Trees in Data Structures

Presented By:
Pawan Kumar
Assistant Professor
GJUS&T, Hisar

Outlines

- **Introduction**
- **Binary Tree**
- **Binary Search Tree (BST)**
- **BST Operations**
  - **Searching**
  - **Insertion**
  - **Deletion**
  - **Traversal**
- **Complexity in BST**
- **Applications of BST**
- **Types of BST**

Data Structure Hierarchy

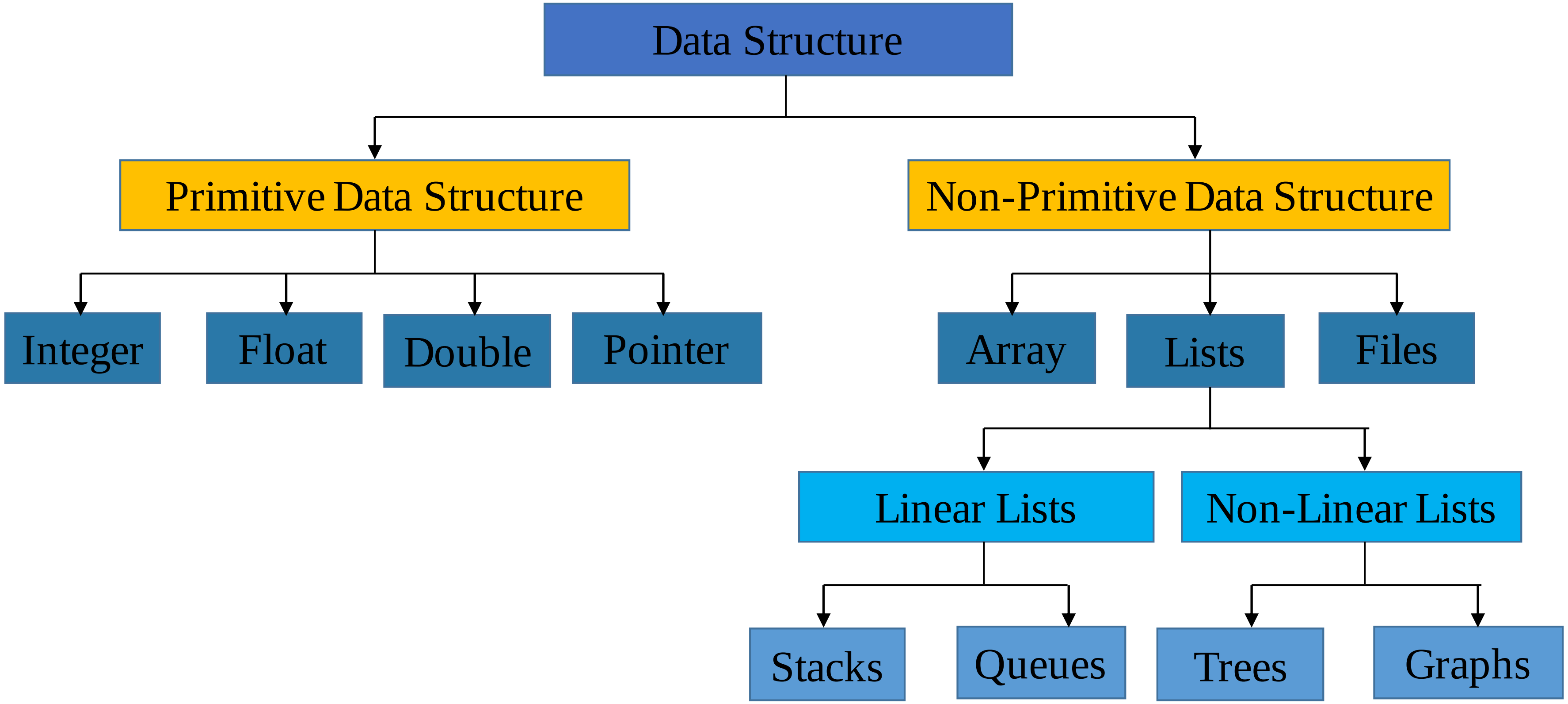


Figure 1: Data Structure Hierarchy

Tree

- **Tree – A tree can be defined as finite set of data items (nodes).**
- **Tree is a non-linear type of data structure in which data items are stored in order.**
- **It is mainly used to represent data containing a hierarchical relationship between elements.**
- **Each node in a tree can have 0 or more children.**
- **Each node can have at most one parent.**

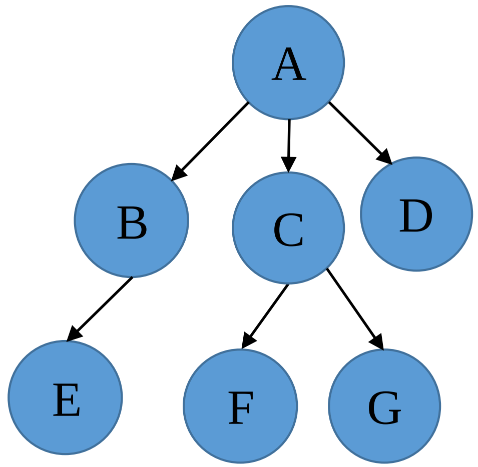


Figure 2: Tree

Terminology

- **Root → No parent (Level 0)**
- **Leaf → No Child**
- **Interior → Non-leaf**
- **Height → Distance from root to leaf**

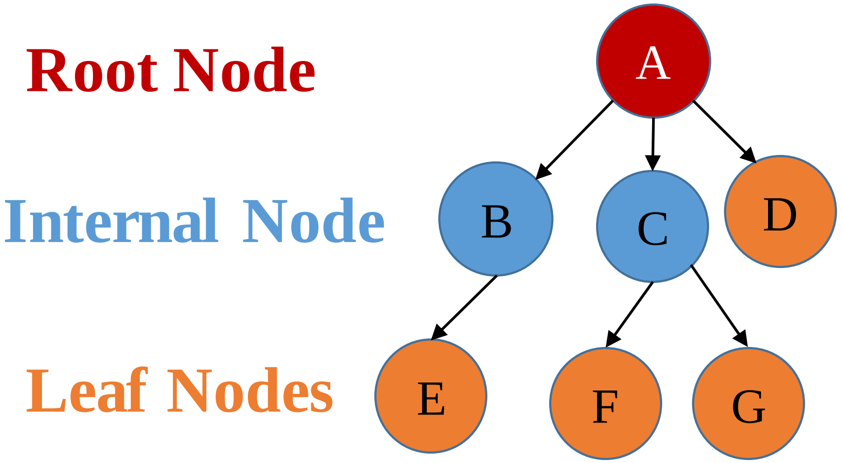


Figure 3: Terminology

Binary Tree (BT)

- Binary tree is also finite set of data items.
- Binary tree is a tree having maximum of 2 children.
- Each node can have either 0 or 1 or 2 child only.
- Also known as Decision Making Tree.
- It is helpful in designing routing protocols and other applications.

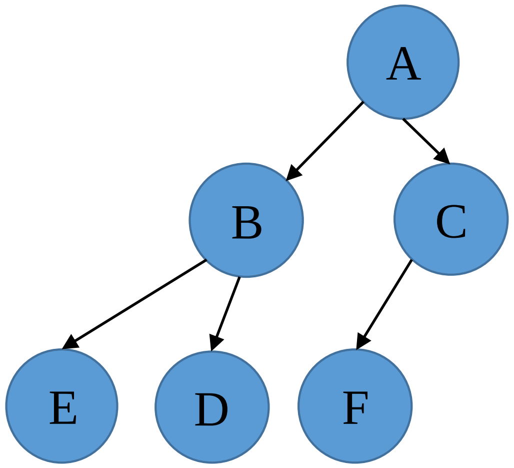


Figure 4: Binary Tree

- **Strict Binary Tree (SBT)**– A binary tree in which every node has either zero or two nodes.
- It is also known as Full Binary Tree, Proper Binary Tree or 2-Tree.
- 2-tree states that a node either can have 0 or 2 child only.

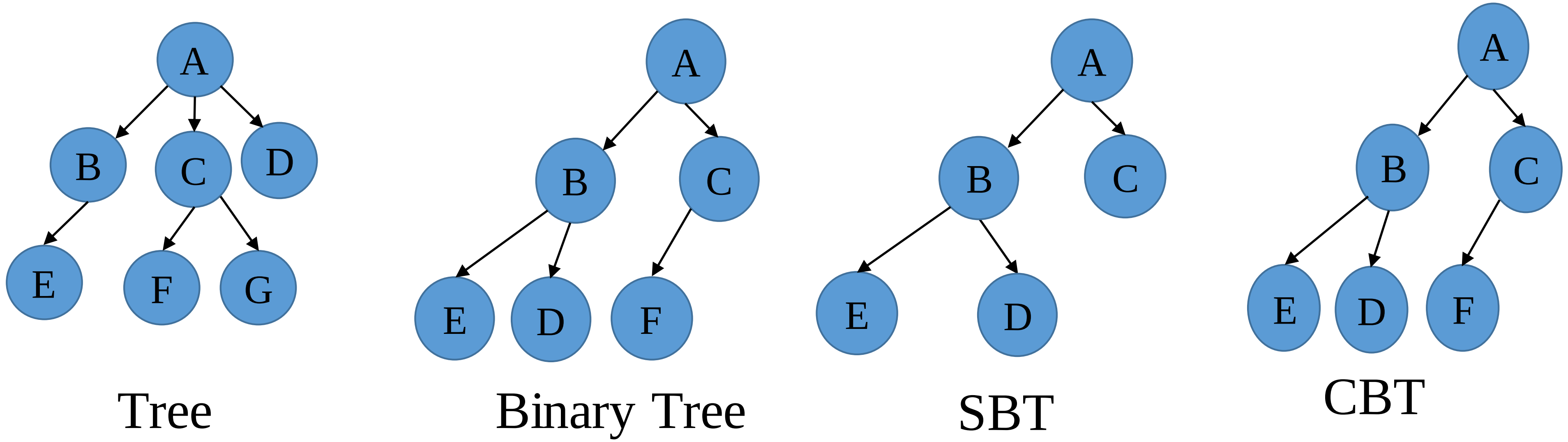
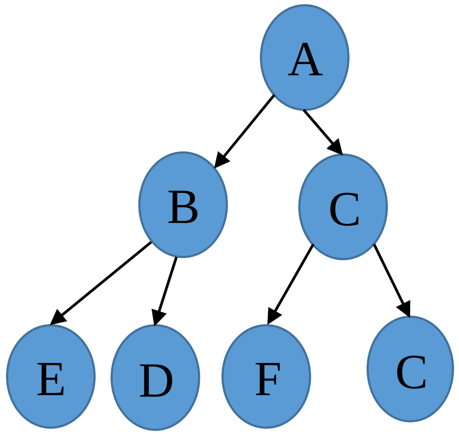


Figure 5: Trees

Complete Binary Tree (CBT)

- **Complete Binary Tree (CBT)**– A binary tree in which every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible.



Binary Search Tree (BST)

- **Binary Search Tree is a binary tree which is either empty or satisfies following rules:**
  - **Value of key on left is smaller than root**
  - **Value of key on right is greater than or equal to root**
- **All the sub-trees of the left and right children follow the above two rules.**
- **It was invented by P.F. Windley, A.D. Booth, A.J.T. Colin, and T.N. Hibbard in 1960.**

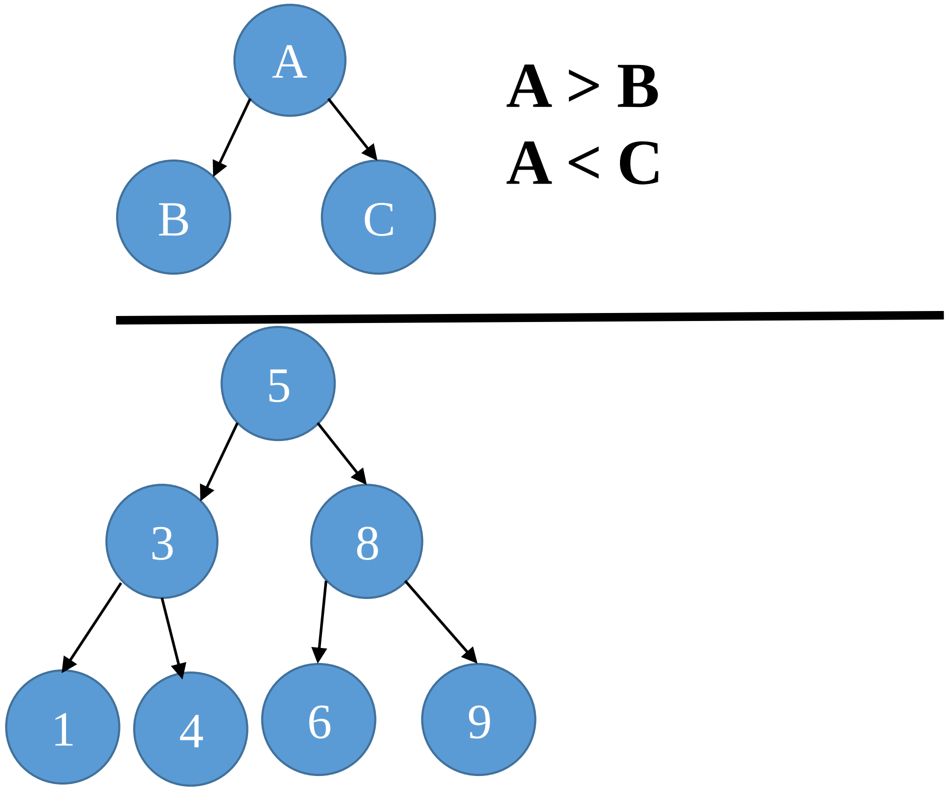


Figure 6: BST Example

Difference Between BT and BST

- **Binary Tree is simply a tree in which each node can have at most two children.**
- **Binary Search Tree is a binary tree in which the nodes are assigned values, with the following restrictions :**
  - No duplicate values.
  - The left subtree of a node can only have values less than the node.
  - The right subtree of a node can only have values greater than the node and recursively defined.
  - The left subtree of a node is a binary search tree.
  - The right subtree of a node is a binary search tree

Binary Search Tree (BST) Operations

- **Four basic operations are performed on BST**
  - **Search**
  - **Insertion**
  - **Deletion**
  - **Traversal**

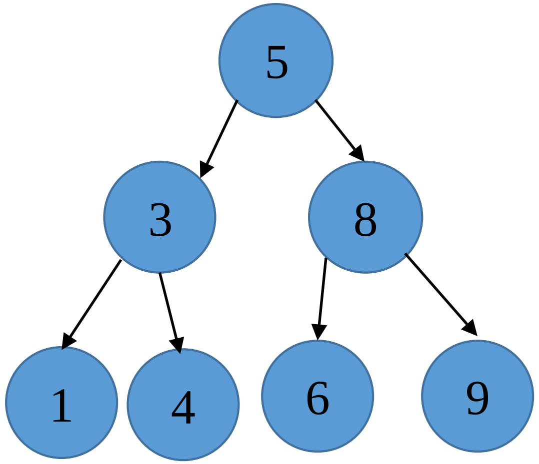
Binary Tree Search

- **Searching a node K in a BST follow the rules:**
- **Compare the key with root node, if root X is empty or equal to key K then return X**
- **If key K is smaller than root X , then search again on left subtree of root X.**
- **If key is matched than return the node.**
- **Algorithm to search key**
    - TREE-SEARCH(x,k)

      ```
      If x==NIL or k==x.key
      return x
      If k < x.key
      return TREE-SEARCH(x.left,k)
      else
      return TREE-SEARCH(x.right,k)
      ```

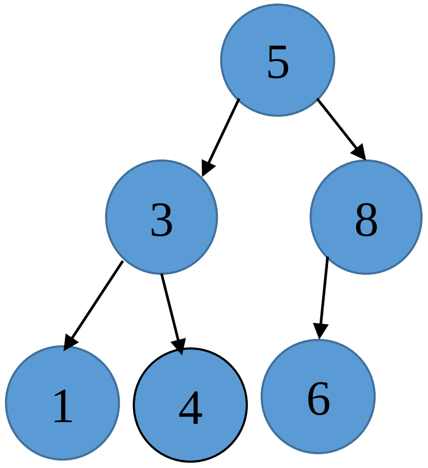
Binary Search Tree Insertion

5	**3**	**1**	**4**	**8**	**6**	**9**

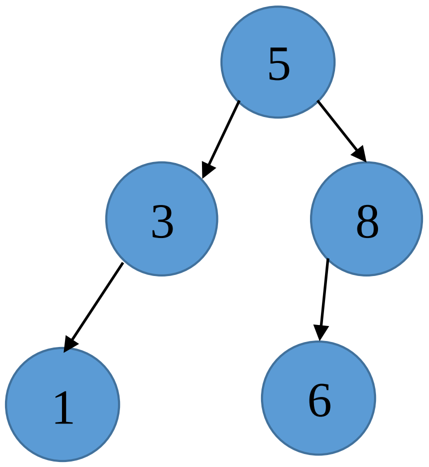


BST Deletion

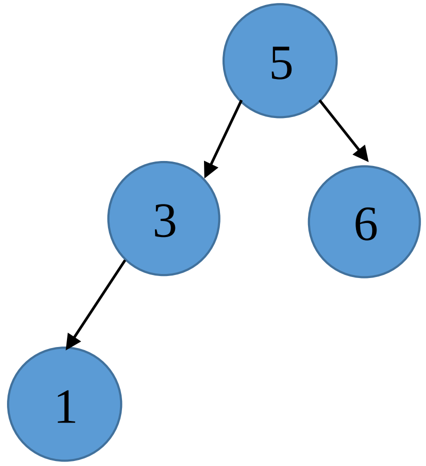
- **Case 1: Element X has no children, then simply delete X and update location of X in parent node P(X) by null pointer.**
- **X has exactly one child, then delete X and replace location of X in P(X) by location of single child.**
- **Delete the element 4, 8 from the given tree**



Delete 4 →

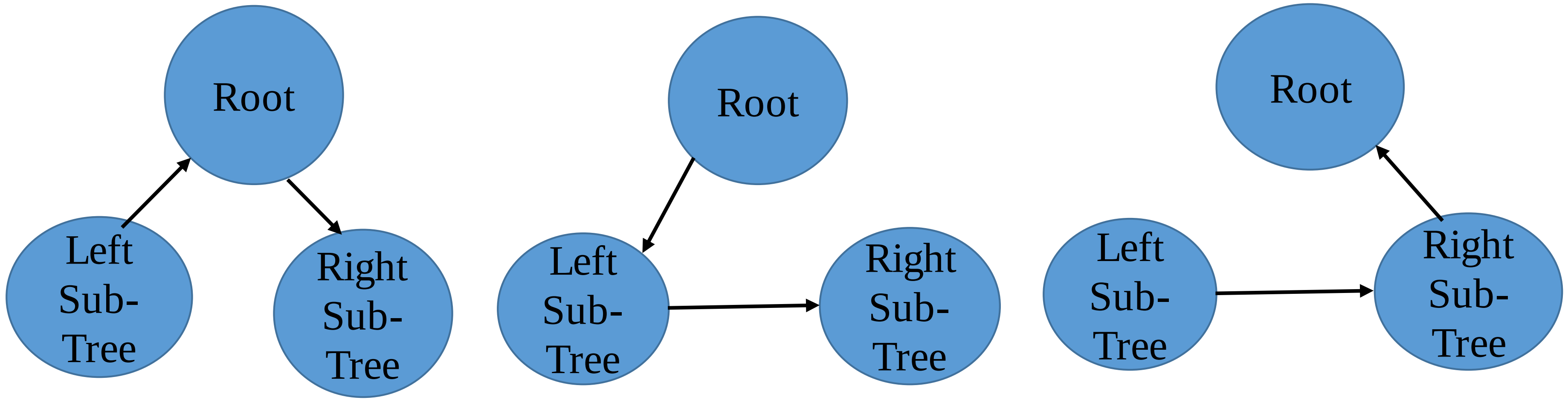


Delete 8→



Binary Search Tree Traversal

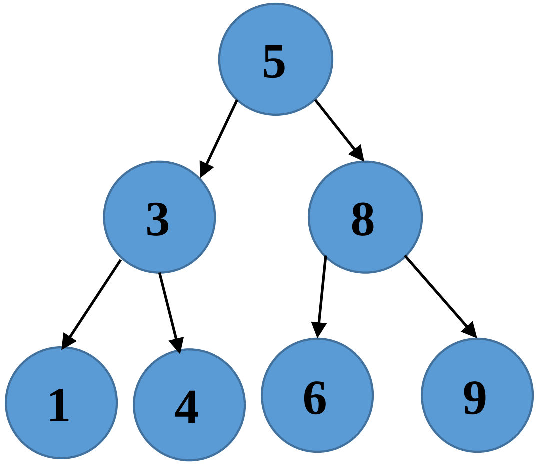
- **Traversal is one of the most common operation on tree data structure in which each node in the tree is visited exactly once in a systematic manner.**



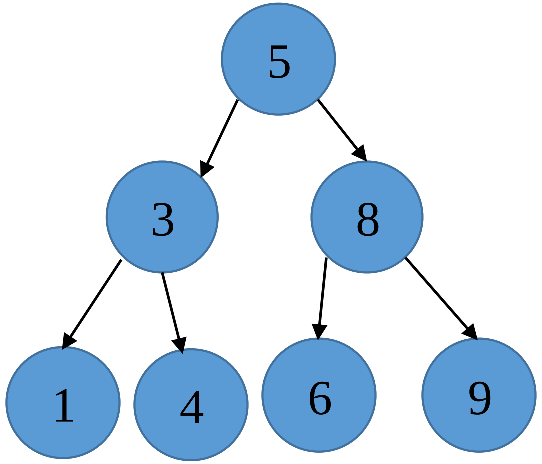
Inorder traversal **Preorder traversal** **Postorder traversal**

Inorder Traversal

- **Inorder Traversal –**
  - **Traverse left subtree in Inorder (L)**
  - **Visit the root node (N)**
  - **Traverse right subtree in Inorder (R)**



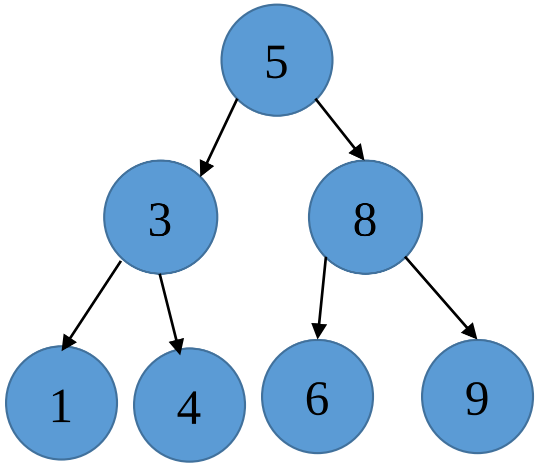
Left→ Root→ Right



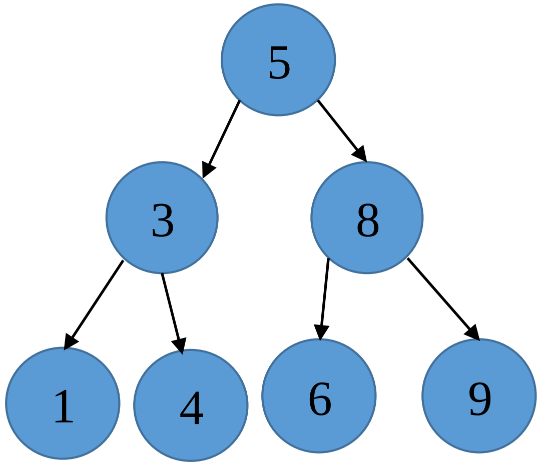
1	**3**	**4**	**5**	**6**	**8**	**9**

Preorder Traversal

- **Preorder Traversal** –
  - Visit the root node **(N)**
  - Traverse left subtree in **Preorder (L)**
  - Traverse right subtree in **Preorder (R)**



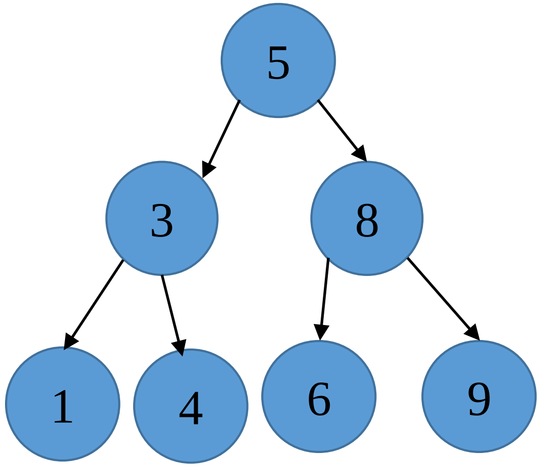
Root→ Left→ Right



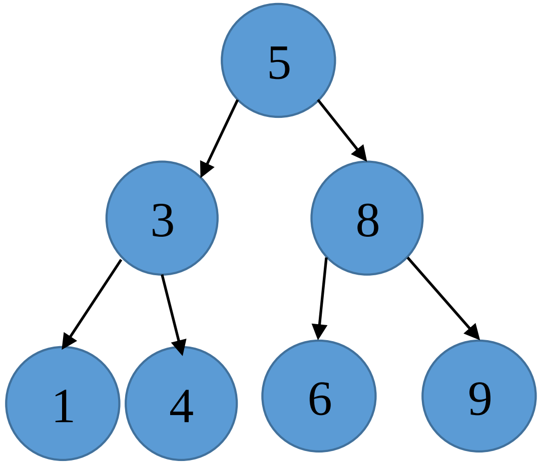
5	**3**	**1**	**4**	**8**	**6**	**9**

Postorder Traversal

- **Postorder Traversal** –
  - Traverse left subtree in **Postorder (L)**
  - Traverse right subtree in **Postorder (R)**
  - Visit the root node **(N)**



Left→ Right→ Root



1	**4**	**3**	**6**	**9**	**8**	**5**

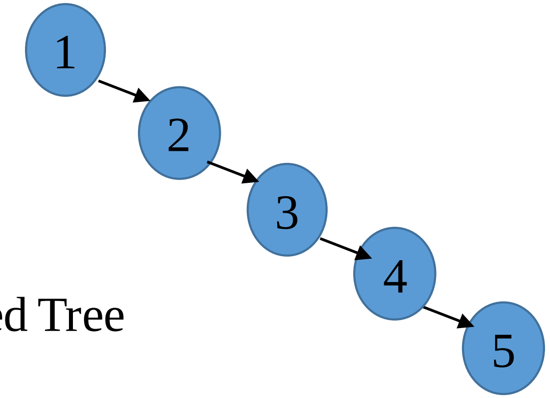
Advantages of BST

- **Cost for N nodes BST is O(log n) for the operations Search(), Insert(), Delete() as comparing to Binary tree having O(n).**

Disadvantages of BST

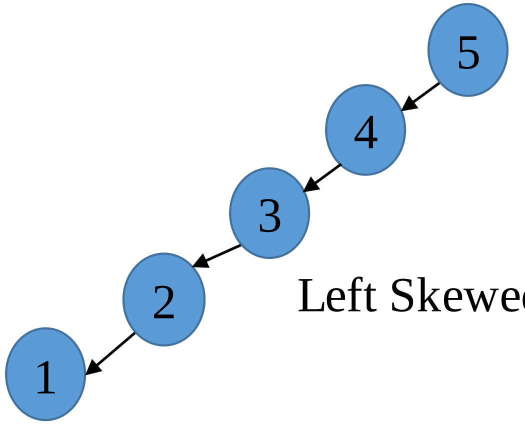
- **If the element which is inserted to be next is greater than previous item then we will get right skewed tree.**
- **If the element which is inserted to be next is smaller than previous item then we will get left skewed tree.**

1	**2**	**3**	**4**	**5**



Right Skewed Tree

5	**4**	**3**	**2**	**1**



Left Skewed Tree

Complexity in BST

Operation	Average Case	Worst Case	Best Case
Search	O(Log n)	O(n)	O(1)
Insertion	O(Log n)	O(n)	O(1)
Deletion	O(Log n)	O(n)	O(1)

Applications of Binary Tree

- Used in many search applications where data is constantly entering/leaving, such as map and set objects in many language's libraries.
- **Binary Trees –** Used in almost every high bandwidth router for storing routing tables.
- **Hash Trees** – Used in P2P programs in which hash needs to be verified.
- Heaps – Used in implementing efficient priority queues.
- **Huffman Coding** – Used in compression algorithms such as used by .jpeg, .mp3 file format.
- **Syntax Tree** – Constructed by compilers and (implicitly) calculators to parse expressions.
- T-tree – Most databases used some form of B-Tree to store data on the drive

Types of BST

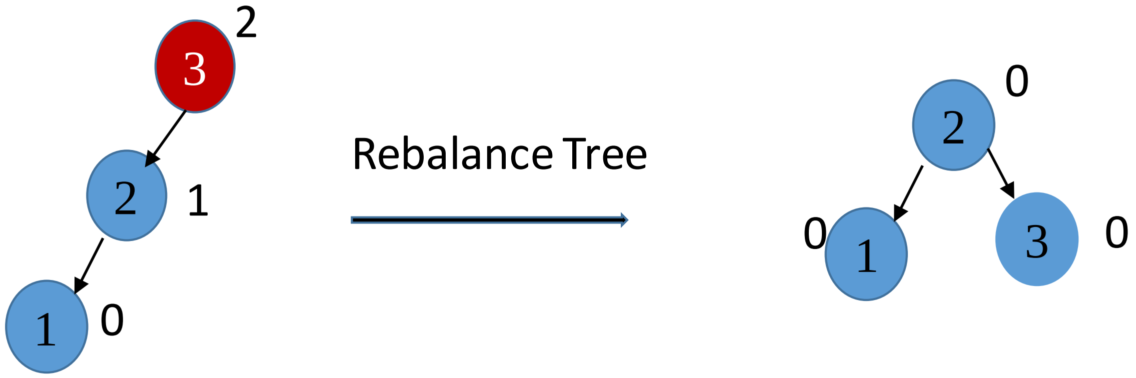
- **AVL Tree**
- **Red Black Tree**
- **Splay Tree**

AVL Tree

- **To overcome skewness problem in BST, AVL tree concept came into existence in 1962 invented by Adelson Velsky and Evgenii Landis.**
- **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.**
- **Balancing Factor (BF)**
  - **BF ranges - (-1,0,1)**
  - **BF = Height of left subtree - Height of right subtree**
- **Time Complexity: O(Log n)**
- **AVL tree mainly faces four unbalancing issues:**
  - **LL (left-left) Issue**
  - **RR (right-right) Issue**
  - **LR (left-right) Issue**
  - **RL (right-left) Issue**

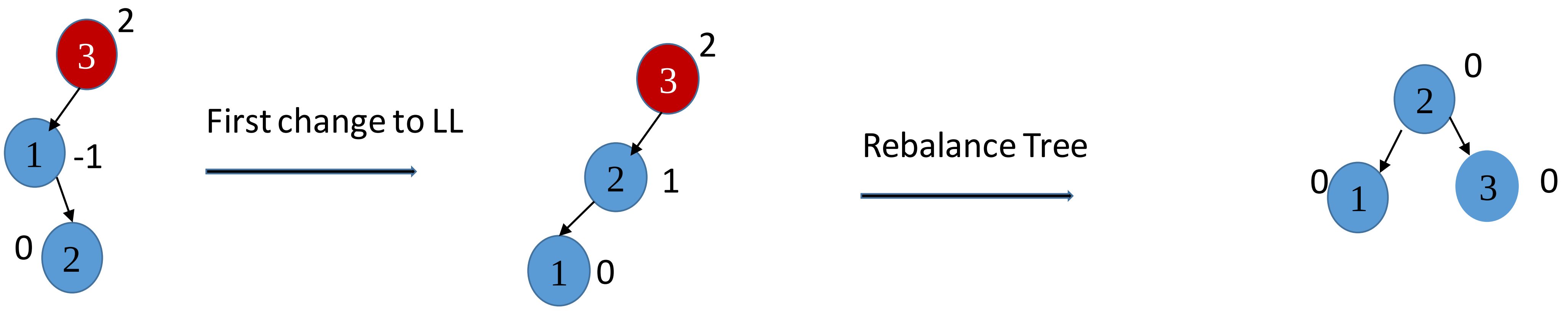
AVL Tree

- **LL problem:** **Occur due to insertion on left of left subtree**
- **When a new node is inserted, calculate BF of all nodes on path from root to newly inserted node**
- **Consider the node as pivot node whose BF is 2 or -2 and rebalance it by rotating pivot node on right of its left child.**



AVL Tree

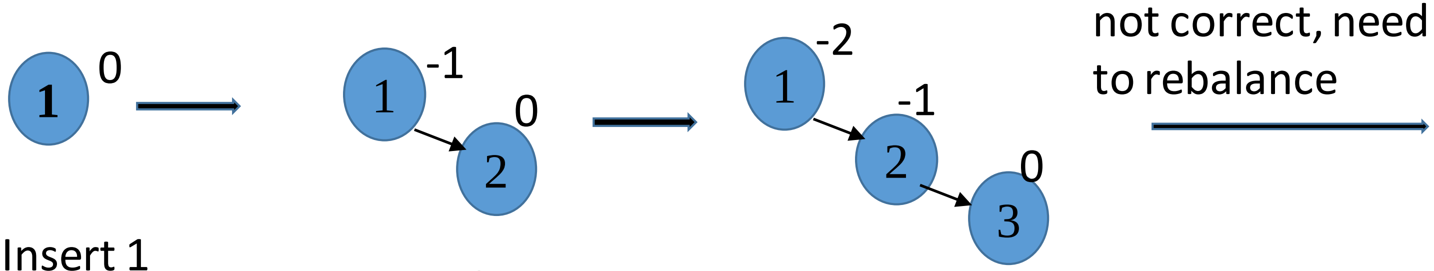
- **LR problem: Occur due to insertion on right of left subtree**
- **When a new node is inserted, calculate BF of all nodes on path from root to newly inserted node**
- **Consider the node as pivot node whose BF is 2 or -2.**
- **To rebalance it first change in LL problem then balance the tree.**



AVL Tree

Elements to insert

1	**2**	**3**	**4**	**5**



Insert 1

Insert 2

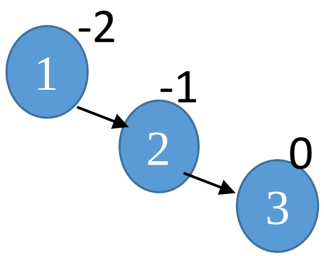
Insert 3

BF of one node is not correct, need to rebalance

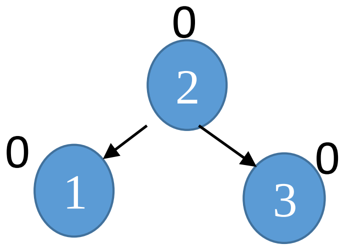
AVL Tree

Elements to insert

1	**2**	**3**	**4**	**5**



BF of one node is not correct, need to rebalance

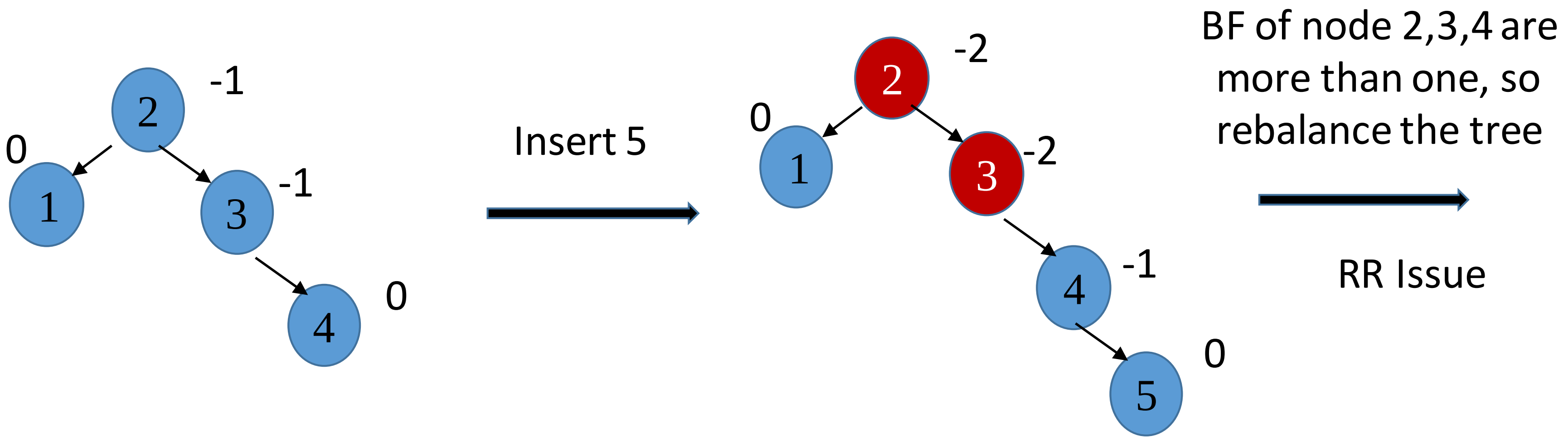


Insert 4

AVL Tree

Elements to insert

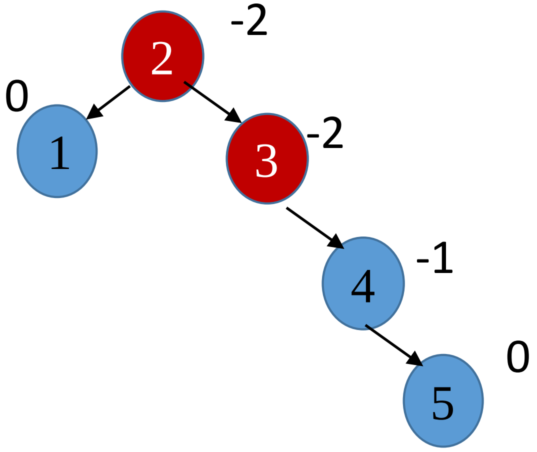
1	**2**	**3**	**4**	**5**



AVL Tree

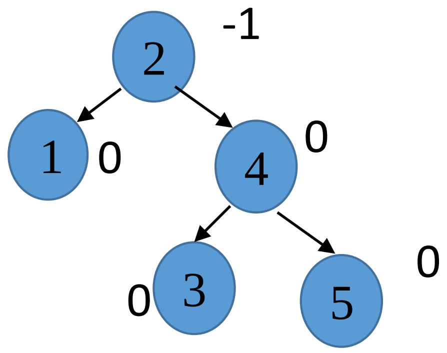
Elements to insert

1	**2**	**3**	**4**	**5**



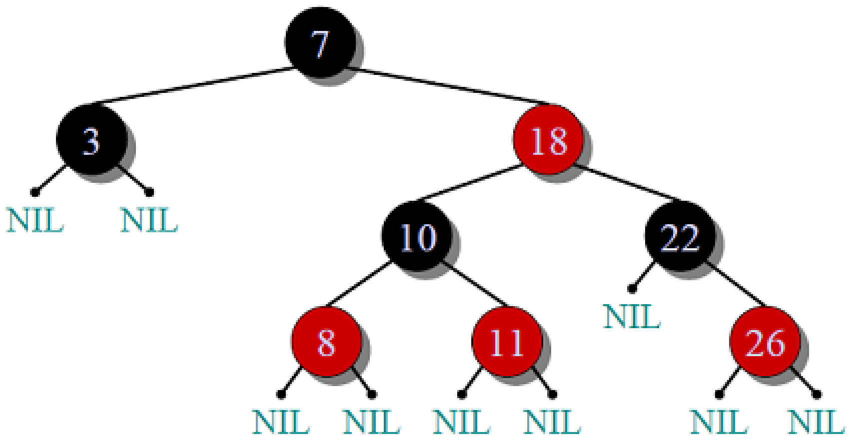
BF of node 2,3,4 are more than one, so rebalance the tree

RR Issue



Red Black Tree

- **Every node has a color either red or black.**
- **Root of tree is always black.**
- **There are no two adjacent red nodes (A red node cannot have a red parent or red child).**
- **Every path from root to a NULL node has same number of black nodes.**
- **Time Complexity: O(Log n)**



Splay Tree

- **Automatically moves frequently accessed elements nearer to the root for quick to access.**
- **Search operation in Splay tree does the standard BST search, in addition to search, it also splays (move a node to the root).**
- **Time Complexity – O(log n)**
- **Splay trees are used in Windows NT (in the virtual memory, networking, and file system code)**

References

1. **Seymour Lipschutz, *Data Structures with C*, Tata McGraw-Hill Pvt. Ltd., 2011.**
2. **Sartaj Sahni, *Data Structure, Algorithms and Applications 2nd Edition*, Universities Press Pvt. Ltd., 2005.**
3. **Debasis, Samanta. *Classic Data Structures 2nd Edition*, PHI Learning Pvt. Ltd., 2009.**