DS UNIT 5

[4-marks]

1. Define tree data structure. Give its terminology.

A **tree** is a **non-linear data structure** used to store data in a hierarchical form. It consists of nodes connected by edges, and there is one special node called the root.

The **root** is the topmost node of the tree. It is the starting point from where all other nodes are connected.

A **node** is an individual element of a tree that stores data. Each node may have one or more child nodes connected to it.

An **edge** is a connection between two nodes in a tree. It represents the relationship between a parent and a child.

A **parent node** is a node that has one or more child nodes. The child nodes directly come under the parent node.

A **child node** is a node that descends from another node. Every node except the root has exactly one parent.

Leaf nodes are nodes that do not have any children. They are also called terminal nodes.

Level of a node shows its position in the tree starting from root. The root node is at level 0.

Height of a tree is the maximum number of edges from the root to the deepest leaf. It shows how tall the tree is.

2. Define binary tree. List its types and explain them with example.

A binary tree is a tree data structure in which each node can have at most two children. These children are called the left child and the right child.

In a **Full Binary Tree**, every node has either 0 or 2 children. No node is allowed to have only one child.
Example: A tree where each internal node has exactly two children.

A **Complete Binary Tree** is a binary tree where all levels are completely filled except possibly the last level. The last level is filled from left to right.

A Perfect Binary Tree is a binary tree in which all internal nodes have two children and all leaf nodes are at the same level. This type of tree is completely balanced.

A **Skewed Binary Tree** is a tree where all nodes have only one child. It can be either left-skewed or right-skewed.

A **Balanced Binary Tree** is a tree where the height difference between left and right subtrees is minimal. This helps in improving performance of operations.

3. **What is Binary Search Tree (BST)?**

A Binary Search Tree (BST) is a special type of binary tree used for efficient searching. It follows a specific ordering rule for storing elements.

In a BST, the left subtree contains only values smaller than the root node. This rule applies to all nodes in the tree.

The right subtree contains only values greater than the root node. This helps in fast searching and sorting.

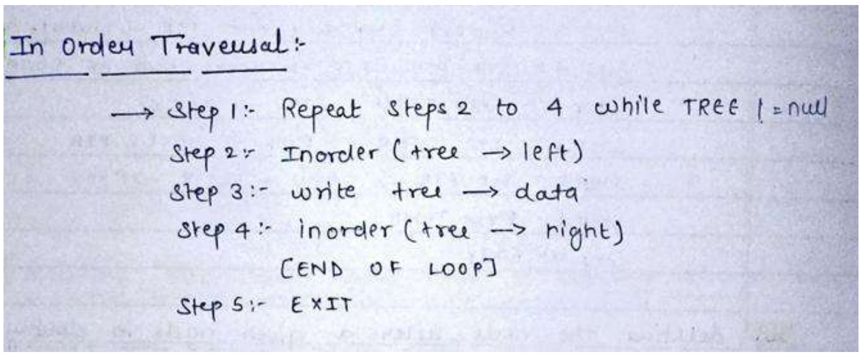
Each subtree of a BST is also a binary search tree. This recursive property makes BST powerful and easy to use.

Searching in a BST is faster compared to a normal binary tree. The time complexity is generally O(log n) when the tree is balanced.

BSTs are commonly used in searching, sorting, and maintaining ordered data. Examples include dictionary implementation and database indexing.

4. **Write the algorithm for inorder traversal.**

Inorder traversal is a method of visiting nodes in a binary tree. In this traversal, nodes are visited in the order Left subtree → Root → Right subtree.



In Order Traversal :-

```
→ Step 1 :- Repeat steps 2 to 4 while TREE != null
  Step 2 :- Inorder (tree → left)
  Step 3 :- write tree → data
  Step 4 :- inorder (tree → right)
            [END OF LOOP]
  Step 5 :- EXIT
```

5. **Differentiate between full binary tree and complete binary tree.**

Full Binary Tree

- A full binary tree is a binary tree where each node has either 0 or 2 children. No node is allowed to have exactly one child.
- Leaf nodes do not have any children. Internal nodes always have two children.
- The structure of a full binary tree is strict. Missing children are not allowed.

Complete Binary Tree

- A complete binary tree is a binary tree where all levels are completely filled except possibly the last level. The last level is filled from left to right.
- Nodes in the last level may not have both children. However, gaps are not allowed in between nodes.
- This type of tree is commonly used in heap data structures. It ensures efficient memory usage.

6. Define AVL tree. Why is it called height-balanced tree?

An AVL tree is a self-balancing binary search tree. It automatically maintains balance after insertion and deletion operations.

In an AVL tree, the difference between the heights of the left and right subtrees of any node is at most 1. This difference is called the balance factor.

The balance factor can have values –1, 0, or +1 only. If the balance factor goes outside this range, rotations are performed.

It is called a height-balanced tree because it keeps the height of the tree as small as possible. This ensures faster searching, insertion, and deletion.

Due to height balancing, AVL trees provide O(log n) time complexity. This makes them more efficient than normal BSTs.

7. What is a threaded binary tree?

A **threaded binary tree** is a special type of binary tree. It uses empty left or right pointers to store links called threads.

In a normal binary tree, many pointers are NULL. A threaded binary tree replaces these NULL pointers with pointers to inorder predecessor or successor.

Threaded binary trees allow traversal without using a stack or recursion. This reduces memory usage and improves performance.

There are two types of threaded binary trees: **single-threaded** and **double-threaded**. Single-threaded uses one thread, while double-threaded uses both left and right threads.

Threaded binary trees are mainly used to make inorder traversal faster and simpler. They are useful in memory-constrained systems.

[5 – marks]

1. **Explain representation of binary tree using array and linked list.**

Array Representation

- In array representation, the binary tree is stored in a sequential manner. The root node is stored at the first index of the array.
- If a node is stored at index i, its left child is stored at index 2i and right child at index 2i + 1. This rule helps in easily locating child nodes.
- Array representation is simple and easy to implement. It works best when the tree is complete or nearly complete.
- The main disadvantage is memory wastage. If the tree is sparse, many array locations remain unused.

Linked List Representation

- In linked list representation, each node contains data, left pointer, and right pointer. Pointers store addresses of child nodes.
- Memory is allocated dynamically for each node. This avoids wastage of memory.
- This method is suitable for all types of binary trees. It is more flexible than array representation.

2. **Explain preorder, inorder, and postorder traversals with example.**

Tree traversal means visiting all nodes of a binary tree exactly once. Traversals help in accessing and processing tree elements.

```
      A
     / \
    B   C
   / \   \
  D   E   F
```

Preorder Traversal (Root → Left → Right)

- In preorder traversal, the root node is visited first. Then the left subtree and right subtree are visited.
- It is mainly used to copy trees or create prefix expressions.
- Example order: A B D E C F

Inorder Traversal (Left → Root → Right)

- In inorder traversal, the left subtree is visited first. Then the root and finally the right subtree.
- In a Binary Search Tree, inorder traversal gives elements in sorted order.
- Example order: D B E A F C

Postorder Traversal (Left → Right → Root)

- In postorder traversal, both subtrees are visited before the root node.
- It is useful for deleting a tree and postfix expressions.
- Example order: D E B F C A

3. **Explain insertion and deletion operations in a Binary Search Tree.**

Insertion in BST

- Insertion starts from the root node. The new value is compared with the root value.
- If the new value is smaller, it is placed in the left subtree. If it is greater, it is placed in the right subtree.
- This comparison continues until an empty position is found. The new node is inserted at that position.
- Insertion maintains the BST property. It ensures faster searching later.

Deletion in BST

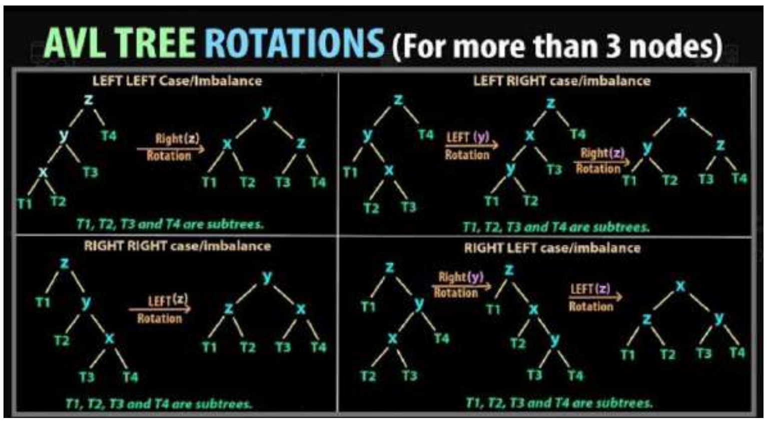
- Deletion is more complex than insertion. It depends on the number of children of the node.
- If the node has no children, it is simply removed.
- If the node has one child, the child replaces the deleted node.
- If the node has two children, the inorder successor or predecessor replaces the node.
- After deletion, BST properties are preserved.

4. **Explain AVL tree rotations with suitable examples. (just given image for reference)**

An **AVL tree** is a self-balancing Binary Search Tree. It maintains balance by ensuring that the **height difference (balance factor)** between the left and right subtrees of any node is at most **1**.

When insertion or deletion causes the balance factor to become **–2 or +2**, the tree becomes unbalanced. To restore balance, **rotations** are performed.

Rotations are local rearrangements of nodes. They do not change the inorder sequence, so the BST property remains intact.


AVL TREE ROTATIONS (For more than 3 nodes)
LEFT LEFT Case/Imbalance
Right(z) Rotation
T1, T2, T3 and T4 are subtrees.
LEFT RIGHT case/imbalance
LEFT (y) Rotation
Right(z) Rotation
T1, T2, T3 and T4 are subtrees.
RIGHT RIGHT case/imbalance
LEFT(z) Rotation
T1, T2, T3 and T4 are subtrees.
RIGHT LEFT case/imbalance
Right(y) Rotation
LEFT(z) Rotation
T1, T2, T3 and T4 are subtrees.

1. LL Rotation (Left-Left Case)

- This case occurs when a node becomes unbalanced due to insertion in the **left subtree of the left child**.
- A **single right rotation** is performed on the unbalanced node.
- Example: Insert 30, 20, 10 → node 30 becomes unbalanced, so right rotation fixes it.

2. RR Rotation (Right-Right Case)

- This case occurs when insertion happens in the **right subtree of the right child**.
- A **single left rotation** is performed on the unbalanced node.
- Example: Insert 10, 20, 30 → node 10 becomes unbalanced, so left rotation balances the tree.

3. LR Rotation (Left-Right Case)

- This occurs when insertion is in the **right subtree of the left child**.
- It requires **two rotations**: first a left rotation on the left child, then a right rotation on the unbalanced node.
- Example: Insert 30, 10, 20.

4. RL Rotation (Right-Left Case)

- This occurs when insertion is in the **left subtree of the right child**.
- It also needs **two rotations**: first a right rotation on the right child, then a left rotation on the unbalanced node.
- Example: Insert 10, 30, 20.

5. **Explain properties and applications of B-Tree.**

Properties of B-Tree

- A B-Tree is a self-balancing multi-way search tree. It is mainly used to store large amounts of data efficiently.
- In a B-Tree of order m, each node can have at most m children. It can have at least ⌈m/2⌉ children, except the root.
- All leaf nodes are at the same level. This makes the tree perfectly balanced.
- A node can store multiple keys in sorted order. This reduces the height of the tree.
- The root node has a minimum of two children unless it is a leaf. This rule ensures stability of the tree.
- Searching, insertion, and deletion operations take O(log n) time. Performance remains efficient even with large data.

Applications of B-Tree

- B-Trees are widely used in database systems. They help in fast searching and indexing of records.
- They are used in file systems to store directory structures. Examples include NTFS and ext file systems.
- B-Trees reduce disk access because they store more keys in one node. This improves performance in secondary storage.
- They **are ideal for systems where data is frequently inserted and deleted.**

6. **Differentiate between BST and AVL tree.**

Binary Search Tree (BST) : A BST is a binary tree where the left subtree contains smaller values and the right subtree contains larger values. This rule applies to every node.

BST does not automatically balance itself. The tree may become skewed.

In the worst case, time complexity becomes O(n). This happens when the tree looks like a linked list.

BST is simpler to implement. It requires fewer operations.

It is suitable when data is mostly random and balancing is not critical.

AVL Tree : An AVL tree is a self-balancing binary search tree. It maintains balance using rotations.

The height difference between left and right subtrees is at most 1. This ensures the tree remains balanced.

Time complexity for search, insertion, and deletion is always O(log n).

AVL trees require extra rotations, making insertion and deletion slower than BST.

They are used where fast searching is more important than insertion speed.

7. **Explain height and degree of a tree with example.**

Example:

```
      A
     / \
    B   C
   /
  D
```

Height of a Tree

- The height of a tree is the maximum number of edges from the root node to the deepest leaf node. It shows how tall the tree is.
- Height is used to measure the efficiency of tree operations like searching and insertion. A smaller height usually means faster operations.
- The height of a single-node tree (only root) is 0. If we count nodes instead of edges, height may be counted as 1 depending on definition.
- Longest path from root A → B → D has 2 edges.
- So, height of the tree = 2.

Degree of a Tree

- The degree of a node is the number of children that node has. It tells how many direct branches come out from a node.
- The degree of a tree is the maximum degree of any node in that tree.
- Leaf nodes always have degree 0 because they have no children.

Degree of A = 2

Degree of B = 1

Degree of C = 0

Degree of D = 0

Degree of the tree = 2

8. **Write Step to convert general tree to binary tree. Construct binary tree from given general tree.**

A general tree can have any number of children. To convert it into a binary tree, we use the Left Child–Right Sibling (LCRS) method.

Steps to Convert General Tree to Binary Tree

- Step 1: Make the leftmost child of each node as its left child in the binary tree.
- Step 2: Connect the remaining children of the same parent using right pointers. These children become right siblings.
- Step 3: Remove all other parent-child links except the leftmost one. This ensures each node has at most two children.
- Step 4: Repeat the same process for all nodes in the tree. The final structure is a binary tree.

Example :

General Tree

```
    A
 /  |  \
 B  C  D
   / \
  E   F
```

Binary Tree

```
   A
 /
 B
  \
   C
  / \
 E   D
```

[10 – marks]

1. **Explain binary tree in detail. Describe its types, representation, and traversal techniques with algorithms.**

Binary Tree – Definition

- A binary tree is a non-linear data structure in which each node can have at most two children. These children are called the left child and the right child.
- It is used to represent hierarchical data such as expressions, file systems, and search structures. Binary trees help in efficient searching, insertion, and deletion of data.

  *

Types of Binary Tree

- A Full Binary Tree is a tree in which every node has either 0 or 2 children. No node is allowed to have exactly one child.
- A Complete Binary Tree is a tree where all levels are completely filled except possibly the last level. The last level is filled from left to right.
- A Perfect Binary Tree is a tree where all internal nodes have two children and all leaf nodes are at the same level.
- A Skewed Binary Tree is a tree where every node has only one child. It can be left-skewed or right-skewed.
- A Balanced Binary Tree is a tree where the height difference between left and right subtrees is minimal. This improves performance.

Representation of Binary Tree

Array Representation

- In array representation, the root is stored at index 1. If a node is at index i, its left child is at 2i and right child at 2i + 1.
- This method is simple and efficient for complete binary trees. However, it causes memory wastage for sparse trees.

Linked List Representation

- In linked list representation, each node contains data, left pointer, and right pointer.
- Memory is allocated dynamically, so there is no wastage. This method is flexible and commonly used.

Traversal Techniques of Binary Tree

Tree traversal means visiting all nodes of a binary tree exactly once.

1. Preorder Traversal (Root → Left → Right)

- In preorder traversal, the root node is visited first. Then the left subtree and right subtree are visited.

Algorithm (Preorder):

- Visit the root node
- Traverse left subtree
- Traverse right subtree

2. Inorder Traversal (Left → Root → Right)

- In inorder traversal, the left subtree is visited first. Then the root and finally the right subtree.
- In a Binary Search Tree, inorder traversal gives data in sorted order.

Algorithm (Inorder):

- Traverse left subtree
- Visit the root node
- Traverse right subtree

3. Postorder Traversal (Left → Right → Root)

- In postorder traversal, both subtrees are visited before the root node.
- It is useful for deleting a tree and evaluating postfix expressions.

Algorithm (Postorder):

- Traverse left subtree
- Traverse right subtree
- Visit the root node

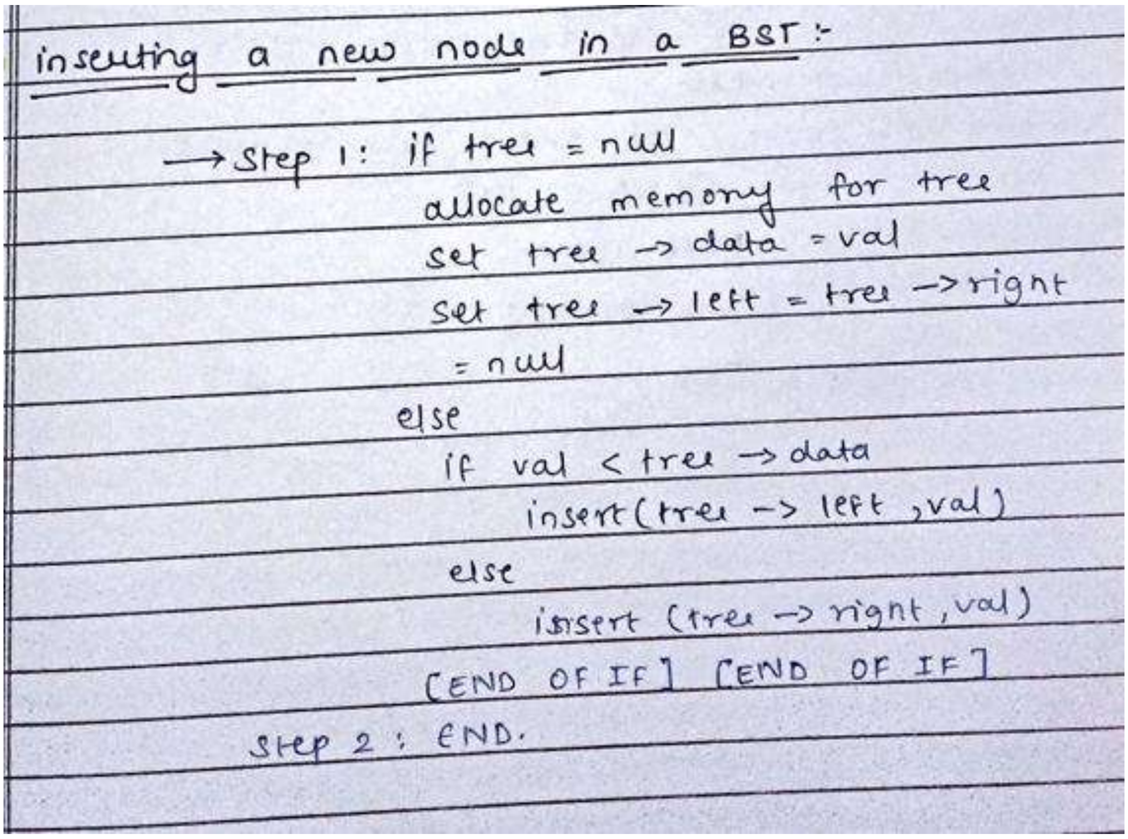
2. **Explain Binary Search Tree with insertion, deletion, and traversal operations using algorithms.**

Binary Search Tree (BST) – Definition

- A Binary Search Tree (BST) is a special type of binary tree used for efficient searching. It follows a strict ordering rule for storing elements.
- In a BST, all values in the left subtree are smaller than the root, and all values in the right subtree are greater than the root. This rule applies to every node in the tree.
- Because of this property, searching, insertion, and deletion operations become faster. When the tree is balanced, the time complexity is O(log n).

Insertion Operation in BST

- Insertion starts from the root node. The new value is compared with the current node value.
- If the new value is smaller, move to the left subtree. If it is greater, move to the right subtree.
- This comparison continues until an empty position (NULL) is found. The new node is inserted there.

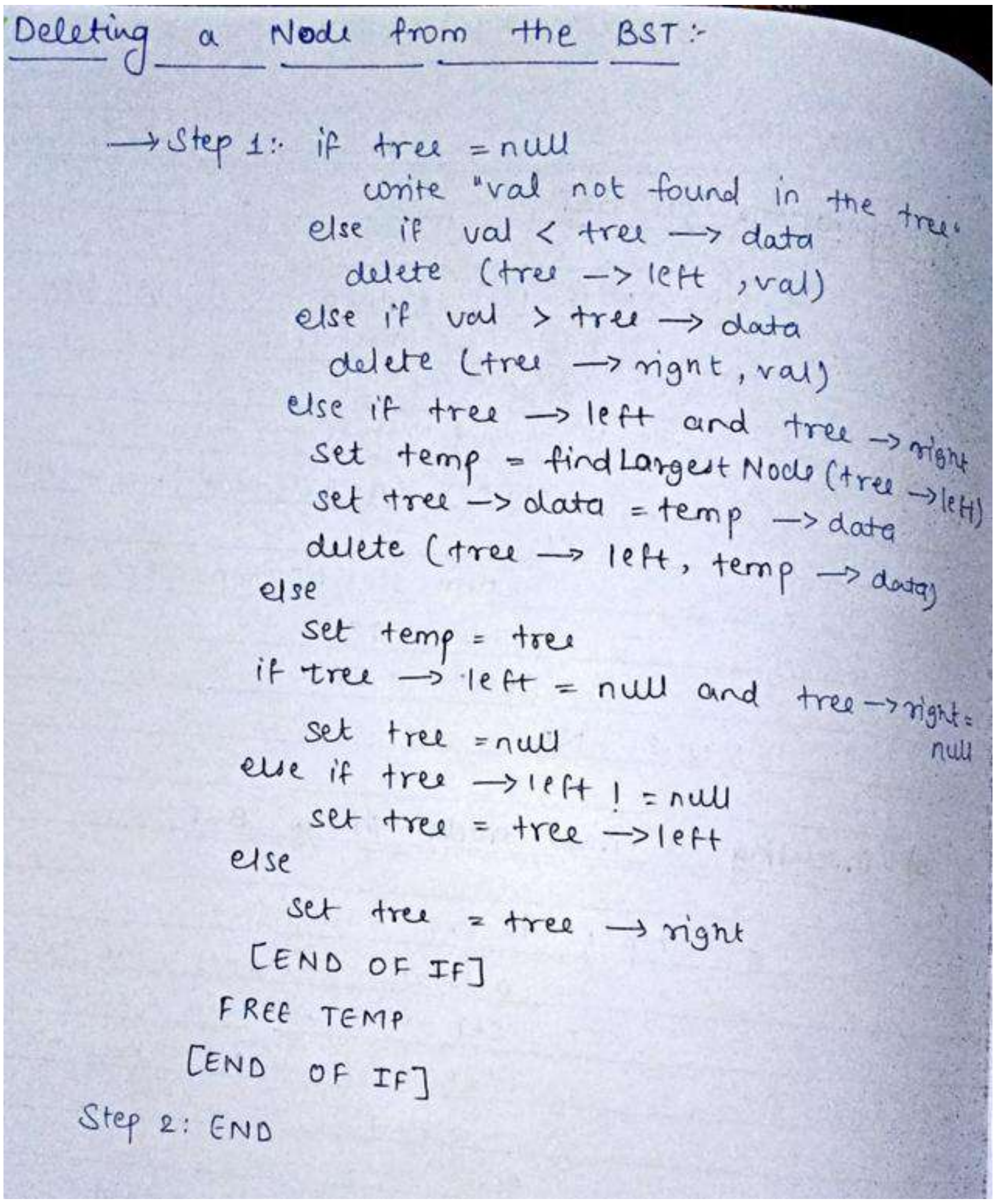


inserting a new node in a BST :-

```
→ Step 1: if tree = null
              allocate memory for tree
              set tree -> data = val
              set tree -> left = tree -> right
               = null
          else
              if val < tree -> data
                  insert(tree -> left ,val)
              else
                  insert (tree -> right ,val)
          [END OF IF] [END OF IF]
   Step 2 : END.
```

Deletion Operation in BST

Deletion is more complex because it depends on the number of children of the node to be deleted.



Deleting a Node from the BST:-

```
→ Step 1:- if tree = null
              write "val not found in the tree"
           else if val < tree → data
              delete (tree → left , val)
           else if val > tree → data
              delete (tree → right , val)
           else if tree → left and tree → right
              set temp = findLargest Node (tree → left)
              set tree → data = temp → data
              delete (tree → left , temp → data)
           else
              set temp = tree
           if tree → left = null and tree → right = null
              set tree = null
           else if tree → left ! = null
              set tree = tree → left
           else
              set tree = tree → right
           [END OF IF]
           FREE TEMP
        [END OF IF]
Step 2: END
```

3. **Explain AVL tree insertion and deletion with balancing and rotations.**

An AVL tree is a self-balancing Binary Search Tree. After every insertion or deletion, it checks the balance factor of each node and performs rotations if required to keep the tree height-balanced.

AVL Tree Insertion with Balancing

- Insertion in an AVL tree is done similar to a BST. The new node is inserted by comparing values and placing it in the correct position.
- After insertion, the height of each node is updated. The balance factor is calculated as Balance Factor = Height of Left Subtree − Height of Right Subtree.
- If the balance factor becomes +2 or −2, the tree becomes unbalanced. Rotations are required to restore balance.
- There are four imbalance cases:
    - LL Rotation: Insertion in left subtree of left child.
    - RR Rotation: Insertion in right subtree of right child.
    - LR Rotation: Insertion in right subtree of left child.
    - RL Rotation: Insertion in left subtree of right child.
- These rotations rearrange nodes locally. They do not disturb the BST property.

AVL Tree Deletion with Balancing

- Deletion in an AVL tree also follows BST deletion rules. The node is deleted based on whether it has zero, one, or two children.
- After deletion, heights of nodes are recalculated. Balance factor is checked from the deleted node up to the root.
- If imbalance is detected, rotations are applied just like insertion. However, deletion may require multiple rotations.
- The same four cases (LL, RR, LR, RL) are used for rebalancing after deletion.
- Balancing ensures the height of the AVL tree remains minimum. This keeps search, insert, and delete operations efficient.

Rotations : (for detail refer 4th question of 5 marks)

LL and RR rotations use a single rotation.

LR and RL rotations use double rotations.

4. **Explain threaded binary tree with suitable diagrams and traversal.**

Threaded Binary Tree – Definition

A threaded binary tree is a special type of binary tree in which NULL pointers are replaced by special links called threads. These threads point to the inorder predecessor or inorder successor of a node.

In a normal binary tree, many left and right pointers are NULL, which wastes memory. Threaded binary trees make use of these NULL pointers efficiently.

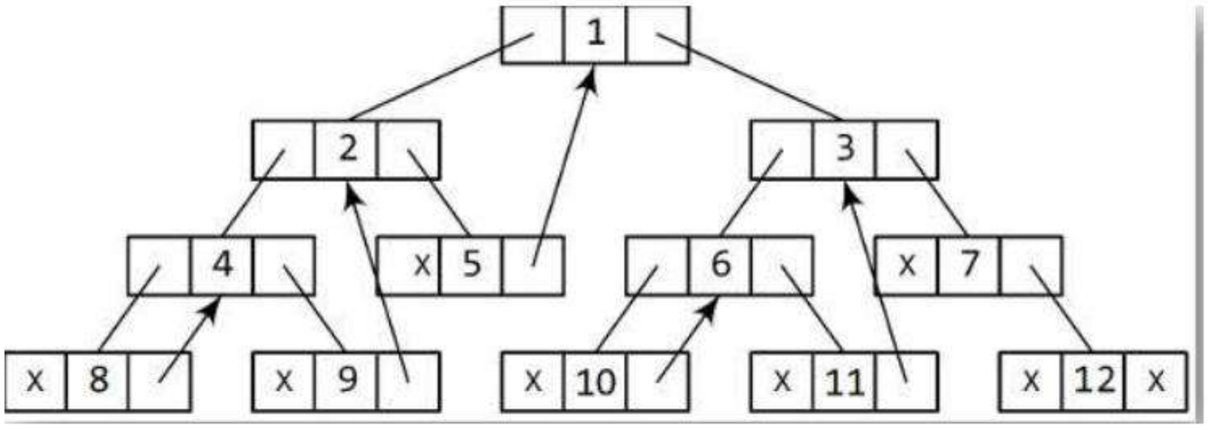
Types of Threaded Binary Trees

A Single Threaded Binary Tree uses only one type of thread. It may have either left threads or right threads.

A Left Threaded Binary Tree uses left NULL pointers to point to the inorder predecessor.

A Right Threaded Binary Tree uses right NULL pointers to point to the inorder successor.

A Double Threaded Binary Tree uses both left and right NULL pointers as threads. This allows traversal in both directions.



5. **Define Binary tree? Explain types of traversal of Binary tree. Reconstruct the binary tree using following traversal—Inorder: Any sequence, Preorder: Any sequence. After construction also write Post-order traversal of the binary tree.**

Definition of Binary Tree

- A binary tree is a non-linear data structure in which each node can have at most two children. These children are called the left child and the right child.
- Binary trees are used to represent hierarchical data efficiently. They are widely used in searching, expression evaluation, and sorting.

Types of Traversal of Binary Tree

Traversal means visiting each node of a binary tree exactly once.

1. Preorder Traversal (Root → Left → Right)

- The root node is visited first, followed by the left subtree and then the right subtree.
- It is mainly used to copy trees and create prefix expressions.

2. Inorder Traversal (Left → Root → Right)

- The left subtree is visited first, then the root, and finally the right subtree.
- In a BST, inorder traversal gives elements in sorted order.

3. Postorder Traversal (Left → Right → Root)

- Both subtrees are visited before visiting the root node.
- It is useful for deleting trees and evaluating postfix expressions.

Reconstruction of Binary Tree

Given Traversals

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

Steps to Construct Binary Tree

- Step 1: The first element of preorder is always the root.
  Root = A
- Step 2: Find A in inorder sequence.
  Left subtree = D B E
  Right subtree = F C
- Step 3: Apply the same logic recursively for left and right subtrees.

  Tree :

```
      A
     / \
    B   C
   / \   \
  D   E   F
```

Postorder Traversal of Constructed Tree

- **Postorder (Left → Right → Root):**
  D E B F C A

6. **Explain B-Tree in detail. Discuss its properties, insertion, deletion, and applications.**

B-Tree – Definition

- A B-Tree is a self-balancing multi-level search tree. It is mainly used for storing large amounts of data on secondary storage like disks.
- Unlike binary trees, a B-Tree node can have more than two children. This reduces the height of the tree and improves performance.

Properties of B-Tree

- A B-Tree of order m can have at most m children and m − 1 keys in one node. The keys inside a node are stored in sorted order.
- Every node (except root) must have at least ⌈m/2⌉ children. This ensures the tree remains balanced.
- The root node can have a minimum of two children if it is not a leaf. If the tree has only one node, the root can be a leaf.
- All leaf nodes appear at the same level. Hence, the B-Tree is always height balanced.
- Searching, insertion, and deletion operations take O(log n) time.

Insertion in B-Tree

- Insertion starts at the leaf node. The key is inserted in sorted order.
- If the node becomes full, it is split into two nodes. The middle key is moved up to the parent node.
- This splitting may continue up to the root. If the root splits, a new root is created.
- This process keeps the B-Tree balanced at all times.

Deletion in B-Tree

- Deletion is done by first finding the key in the tree.
- If the key is in a leaf node, it is removed directly. If the node has fewer keys than allowed, redistribution or merging is done.
- If the key is in an internal node, it is replaced by its predecessor or successor. Then deletion continues.
- After deletion, balancing is ensured by borrowing keys or merging nodes.

Applications of B-Tree

- B-Trees are widely used in database indexing. They allow fast searching of records.
- They are used in file systems to store directories and metadata.
- B-Trees reduce disk access because each node holds multiple keys.
- They are ideal for systems where frequent insertion and deletion occur.

7. **Define Binary tree? Explain types of traversal of Binary tree. Reconstruct the binary tree using following traversal—Inorder: Any sequence, Postorder: Any sequence. After construction also write Pre-order traversal of the binary tree.**

Definition of Binary Tree

- A binary tree is a non-linear data structure in which each node can have at most two children. These children are known as the left child and the right child.
- Binary trees are used to represent hierarchical relationships. They are widely used in searching, sorting, expression evaluation, and file systems.

Types of Traversal of Binary Tree : Traversal means visiting every node exactly once in a specific order. (write prorder , inorder and postorder traversal with def or algo)

Reconstruction of Binary Tree

Given Traversals (Any Sequence Example)

- Inorder Traversal: D B E A F C
- Postorder Traversal: D E B F C A

Steps to Construct the Binary Tree

- Step 1: The last element of postorder traversal is always the root.
  Root = A
- Step 2: Find A in the inorder sequence.
  - Left inorder subtree: D B E
  - Right inorder subtree: F C
- Step 3: Split the postorder sequence accordingly.
  - Left postorder subtree: D E B
  - Right postorder subtree: F C
- Step 4: Repeat the same steps recursively for left and right subtrees.

```
      A
     / \
    B   C
   / \    \
  D   E    F
```

Preorder Traversal of Constructed Tree

- Preorder (Root → Left → Right):
  A B D E C F