Tree

TREE

Binary Tree

- Terms
  - Depth
  - Height
  - Subtree
  - Ancestor
  - Leaf node
  - Root node
  -

Types of Binary Tree

- Complete
- Nearly complete
- Strict
- Binary Search Tree

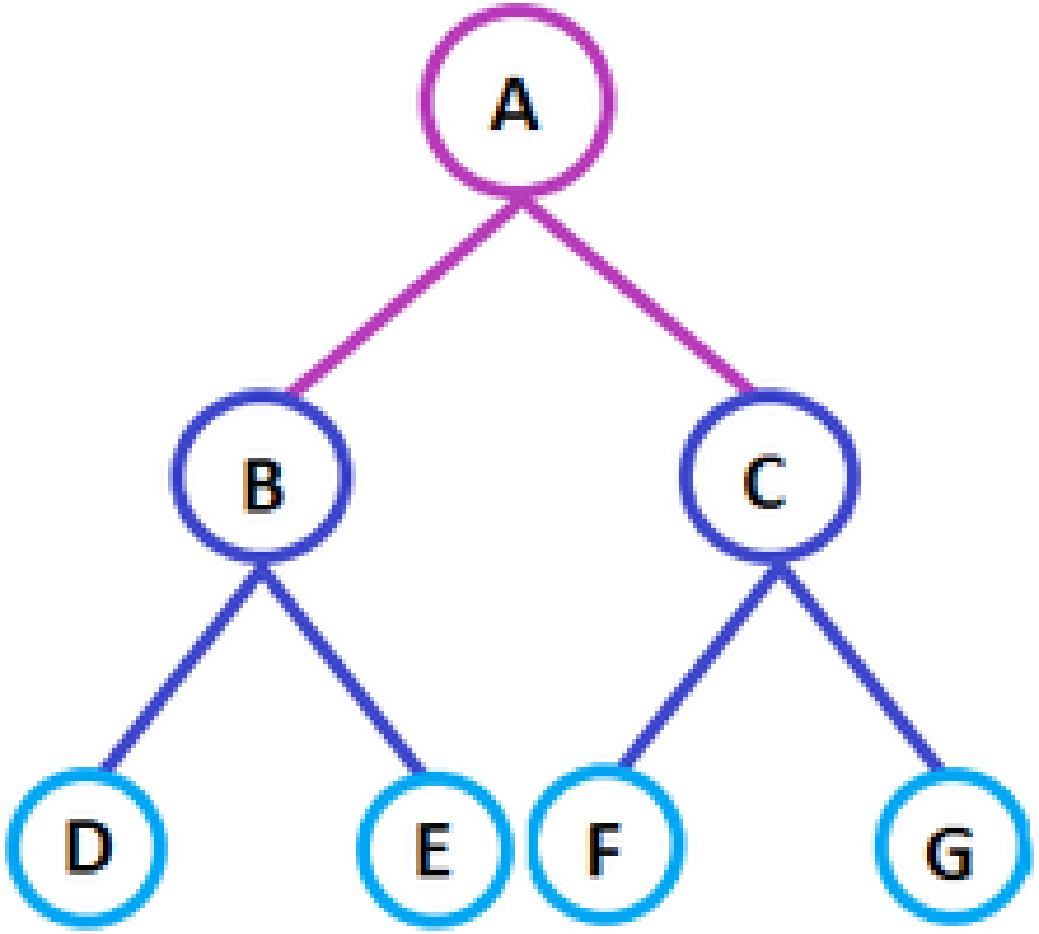
Construct Binary Search Tree

- 10,2,7,1,20,15,11,22,28

Construct Binary Search Tree

- 50,25,12,60,80,55,50,4,6,28

Tree Traversal



Construct tree

- Inorder sequence: D B E A F C
- Preorder sequence: A B D E C F

Construct tree

- preorder [3,9,20,15,7]
- inorder = [9,3,15,20,7]

Construct tree

- Inorder Traversal : { 4, 2, 1, 7, 5, 8, 3, 6 }
- Preorder Traversal: { 1, 2, 4, 3, 5, 7, 8, 6 }

Construct tree

- in[] = {4, 8, 2, 5, 1, 6, 3, 7}
- post[] = {8, 4, 5, 2, 6, 7, 3, 1}

Construct tree

- inorder = [9,3,15,20,7]
- postorder = [9,15,7,20,3]

Construct tree

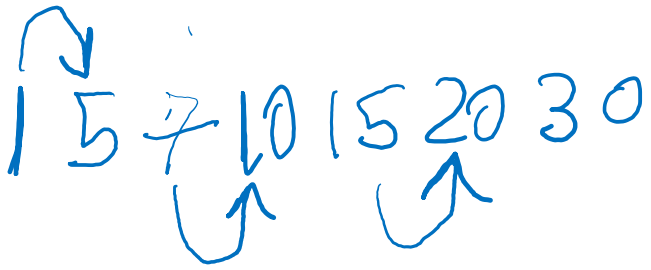
- Inorder Traversal : { 4, 2, 1, 7, 5, 8, 3, 6 }
  Postorder Traversal : { 4, 2, 7, 8, 5, 6, 3, 1 }

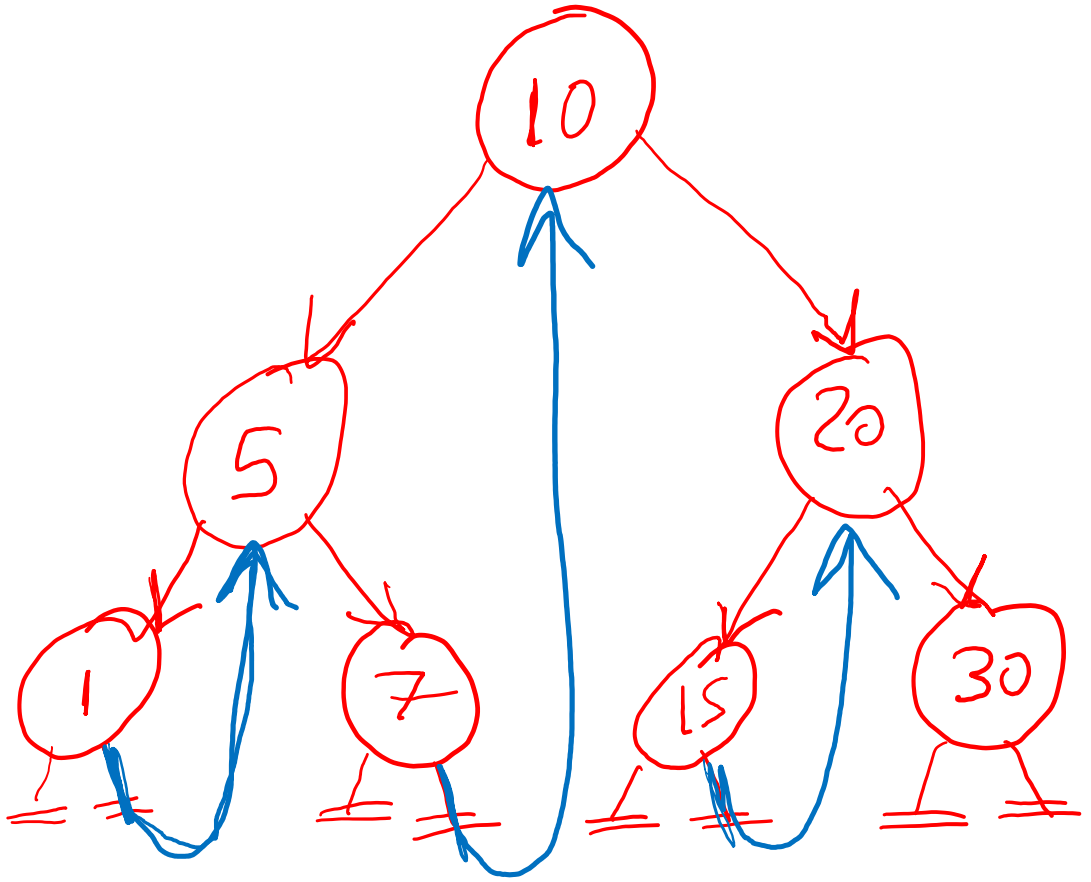
Threaded Tree

- Inorder traversal of a Binary tree can either be done using recursion or with the use of a auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).

Threaded Tree

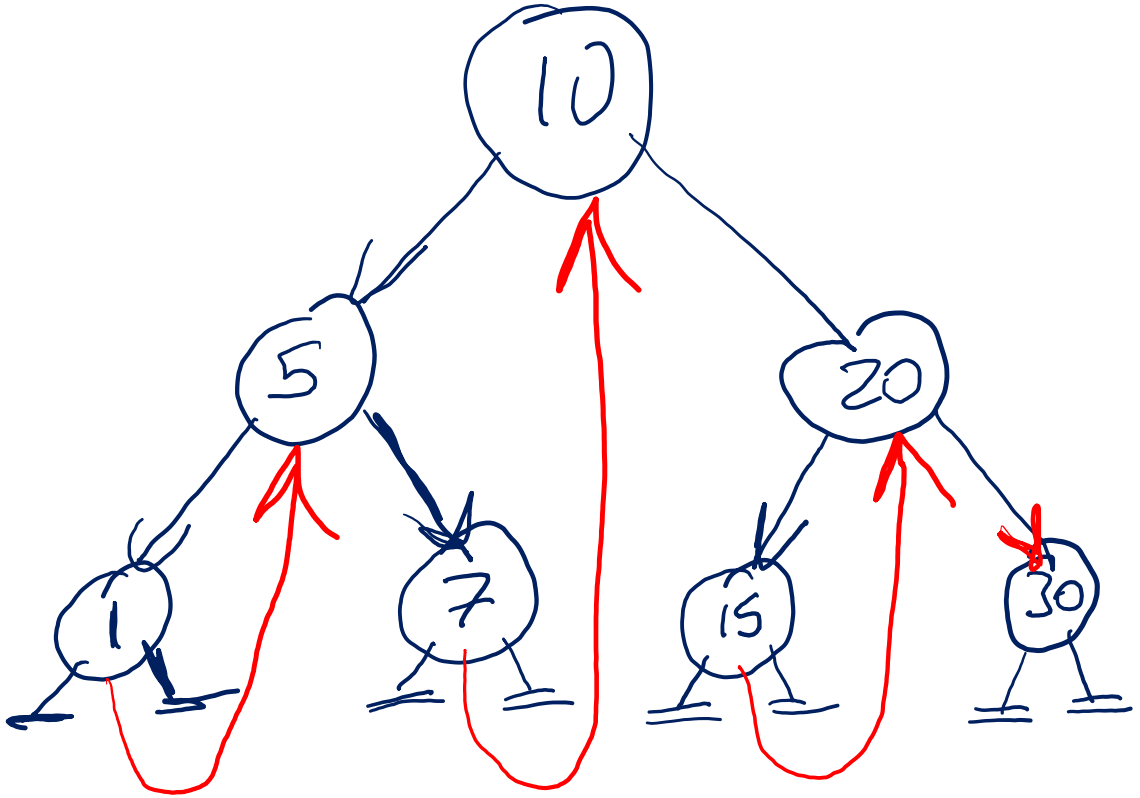
- Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)





Threaded Tree

- Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.



AVL TREE

⇒ insertion in BST form 1 element at a time

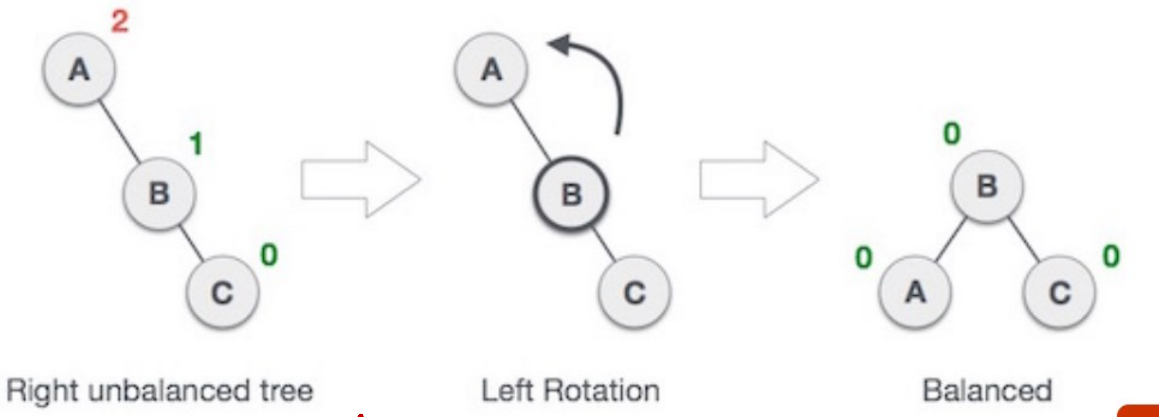
⇒ after every insert cal B.F from bottom to top

⇒ stop when B.F goes Beyond +1 to -1 & search path from unbalanced to newly inserted (3 node path)

⇒ check case-1/2/3/4 & apply Rotation

Rotations

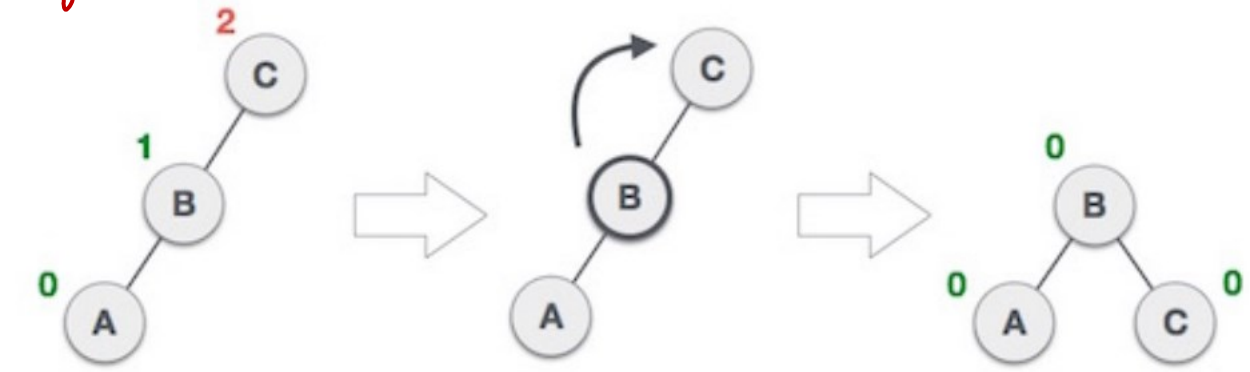
single



Right unbalanced tree | Left Rotation | Balanced

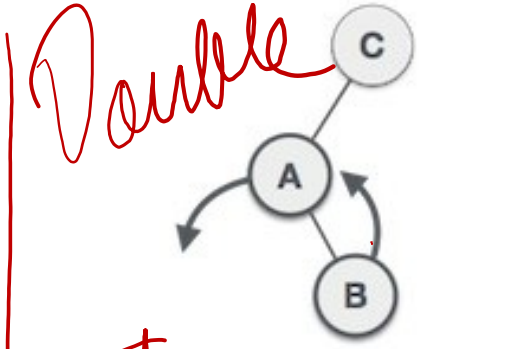
Right of Right

Left of Left

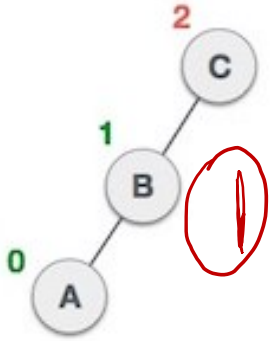


Left unbalanced Tree | Right Rotation | Balanced Tree

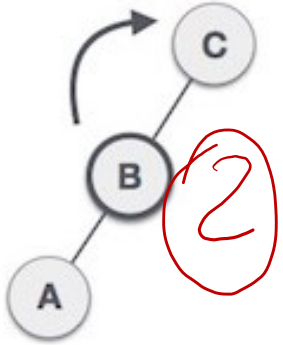
Double



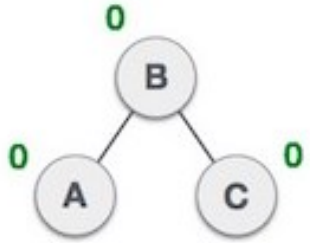
Right of Left

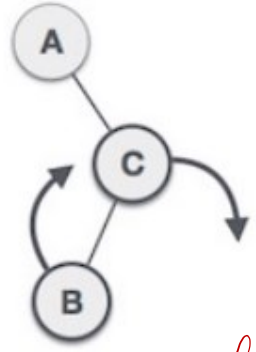


1



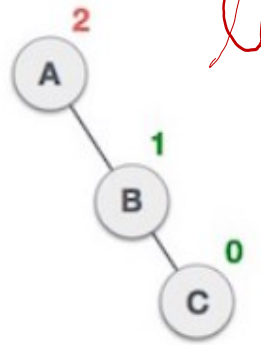
2



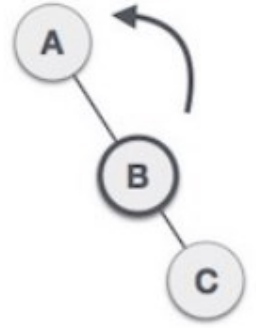


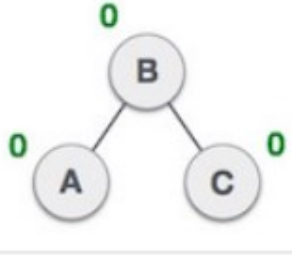
Left of Right

1



2





B-Tree

- B-Tree is a self-balancing search tree.
- Single node consist multiple elements this reduces number of read write and memory swaps.
- the B-Tree node size is kept equal to the disk block size
- All leaves are at the same level.
- B-Tree is defined by the term minimum degree ‘M‘.
- M is odd.
- On Order M:
  - Maximum M-1 elements at a node
  - Minimum M/2 elements at a node
  - Split when M elements reached
- All keys of a node are sorted in increasing order.
- B-Tree grows and shrinks from the root which is unlike Binary Search Tree.
- Like other balanced Binary Search Trees, the time complexity to search, insert and delete is O(log n).
- Insertion of a Node in B-Tree happens only at Leaf Node in order and BST form.

Multiway Search Tree

- 1 node with n elements
- m-order → min m/2
  → max m-1
  → split m
  must be odd
- Data kept in order at node
- insertion done only in leaf creates parent & grandparent

3
!
2
3

B+ Tree

→ Enhanced B tree for sequential access

– Changes: ① all data kept in leaf only

② at split only ref is shifted

③ all leaf nodes are connected via Bidirectional link.

BST IMPLEMENTATION

- Static Array
- Dynamic Linked List
- Dynamic Tree node