**TREE**

* Stores the data in hierarchy structure.
* It is recursive in nature, i.e., tree itself contains many trees known as subtrees.
* Degree of a node is number of children it has.
* Leaf node has degree as zero.

**Applications**:

* To represent hierarchy structure like, Org structure, Folder structure , XML/HTML,JSON Objects, Inheritance in OOP.
* Binary search tree.
* Binary Heap for priority queues.
* B and B+ tree in DBMS for indexing.
* Spanning tree and shortest path trees in computer networks.
* Parse tree, Expression tree in compilers.
* Trie for dictionary, prefix search.
* Suffix tree for pattern search.
* Binary index tree and Segment tree for range query searches.

**Binary tree:** Degree of each node is at most 2. Tree traversals can be of two types:

* Breadth first (level order)
* Depth first:
  + Inorder: left->root->right
  + Preorder: root->left->right
  + Postorder: left->right->root

InOrder and PreOrder/PostOrder traversals uniquely identify a binary tree.

**Binary Search Tree:** Binary tree with condition that for every node ,keys in left subtree are smaller then it and on right subtree are greater than it. In java Treeset and TreeMap implements self-balancing binary search tree.

Inorder traversal of BST is sorted.

Self-balancing BST : They try to maintain height of tree as close to log(n) by restructuring(left or right rotation of tree) at insertion and deletion. E.g Avl ,Red black tree. In Java treeset,TreeMap implements Red black tree which does less restructuring compared to AVL tree.

For AVL tree , for every node Balance factor should be atmax 1

Balance factor =| lh -rh |