Topic: Assignment - 2

Page No. 1
Date

MCA 1st Semester (ADS)

Submitted to - [redacted]

Q. AVL tree ? Write their application & implementation.

→ AVL tree is a Binary Search Tree (BST) and a self-Balanced tree.

So it fulfils all BST conditions -

- The left child contains smaller value than root and the right child contains greater value than root.

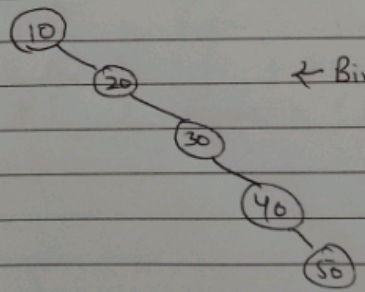
Along with this, it keeps itself height balanced. The height difference b/w left and right subtree is never more than 1

BF = -1, 0 or +1 for every node.

→ ~~How AVL solve this~~ and Why AVL is Needed ?

Normal BST becomes unbalanced when we insert elements in sorted order, which makes the tree skewed (like a linked list).
This increases the height too much and searching becomes slow (O(n)).

To solve this problem, AVL Tree was introduced.



← Binary Tree Problem.

O(n)

Teacher's Signature

Topic.....

→ How AVL solve this -
AVL always keep the tree balanced.
It maintains balanced factor = -1, 0, +1
Height stays O(log n)
All operation (search/insert/delete) remain fast.

✓ Balanced Factor.

Balanced Factor = height of left subtree - height of right subtree

→ For an AVL tree it must be -1, 0 or +1

→ 0 means perfectly balanced +1 means slightly left heavy, -1 means slightly right heavy.

→ If BF becomes ±2 tree becomes unbalanced and rotation are applied.

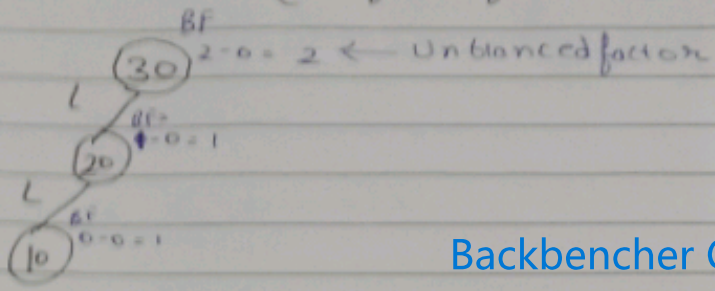
✓ Rotations

AVL rotations are used to restore balanced when the balanced factor becomes =±2

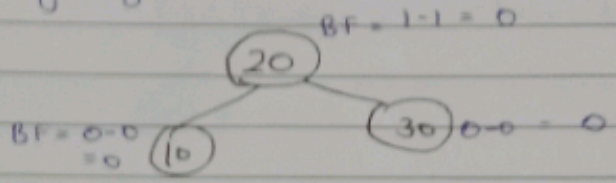
LL → Right Rotation
RR → left Rotation
LR → left + Right Rotation
RL → Right + left Rotation.

Teacher's Signature

1) LL Rotation (left-left case)



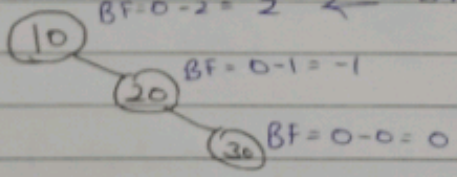
2) Apply right side Rotation on (30) Unbalanced node



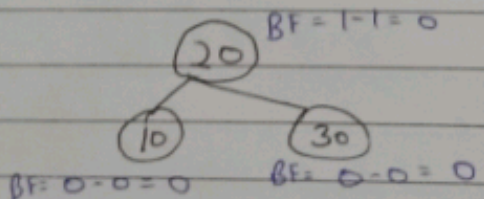
One rotation applied.

Now its Balanced factor Tree.

2) RR Rotation (Right-Right case)



Apply left Rotation on (10) Unbalanced node.



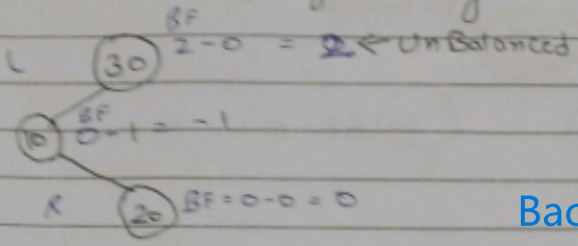
One rotation applied.

Now its Balanced factor Tree.

Topic

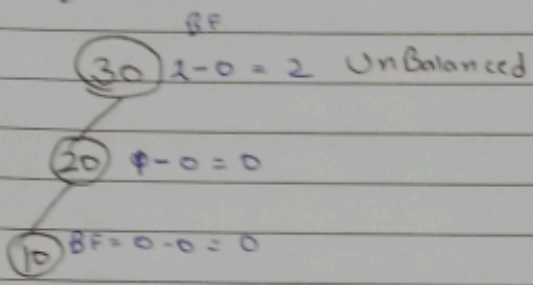
Page No. 4
Date

(3) LR Rotation (left - Right Case)

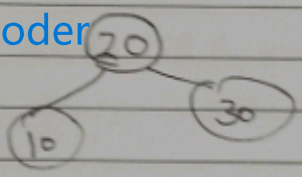


2 rotation are applied.

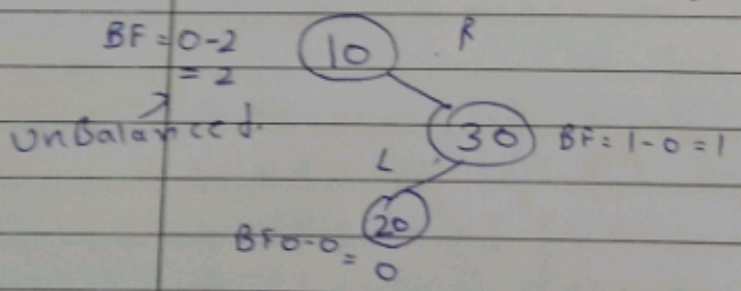
Step1: left Rotation on 10



Step2: Right Rotation on 30

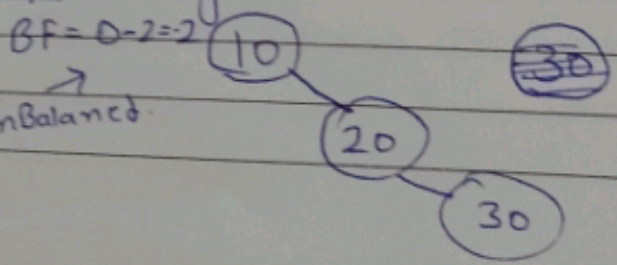


(4) RL Rotation (Right- left Case)



2 rotation are applied
1) Right
2) left

Step1 Right Rotation on 30.



Teacher's Signature

⇒ Implementation of AVL Tree.

The implementation of an AVL tree follows these steps -

1] Insert Node Using BST Rules
First, every new node is inserted just like a Binary Search Tree (BST)
left subtree contains smaller values and right subtree contains larger value.

2] Calculate Balance Factor.
After every insertion we calculate:

Balance Factor (BF) = height(left subtree) - height(right subtree).

- For an AVL tree, BF must be -1, 0 or +1
- If BF is outside this range, the tree become unblanced.

3] Identify the Type of Rotation Needed.
when a node becomes unbalanced, its case is identified. applied Rotaions.
1) LL case (left - left)
2) RR case (left Rotation)
3) LR case (left Rotation + Right Rotation)
4) RL case (Right Rotation + left Rotation)

These rotations restore balanced to the tree.

5] Update height.

After rotation update the height of each affected node.

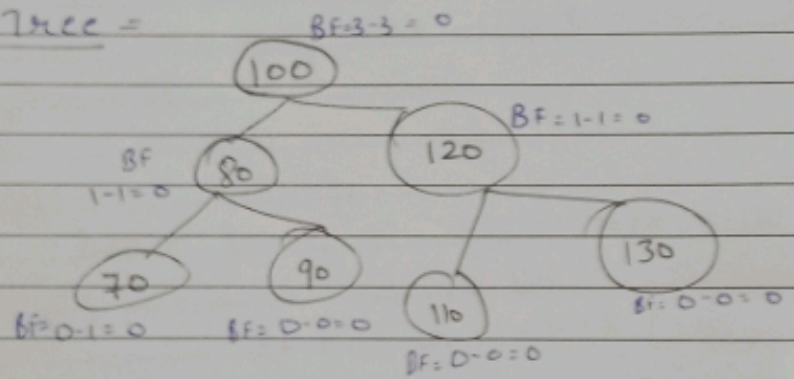
6] Maintain the AVL Property.

After every insertion and deletion, the tree must remain

a] Height balanced

b] Search, insert, delete operation remain O(log n).

AVL Tree =



⇒ AVL Deletion

AVL Tree follow the BST deletion procedure first. After deleting the node like BST, AVL checks the balanced factor of each ancestor and applies rotations if required.

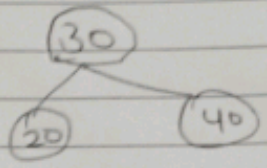
Teacher's Signature

Case1 : Node is a leaf (No children)

The node you want to delete has no left child and no right child.

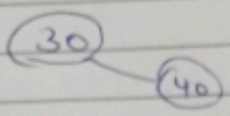
Example:

Delete 20 from this tree



20 is a leaf - simply remove it

After deletion.

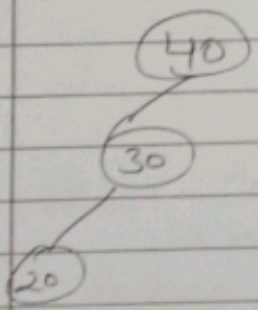


Case2. Node has one child

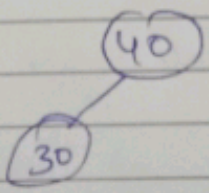
The node has only one child → either left child OR right child.

Example:

Delete 30. ↳ Replace node it's child



30 has one child (20)

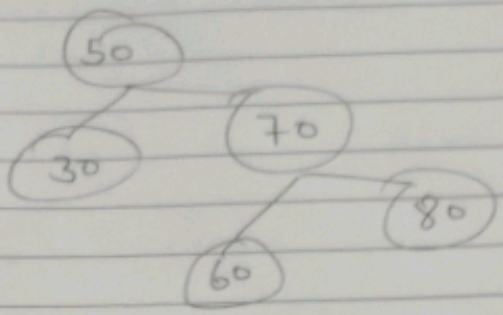


Case3 Node Has Two Children (Important Case)

The node you want to delete has both left child and right child.

Topic

Example:
Delete 50



50 has two children.

→ What to do?
When a node has two children, we cannot directly delete it.

Step1 Find the inorder successor.

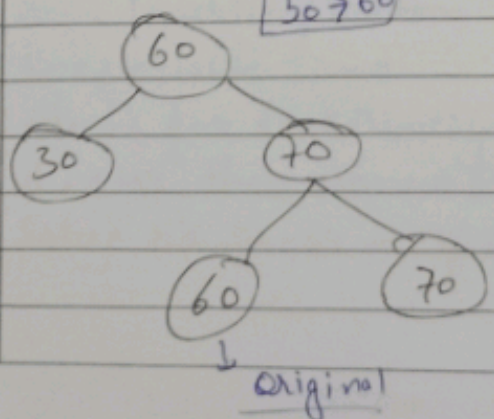
Inorder successor = Smallest value in the right subtree.

– In this case successor is 60 (because 60 is the smallest in right subtree).

Step2. Replace node value with successor value.

Replace 50 with 60.

50 → 60

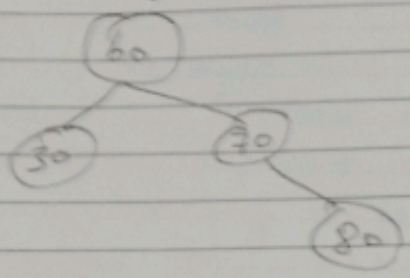


↓ original

We temporarily keep both 60s.

Teacher's Signature

Step3 :- Now delete the successor node. Successor node (60) will always be in Case 1 or Case 2, so deletion becomes easy.



After deleting the extra 60:

=> AVL Tree Applications:

1) Databases - To find and store data quickly.
2) File System - To organize files and folders fast.
3) Memory Management - To manage free and used memory.
4) Networking - Routing - to find path quickly.
5) Compilers - To manage variables and function.
6) Dictionaries / Spellcheck - To search words fast.
7) Real-time System - Because is always fast O(logn)

=> Advantages:

1) Always Balanced
2) Fast Searching
3) Fast Insertion / Deletion
4) Predictable Time
5) Good for Real-Time System
6) Better then Normal BST
7) Solve the problem of BST

Disadvantages:

1) Complex to Implement
2) More Memory.
3) Slower Insertion / Deletion
4) Overhead for Small Tree.

Teacher's Signature

Topic.....

Page No. 10
Date.....

02 Binary Tree

A Binary Search Tree is a special type of binary tree in which:

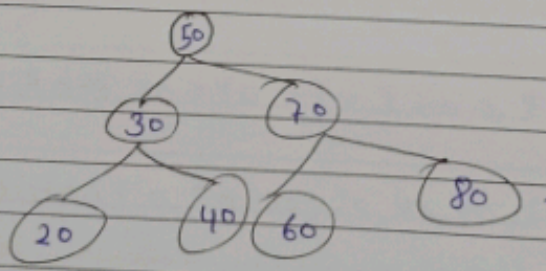
- The left child of a node contains smaller value than the node.
- The right child of a node contains greater value than the node
- NO duplicate values are allowed
- Searching is fast because the tree is arranged in sorted order.

BST Node Placement Rule

1. If new data is smaller than the node → go to LEFT subtree.
2. If new data is greather than the node → go to RIGHT subtree.
3. Leftmost leaf node = smallest value in the BST
4. Rightmost leaf node = largest value in the BST

Example:

insert - 50, 30, 70, 20, 40, 60, 80



smallest value → in BST

← largest value in BST

Teacher's Signature

Page No. 11
Date

Topic

BST Traversals

Traversal means visiting all nodes of the tree in specific order.

There are three main traversals -

1] In Order Traversal (left → Root → Right)

Rule:
Visit left subtree → visit root → visit right subtree

Output is always in sorted order for BST.

Example using the BST above:

Steps.

20 → 30 → 40 → 50 → 60 → 70 → 80

Final Output - 20, 30, 40, 50, 60, 70

2] Pre Order Traversal (Root → left → Right)

Rule: Visit root → visit left subtree → visit right subtree

Steps

50 → 30 → 20 → 40 → 70 → 60 → 80

Final output - 50, 30, 20, 40, 70, 60, 80.

3] Post order Traversal (left → right → Root)

Rule: vist left subtree → vist right subtree → visit root atlast

Exp. 20 → 40 → 30 → 60 → 80 → 70 → 50

final output - 20, 40, 30, 60, 80, 70, 50

Teacher's Signature

Topic

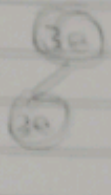
Deletion in BST

In a Binary Search Tree, deletion depends on how many children the node has

Case 1: Node has No child (leaf node)

Just remove the node

It does not affect the rest of the tree



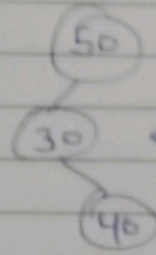
Delete 30

← Easy to remove

Case 2: Node has One Child

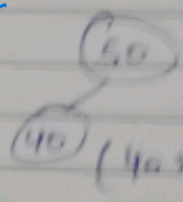
1. Remove the node
2. Connects its child directly to its parent
3. So the child takes the deleted node's place

Delete 30 (it has one child 40)



← remove

After



(40 replaces 30)

Case 3: Node has Two children.

Find its inorder successor

(smallest node in its right subtree)

Copy the successor's value into the node

Then delete the successor node (which will be Case 1 or Case 2)

Teacher's Signature

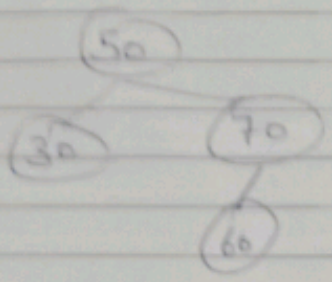
Topic

Example

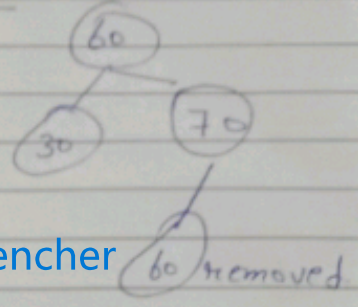
Delete 50

Right subtree has 60, 70, 80

Smallest is 60



After



→ Advantages

1. Fast Searching
2. Easy to Insert & Delete
3. Sorted Data
4. Uses less Memory
5. Good for large Data
6. Easy to implement

Disadvantages

1) Can become Unbalanced - if data sorted
2) Slow operation in worst case
3) No guaranteed performance
4) Not efficient for real time system
5) Duplicate handling is tricky.

=> Application

1) Databases
2) File System
3) Memory Management
5) Dictionary/Spellcheck
7) Compilers
6) Real-time System.

- Time Complexity

Operation	Average Case	Worst Case
Search	O(log n)	O(n)
Insertion	O(log n)	O(n)
Deletion	O(log n)	O(n)

Tree is skewed (linked list) → height = n → slow operation.

Teacher's Signature

Topic

Page No.
Date

Q3 Splay Tree

=> Splay Tree

A Splay Tree is a type of Self-adjusting Binary Search Tree in which the most recently accessed node is moved to the root using rotations.

This process is called splaying.

- It follows all BST rules
- Recently used items become quicker to access
- No strict balancing like AVL, but averages stay fast

Why Splay Tree?

If you search something repeatedly, it becomes faster because the element moves to the top.

Operation in Splay Tree

Search - After Searching, the node is moved to the root
Insert - Insert normally, then Splay (move new node to root).
Delete - After deletion, splay the parent node.

[Splay Tree = BST + Rotations]

Teacher's Signature

Topic

⇒ Splay Tree Rotations

Splay Tree uses three types of rotations during Splaying

1] Zig Rotations 2] Zig-Zig Rotation 3] Zig-Zag Rotation.

Splaying happens in insertion, Search, and deletion Every time you access a node, you bring it to the root using rotations.

⇒1] ZIG Rotation (Single Rotation)

Used when the node x has no grandparent (x's parent is root).

Two Case of ZIG Rotation.

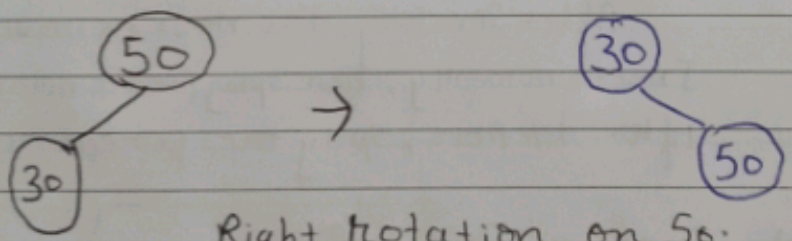
Case A: Zig-Right (Right Rotation)

Used when: x is left child of the root

Example:

Insert: 50 → 30

Then splay (30)



Right rotation on 50.

⇒ Case B: Zig-left (left Rotation)

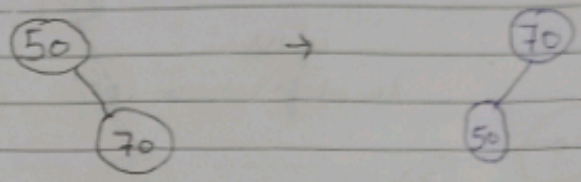
Used when: x is right child of the root

Teacher's Signature

Topic

Example
Insert 50 → 70

Then splay(70)



2] Zig-Zig Rotation (Double Rotation on Same Side)
This happen when
* x is left child of parent or right child of parent
* parent is left child of grandparent or parent right right grandparent.

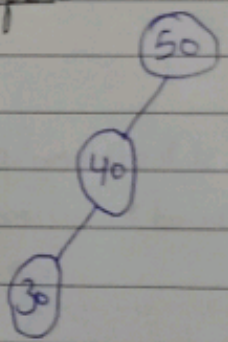
(left-left case → two right rotations)

(right-right case → two left rotations)

→ Example (left-left Zig-Zig)

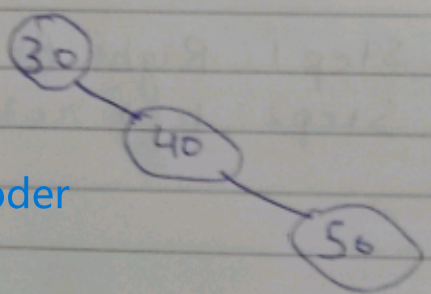
Insert 50 → 40 → 30
Splay(30)

Before



Step1 : Right Rotation on 50
Step2 : Right Rotation on 40

After :

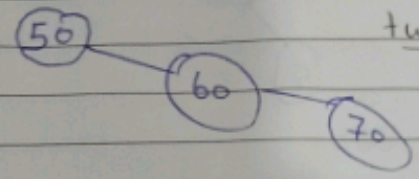


Teacher's Signature

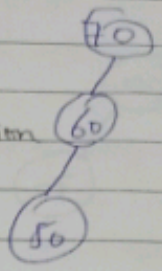
Example: Right-Right (Zig-Zig)

Insert 50→60→70
Splay(70)

Before:



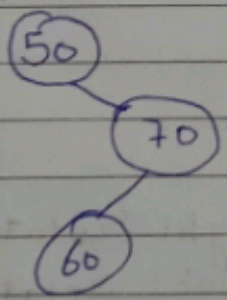
After: two left rotation



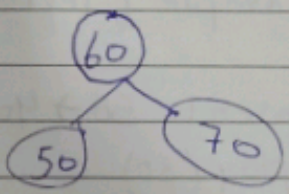
3) ZIG-ZAG Rotation (Double Rotation Opposite Side)
Child and parent opposite Side
i) x is left child of parent
ii) parent is right child of grandparent
(OR vice-versa)

Example:
Insert: 50→70→60
Splay(60)

Before:



After:



Step 1: Right Rotation on 70 (first rotation perform on parent)
Step 2: left rotation on 50 (Second rotation perform on grandparent)

Teacher's Signature

Topic

→ Operations

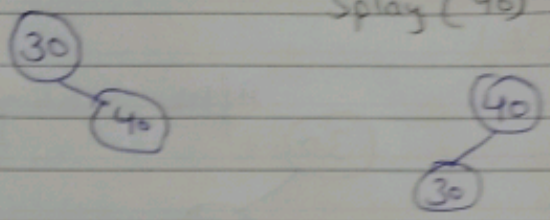
1] Insertion

a) Insert the element like a normal BST
b) Splay the inserted node to the root

Example (Insert 30, 40,)

a) Insert 30 → root
Insert 40

Splay (40)



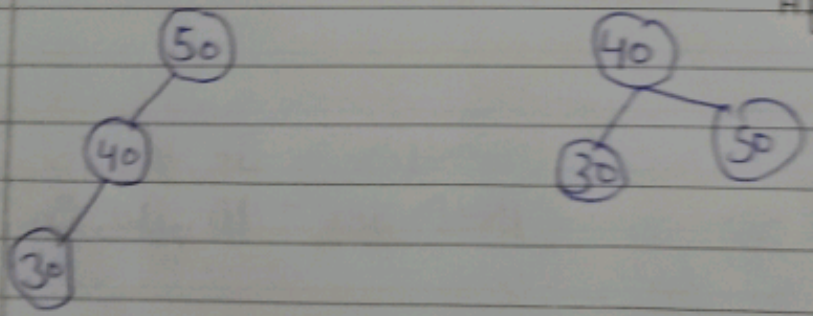
2] Search

a) Search like normal BST
b) If the element is found → splay it to the root
c) If not found → splay the last accessed node.

Example

Search 40 in

After Splaying 40



Teacher's Signature

Topic

3) Deletion:

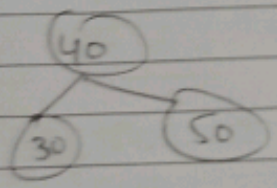
1) First splay the node to be deleted → it becomes root
2) Delete the root

3) Two subtrees remain
   a) left subtree (L)
   b) Right subtree (R)

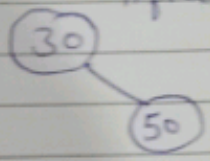
4) Splay the maximum node of L → make it the node
5) Attach R as the right child of this node.

Example (Delete 40)

Before deletion



After deleting 40:

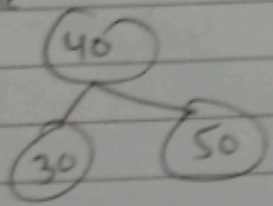


4) Traversals

Same as normal BST

a) Inorder - gives sorted output
   Preorder -
   Postorder -

Example:



InOrder - 30, 40, 50
PreOrder - 40, 30, 50

Teacher's Signature

Topic.....

Page No. 11
Date

→ Application

1] Caching - Because recently used items automatically move top
2] Memory Management - Help the system quickly free or used memory
3] Network Routing Table
4] Garbage collection
5] File systems
6] ~~Used in Network routing tables~~
6] Used in string/text editors - for fast insert, delete and search of characters.

→ Advantages.

1] Fast for frequently accessed elements
2] No extra balance information needed
3] Good average performance
4] Simple to implement.

→ Disadvantages

1] Wrost-case time is O(n)
The tree can become skewed and very tall.

2] Too many rotations

3] No guarnteed balanced height.

4] Not good for Uniform access.

Teacher's Signature

Topic.....

Page No.....
Date.....

B-Tree

B-tree is a self-balanced Search Tree in which every node contains multiple keys and has more than two children.

A B-Tree is a balanced tree where -

- Every node can have more than 1 key
- All leaves are at at same level
- Keeps data sorted for fast search, insert and delete.

Used in: databases, file system, large data storage.

- B-Tree of (order/degree) has the following properties-

let m = Order of B-Tree

1) All leaf nodes must be at same level
2) Maximum keys in a node = $m-1$
3) Maximum children in node = m
4) Minimum keys in a non-root node = $\left\lceil \frac{m}{2} - 1 \right\rceil$
5) Minimum children in a non-root node $\left\lceil \frac{m}{2} \right\rceil$
6) All the key value in a node in Ascending Order.

Teacher's Signature