Tree

Instructors:
Md Nazrul Islam Mondal &
Rizoan Toufiq
Department of Computer Science & Engineering
Rajshahi University of Engineering &
Technology Rajshahi-6204

Outline

- Binary Tree
- Representing Binary Trees in Memory
- Traversing Binary Trees
- Traversal Algorithm using Stacks
- Header Nodes: Threads
- Binary Search Trees
- Searching and Inserting in Binary Search Trees
- Deleting in Binary Search Tree
- **AVL Search Trees**
- **Insertion in an AVL Search Tree**
- **Deletion in an AVL Search Tree**
- m-way Search Trees
- Searching, Insertion and Deletion in an m-way Search Tree
- B Trees
- Searching, Insertion and Deletion in a B-tree

AVL Search Trees

AVL Search Trees

- Right Skewed Binary Search Tree
- Left Skewed Binary Search Tree

Right skewed binary search tree
(a)

Light skewed binary search tree
(b)

AVL Search Trees

- **Disadvantage:** Worst case time complexity of a search is O(n).
- **Solution**: Maintain the binary search tree to be of balanced height.
- Popular balanced trees was introduced in **1962** by **Adelson,Velskii and Landis** and was known ad **AVL tree**.

AVL Search Trees

- **Definition:**
  - An Empty binary tree is an AVL Tree.
  - A non empty **binary tree** T is an **AVL tree** if and only if (iff)
    - Given T^L and T^R to be the left and right subtrees of T
    - $h(T^L)$ and $h(T^R)$ to be the heights of subtrees T^L and T^R respectively
    - T^L and T^R are **AVL trees** and $|h(T^L)-h(T^R)| \leq 1$
  - $h(T^L)-h(T^R)$ is known as the **balance factor** (BF)
  - For an AVL tree, the balance factor of a node can be either 0, 1 or -1
  - An AVL search tree is a binary search tree which is an AVL tree.

AVL Search Trees (Representation)

- AVL search tree like binary search trees are represented using a linked representation.

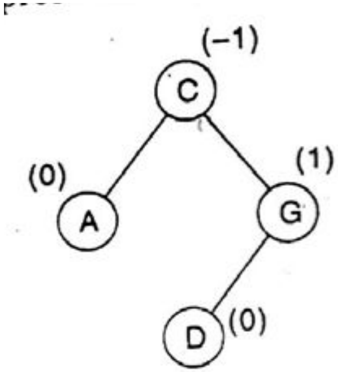


Fig. 7.30 AVL

- Balance Factor of D
  - $h(T^L)-h(T^R) = 0-0 = 0$
- Balance Factor of G
  - $h(T^L)-h(T^R) = 1-0 = 1$
- Balance Factor of A
  - $h(T^L)-h(T^R) = 0-0 = 0$
- Balance Factor of C
  - $h(T^L)-h(T^R) = 1-2 = -1$

AVL Search Trees (Searching)

- Similar to the method used in a binary search tree.

Insertion in an AVL Search Tree

Insertion in an AVL Search Tree

- **First Phase:** Insert an element into AVL binary search tree
- **Second Phase:** The balance factor of any node in the tree is affected, we resort to techniques called Rotation to resort the balance of the search tree.

- Example: If we insert E node -

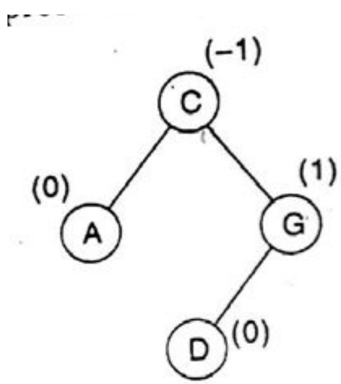
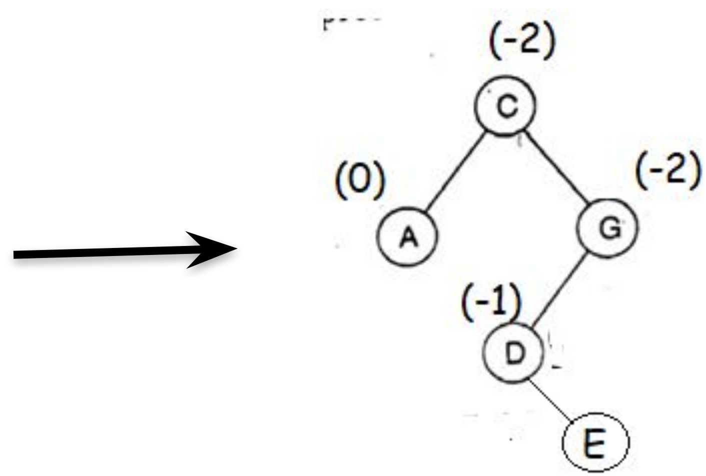


Fig. 7.30 *AVL*

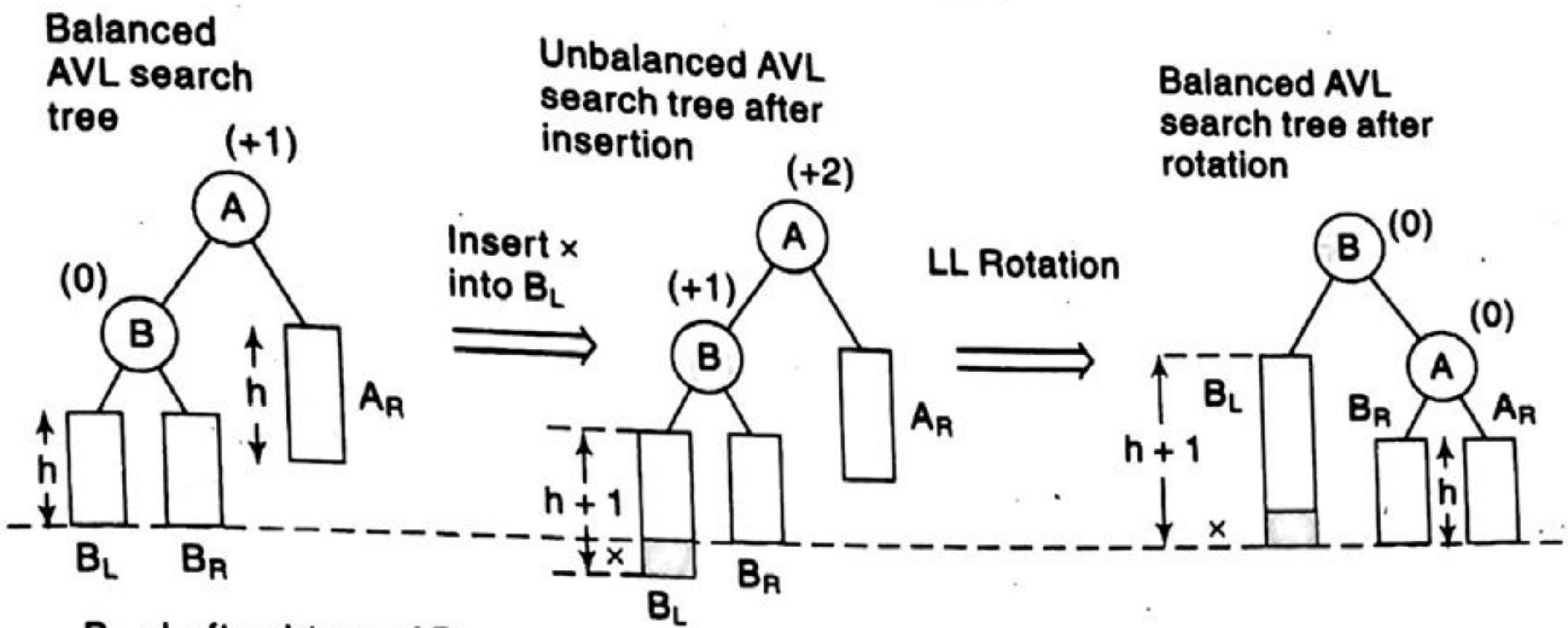


BF(C)=-2

Insertion in an AVL Search Tree

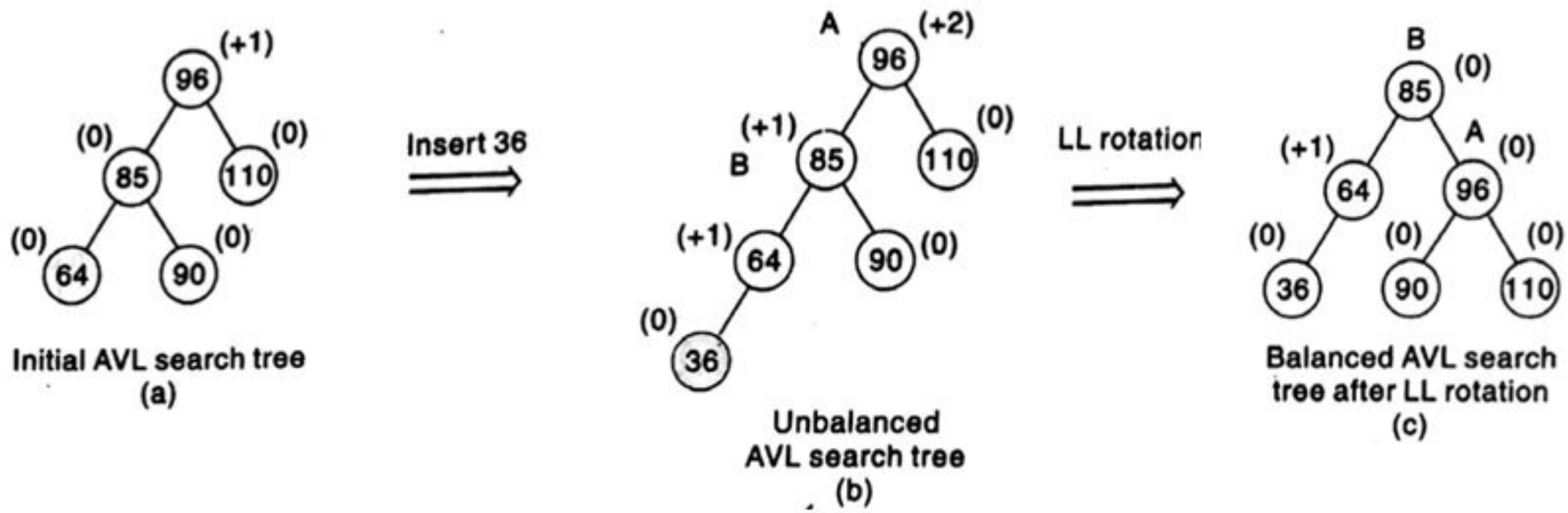
- To perform rotation, it is necessary to identify node A whose BF(A) is either 0, 1 or -1 and which is the nearest ancestor to the inserted node on the path from the inserted node to the root.
- The rebalancing rotations are classified as LL, LR, RR and RL based on the position of the inserted node with reference to A
  - **LL rotation:** Insert node is in the left subtree of the left subtree of node A
  - **RR rotation**: Insert node is in the right subtree of the right subtree of node A
  - **LR rotation**: Insert node is in the right subtree of the left subtree of node A
  - **RL rotation**: Insert node is in the left subtree of the right subtree of node A.

Insertion in an AVL Search Tree (LL Rotation)

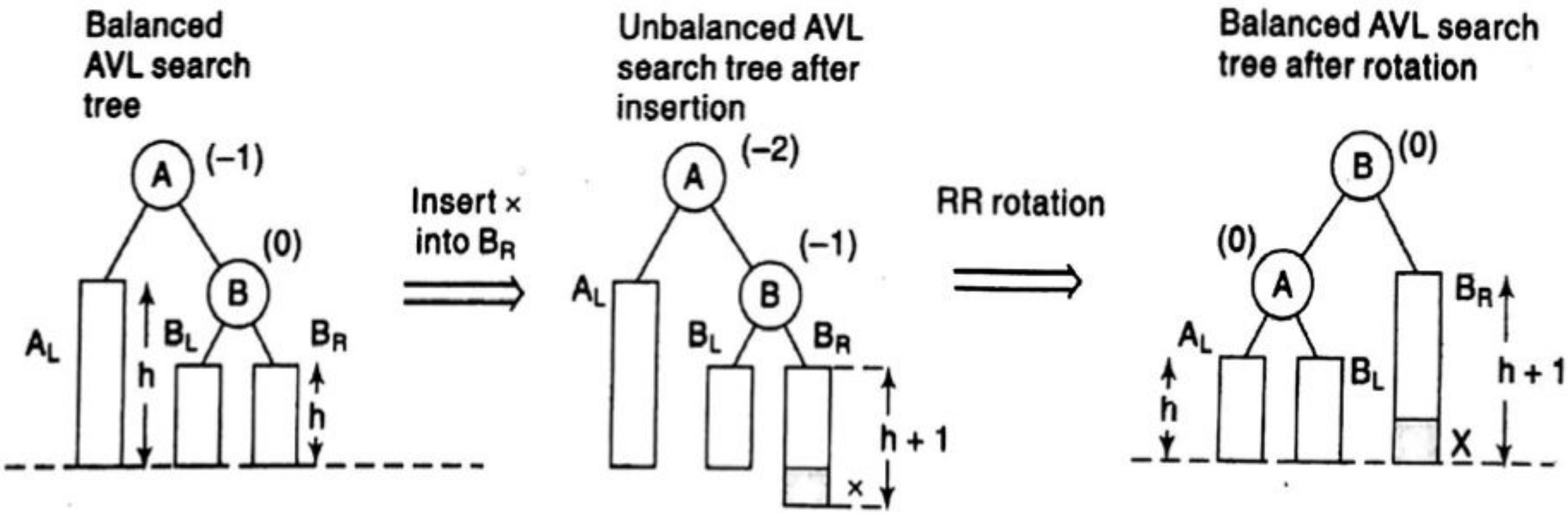


B_L : Left subtree of B
B_R : Right subtree of B
A_R : Right subtree of A
h : Height

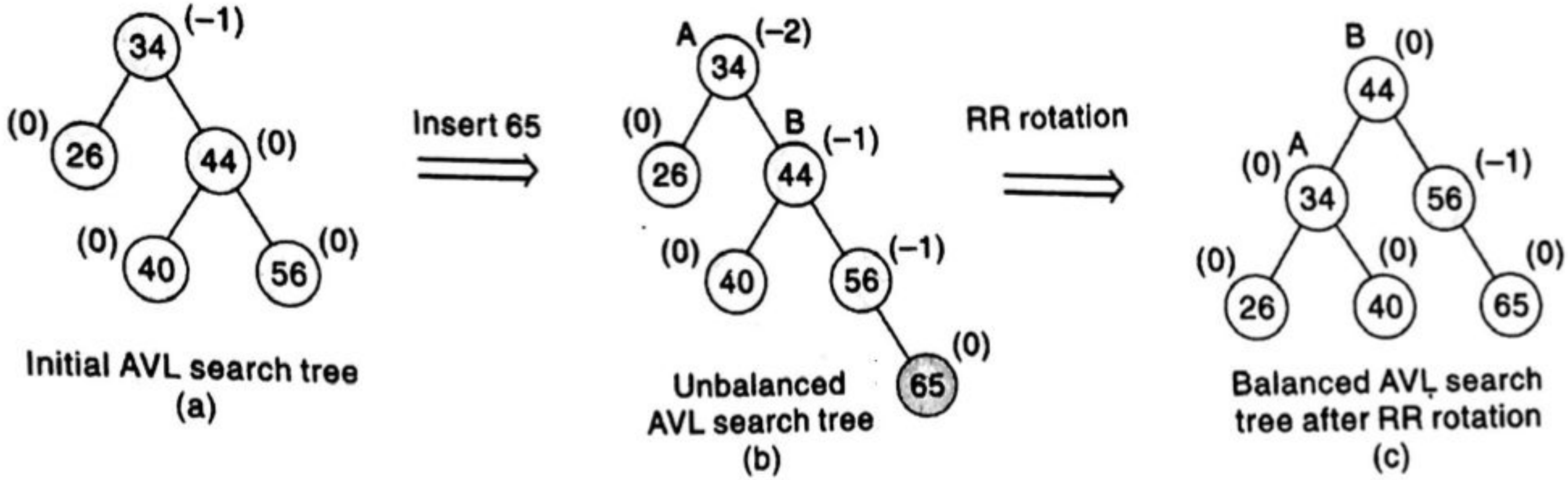
Insertion in an AVL Search Tree (LL Rotation)



Insertion in an AVL Search Tree (RR Rotation)



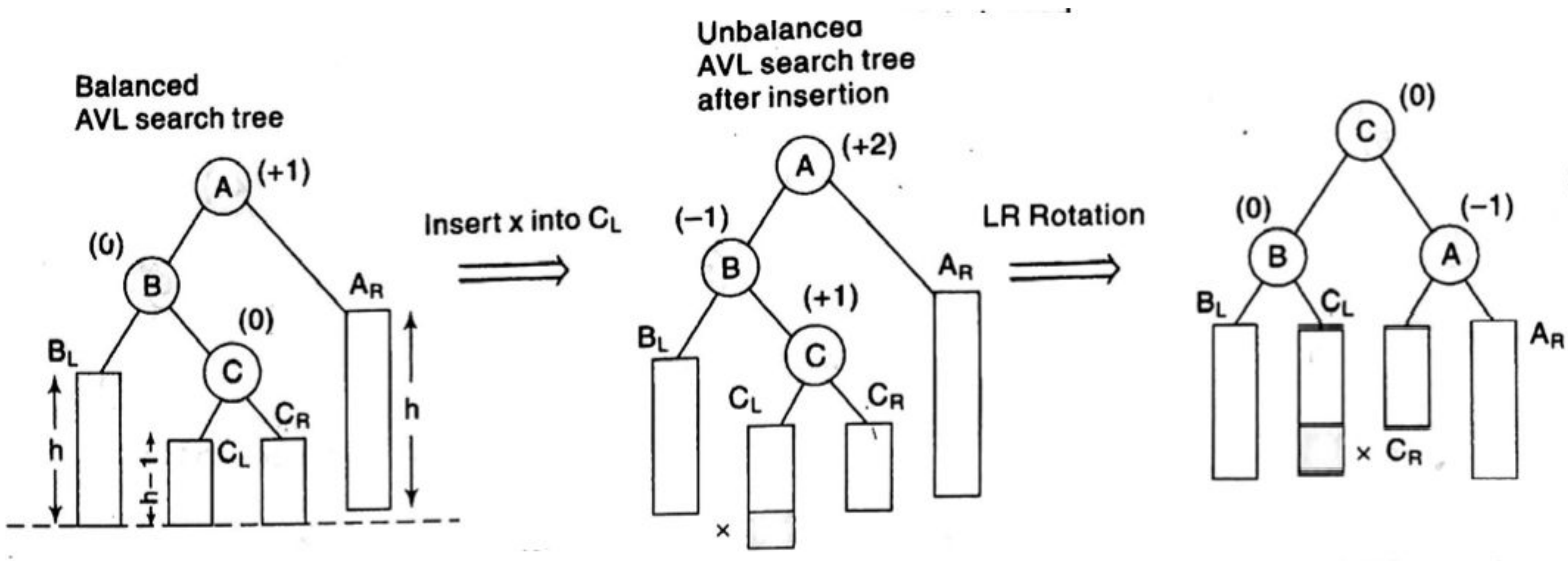
Insertion in an AVL Search Tree (RR Rotation)



Insertion in an AVL Search Tree (LR and RL Rotation)

- The balancing Methodology of LR and RL rotations are similar in nature but are mirror images of one another.
- Illustrates the balancing of an AVL search tree using LR rotation

Insertion in an AVL Search Tree (LR and RL Rotation)



C_L : Left subtree of C
C_R : Right subtree of C

Balanced AVL search tree after L_R rotation

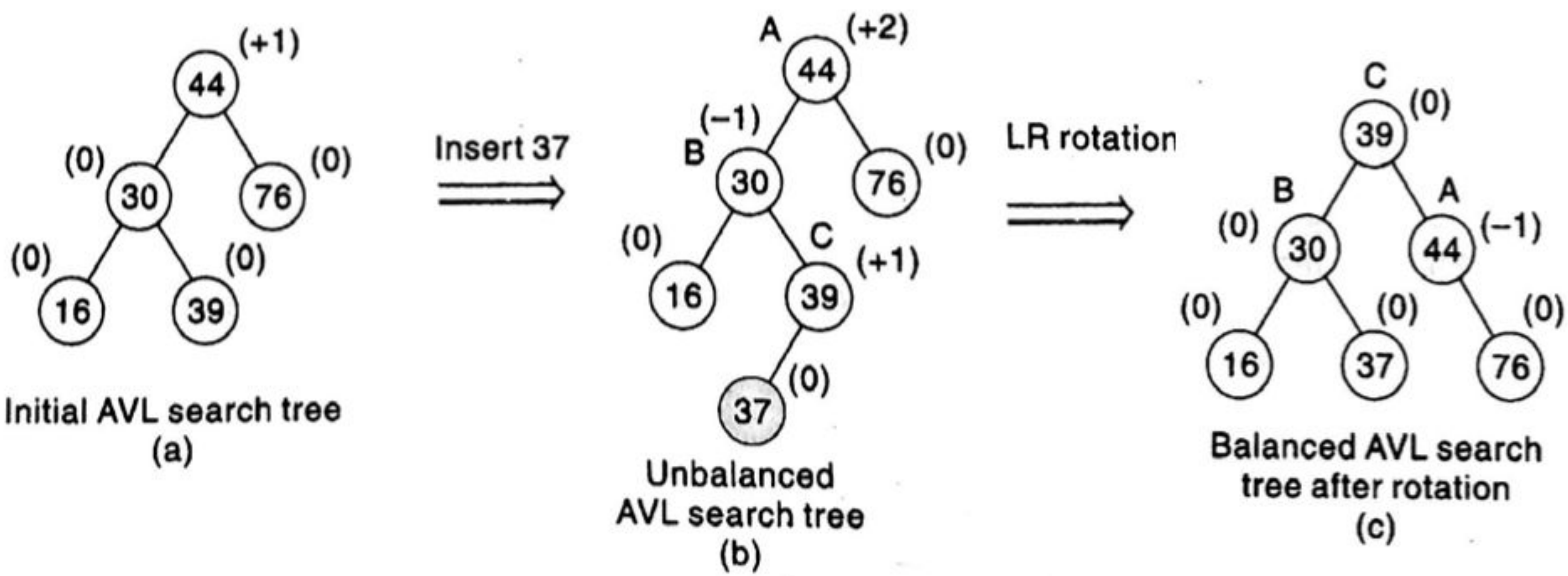
Insertion in an AVL Search Tree (LR and RL Rotation)

- In this case, the BF value of node A and B after balancing are dependent on the BF value of node C after insertion
  - If BF(C) = 0 after insertion then BF(A) = BF(B)=0, after rotation
  - If BF(C) = -1 after insertion then BF(A)=0, BF(B)=1, after rotation
  - **If BF(C)=1 after insertion then BF(A)=-1, BF(B)=0, after rotation (See Previous Slide)**

- LR and RL are known as **double rotation since**
  - **LR can be accomplished by RR followed by LL rotation** (See Previous Slide)
  - RL can be accomplished by LL followed by RR rotation

- LL and RR are known as **single rotation**

Insertion in an AVL Search Tree (LR and RL Rotation)



Initial AVL search tree
(a)

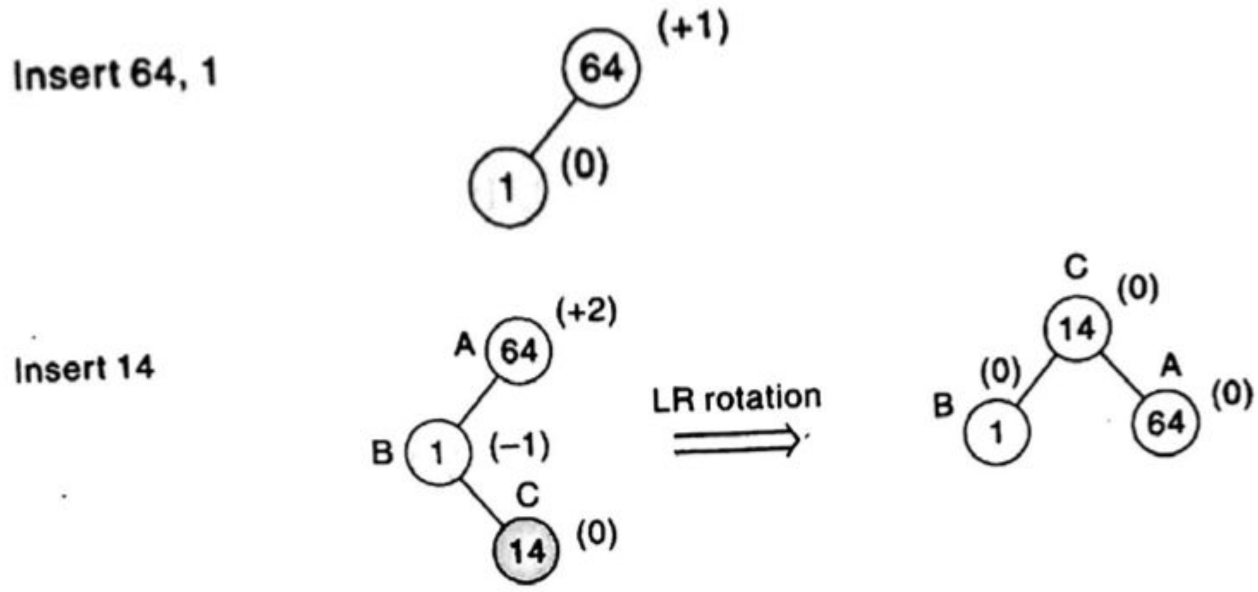
Unbalanced
AVL search tree
(b)

Balanced AVL search
tree after rotation
(c)

Insertion in an AVL Search Tree (Example)

- Example: Construct an AVL tree by inserting the following elements in the order of their occurrence.

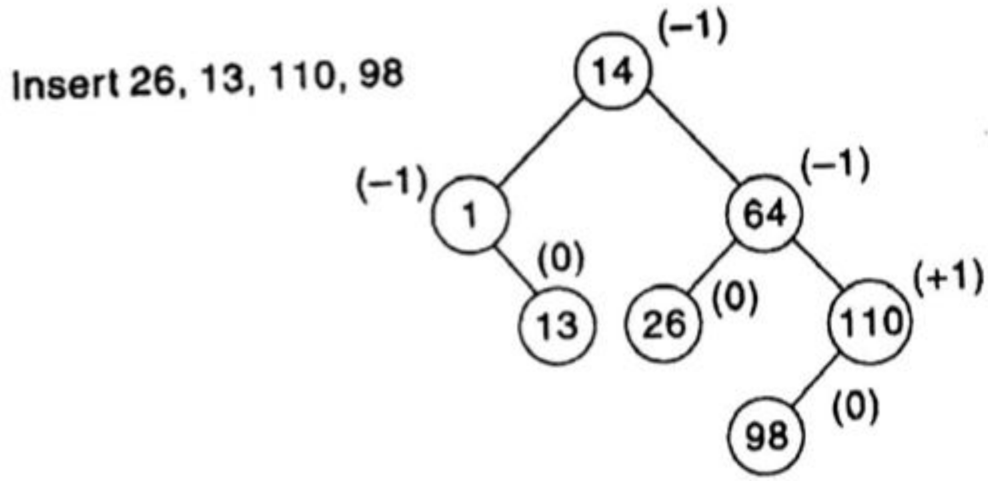
64, 1, 44, 26, 13, 110, 98, 85



Insertion in an AVL Search Tree (Example)

- Example: Construct an AVL tree by inserting the following elements in the order of their occurrence.

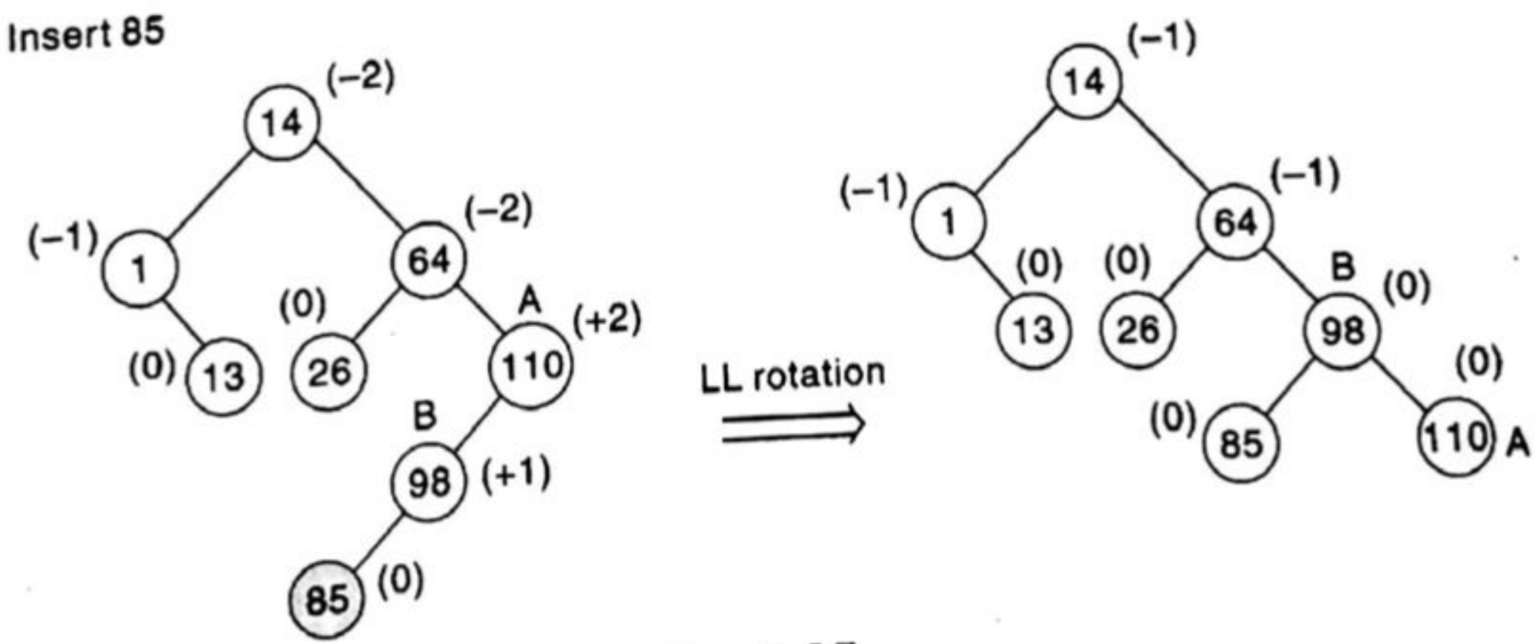
64, 1, 44, 26, 13, 110, 98, 85



Insertion in an AVL Search Tree (Example)

- Example: Construct an AVL tree by inserting the following elements in the order of their occurrence.

64, 1, 44, 26, 13, 110, 98, 85



Deletion in an AVL Search Tree

Deletion in an AVL Search Tree

- **First Step:** Delete an element from an AVL search tree likely as the procedure for deletion of an element in a binary search tree
- **Second Step:** Imbalance occurs due to deletion, one or more rotation need to be applied to balance the AVL search tree.

Deletion in an AVL Search Tree

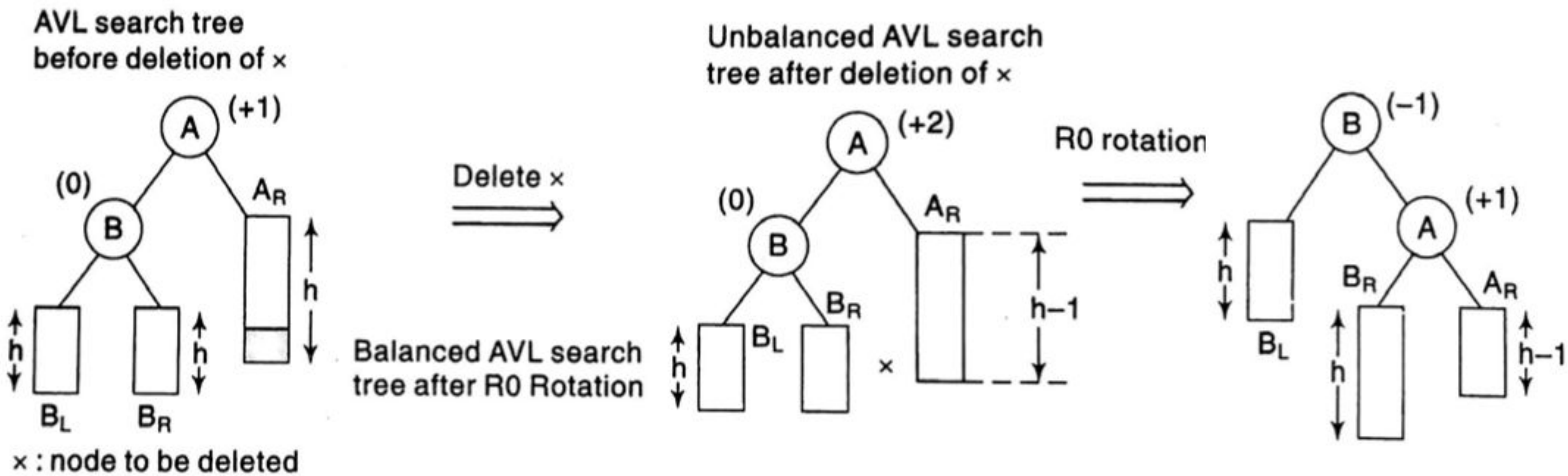
- Delete X node
- A – the closest ancestor node on the path from X to the root node, with balance factor of +2 or -2.
- L or R rotation required depending on whether the deletion occurred on the left or right subtree of A.

Deletion in an AVL Search Tree

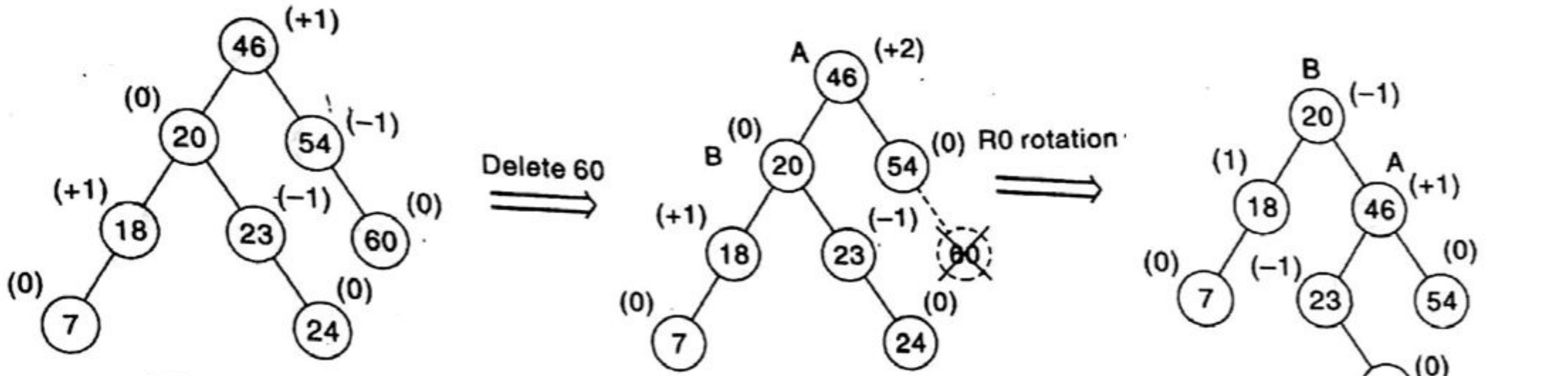
- Now depending on the value of BF(B), B is the root of the left or right subtree of A, the R or L imbalance is further classified as R0, R1 and R-1 or L0, L1 and L-1.
- The L rotation are the mirror image of three kinds of R rotation.

Deletion in an AVL Search Tree (R0 rotation)

- IF BF(B) = 0, R0 is executed



Deletion in an AVL Search Tree (R0 rotation)



AVL search tree before deletion

(a)

Unbalanced AVL search tree after deletion of 60

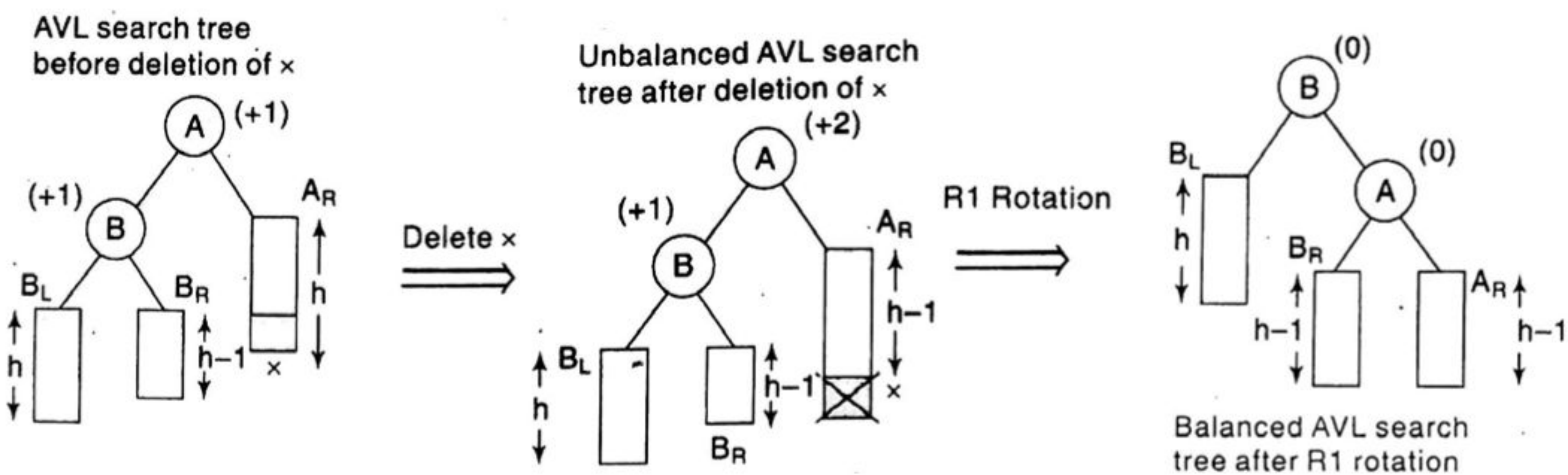
(b)

Balanced AVL search tree after R0 rotation

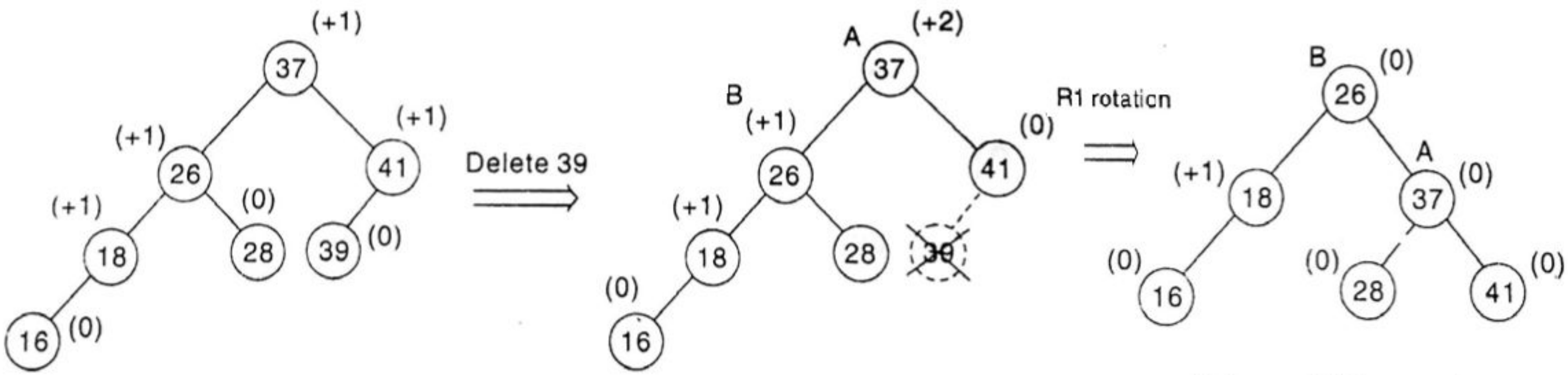
(c)

Deletion in an AVL Search Tree (R1 rotation)

- IF BF(B) = 1, R1 is executed



Deletion in an AVL Search Tree (R1 rotation)



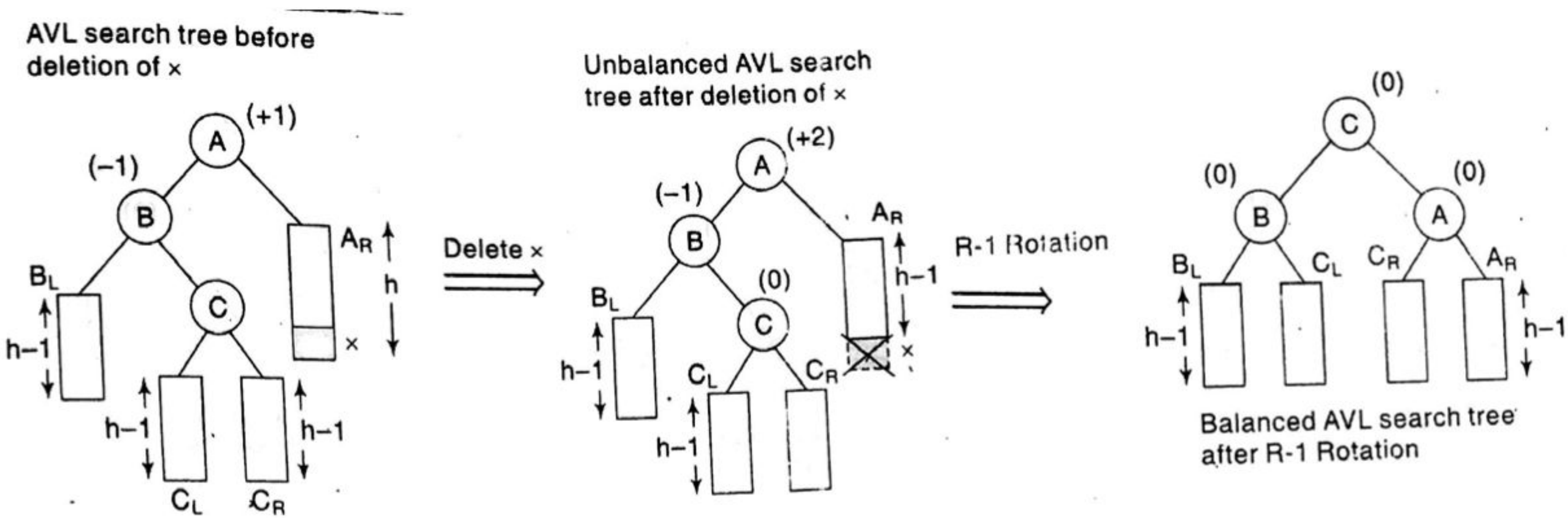
AVL search tree before deletion (a)

Unbalanced AVL search tree after deletion of 39 (b)

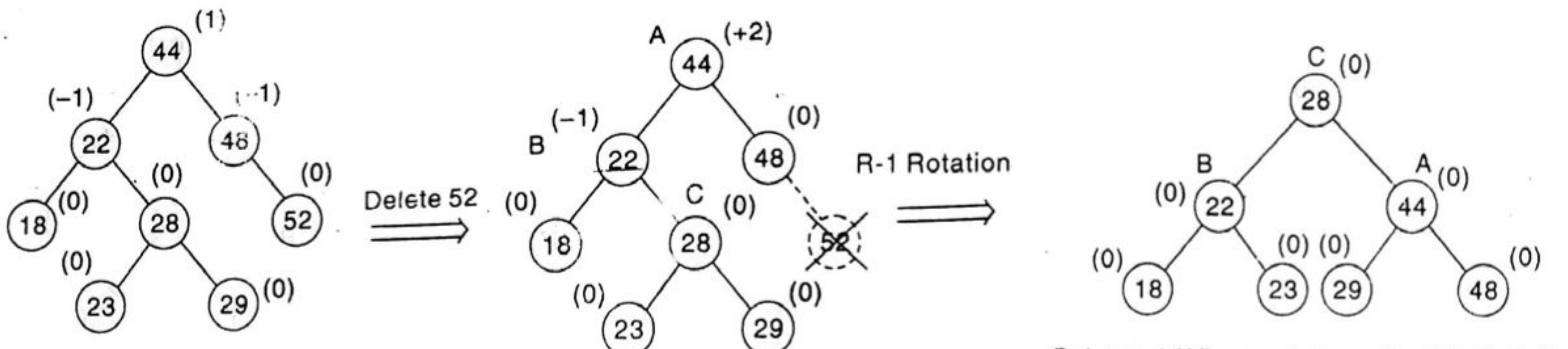
Balanced AVL search tree after R1 rotation (c)

Deletion in an AVL Search Tree (R-1 rotation)

- IF BF(B) = -1, R-1 is executed



Deletion in an AVL Search Tree (R-1 rotation)



AVL search tree before deletion (a)

Unbalanced AVL search tree after deletion (b)

Balanced AVL search tree after R-1 Rotation (c)

Any Query?