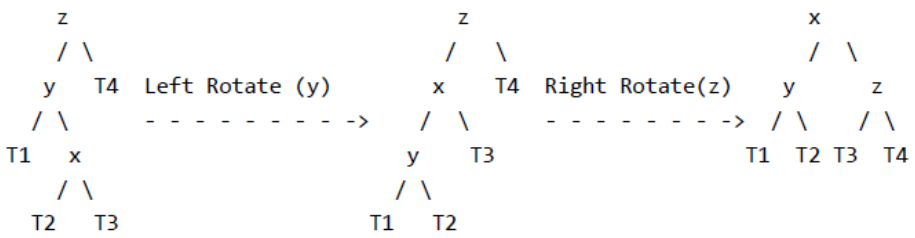
a)Left left case



13

AVL tree Deletion…

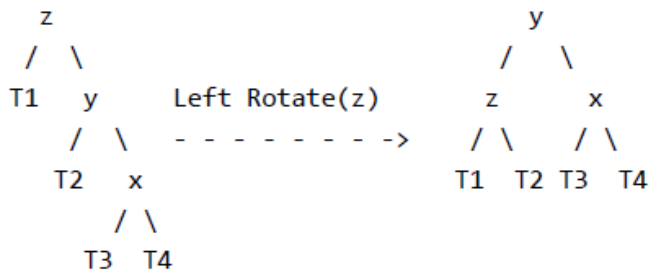
b)Left right case



14

AVL tree Deletion…

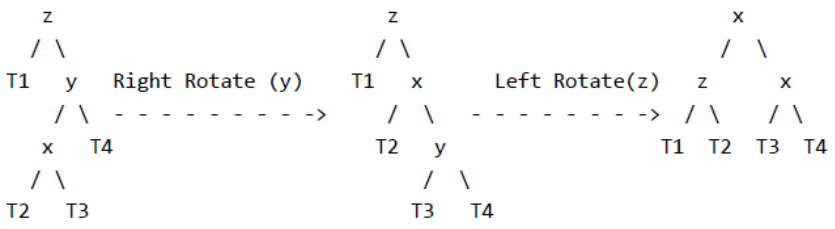
c)Right right case



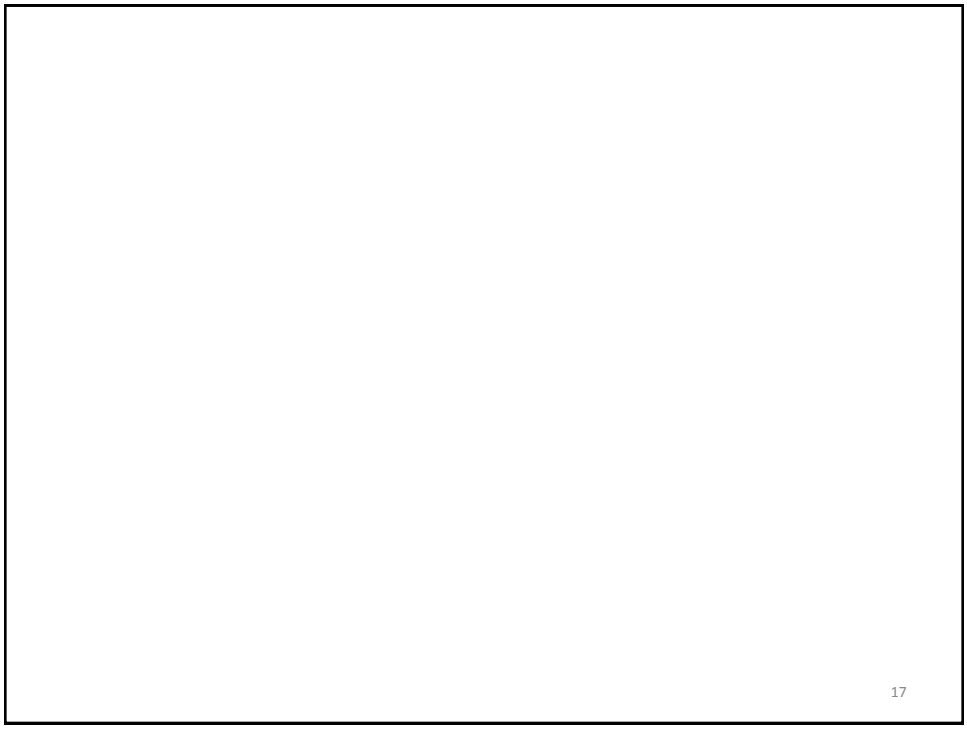
15

AVL tree Deletion…

d)Right left case



16


17