TREES
INTRODUCTION

A tree is recursively defined as a set of one or more nodes where one node is designated as the root of the tree and all the remaining nodes can be partitioned into non-empty sets each of which is a sub-tree of the root.
Basic Terminology

Root node The root node R is the topmost node in the tree. If R = NULL, then it means the tree is empty.

Sub-trees If the root node R is not NULL, then the trees T1, T2, and T3 are called the sub-trees of R.

Leaf node A node that has no children is called the leaf node or the terminal node.

Path A sequence of consecutive edges is called a path. For example, in Fig., the path from the root node A to node I is given as: A, D, and I.

Ancestor node An ancestor of a node is any predecessor node on the path from root to that node. The root node does not have any ancestors. In the tree given in Fig., nodes A, C, and G are the ancestors of node K.
**Descendant node** A descendant node is any successor node on any path from the node to a leaf node. Leaf nodes do not have any descendants. In the tree given in Fig., nodes C, G, J, and K are the descendants of node A.

**Level number** Every node in the tree is assigned a level number in such a way that the root node is at level 0, children of the root node are at level number 1. Thus, every node is at one level higher than its parent. So, all child nodes have a level number given by parent’s level number + 1.

**Degree** Degree of a node is equal to the number of children that a node has. The degree of a leaf node is zero.

**In-degree** In-degree of a node is the number of edges arriving at that node.

**Out-degree** Out-degree of a node is the number of edges leaving that node.
TYPES OF TREES

Trees are of following 6 types:

1. General trees
2. Forests
3. Binary trees
4. Binary search trees
5. Expression trees
6. Tournament trees
TYPES OF TREES

General trees

General trees are data structures that store elements hierarchically. The top node of a tree is the root node and each node, except the root, has a parent. A node in a general tree (except the leaf nodes) may have zero or more sub-trees. General trees which have 3 sub-trees per node are called ternary trees. However, the number of sub-trees for any node may be variable. For example, a node can have 1 sub-tree, whereas some other node can have 3 sub-trees.

Forests

A forest is a disjoint union of trees. A set of disjoint trees (or forests) is obtained by deleting the root and the edges connecting the root node to nodes at level 1.
**TYPES OF TREES**

We have already seen that every node of a tree is the root of some sub-tree. Therefore, all the sub-trees immediately below a node form a forest.

Forest and its corresponding tree
Binary Tree

• A binary tree is a data structure that is defined as a collection of elements called nodes.

• In a binary tree, the topmost element is called the root node, and each node has 0, 1, or at the most 2 children.

• A node that has zero children is called a leaf node or a terminal node. Every node contains a data element, a left pointer which points to the left child, and a right pointer which points to the right child. The root element is pointed by a 'root' pointer.

• If root = NULL, then it means the tree is empty.

Prepare By: Kaushik Vaghani
• **Similar binary trees**: Two binary trees $T$ and $T'$ are said to be similar if both these trees have the same structure.

• **Copies**: Two binary trees $T$ and $T'$ are said to be copies if they have similar structure and if they have same content at the corresponding nodes.

![Similar binary trees](image1)

![T' is a copy of T](image2)
• **Edge**: It is the line connecting a node $N$ to any of its successors. A binary tree of $n$ nodes has exactly $n - 1$ edges because every node except the root node is connected to its parent via an edge.

• **Depth of Node**: The depth of a node $N$ is given as the length of the path from the root $R$ to the node $N$. The depth of the root node is zero.
Complete Binary Tree

• A complete binary tree is a binary tree that satisfies two properties.
  • **First**, in a complete binary tree, every level, except possibly the last, is completely filled.
  • **Second**, all nodes in last level appear as far left as possible.
Full Binary Tree

• A binary tree is a **full binary tree** if it contains the maximum possible number of nodes at all levels.
**Extended Binary Tree**

- A binary tree $T$ is said to be an **extended binary tree** (or a 2-tree) if each node in the tree has either no child or exactly two children.

- Figure shows how an ordinary binary tree is converted into an extended binary tree.

- To convert a binary tree into an extended tree, every empty sub-tree is replaced by a new node. The original nodes in the tree are the internal nodes, and the new nodes added are called the external nodes.
Representation of Binary Trees in the Memory

• *In the linked representation* of a binary tree, every node will have *three parts*: the data element, a pointer to the left node, and a pointer to the right node.

• So in C, the binary tree is built with a node type given below.

```c
struct node
{
    struct node *left;
    int data;
    struct node *right;
};
```

• Every binary tree has a *pointer ROOT*, which points to the root element (topmost element) of the tree. If `ROOT = NULL`, then the tree is empty.
Sequential (Linear) representation of binary trees

- Sequential representation of trees is done **using single or one-dimensional arrays**.

- Though it is the simplest technique for memory representation, **it is inefficient as it requires a lot of memory space**.

- **Rules:**
  1. The root of the tree will be stored in the first location.
  2. Left Child of Parent stored at location \( (i) = (2 \times i) \)
  3. Right Child of Parent stored at location \( (i) = (2 \times i) + 1 \)
Expression Trees

• An Expression tree is a binary tree which stores an arithmetic expression.
• The leaves of the expression tree are operands, and all internal nodes are operators.
• Binary trees are widely used to store algebraic expressions. For example, consider the algebraic expression given as:
• Exp = (a – b) + (c * d)
• Given an expression,

\[( (a + b) - (c * d) ) \% ( (f \wedge g) / (h - i) ) \]

construct the corresponding binary tree.
• Given the binary tree, write down the expression that it represents.

Expression for the above binary tree is

\[
[ \{ (a / b) + (c * d) \} ^ { (f \% g) / (h - i) } ]
\]
• Given the expression,

\[ a + \frac{b}{c} \times d - e \]

construct the corresponding binary tree.
Construct the corresponding binary tree.

• Ex1: \((A ^ B ^ C) + (D - E * F)\)

• Ex2: \(A+(B+C*D+E)+F/G\)

• Ex3: postfix notation: \(A B C * + D E * F + G / -\)

• Ex4: postfix notation: \(ab+cde+**\)
Tournament Trees

• In a tournament tree (also called a selection tree), each external node represents a player and each internal node represents the winner of the match played between the players represented by its children nodes. These tournament trees are also called winner trees because they are being used to record the winner at each level.
Creating a Binary Tree from a General Tree

**Rules:**

**Rule 1:** Root of the binary tree = Root of the general tree

**Rule 2:** Left child of a node in the binary tree = Leftmost child of the node in the general tree

**Rule 3:** Right child of a node in the binary tree = Right sibling of the node in the general tree

Prepare By: Kaushik Vaghani
Convert the given general tree into a binary tree

**Step 1:** Node A is the root of the general tree, so it will also be the root of the binary tree.

**Step 2:** Left child of node A is the leftmost child of node A in the general tree and right child of node A is the right sibling of the node A in the general tree. Since node A has no right sibling in the general tree, it has no right child in the binary tree.
• **Step 3:** Now process node B. Left child of B is E and its right child is C (right sibling in general tree).

• **Step 4:** Now process node C. Left child of C is F (leftmost child) and its right child is D (right sibling in general tree).

• **Step 5:** Now process node D. Left child of D is I (leftmost child). There will be no right child of D because it has no right sibling in the general tree.

• **Step 6:** Now process node I. There will be no left child of I in the binary tree because I has no left child in the general tree. However, I has a right sibling J, so it will be added as the right child of I.
• **Step 7:** Now process node J. Left child of J is K (leftmost child). There will be no right child of J because it has no right sibling in the general tree.

• **Step 8:** Now process all the unprocessed nodes (E, F, G, H, K) in the same fashion, so the resultant binary tree can be given as follows.
Convert the given general tree into a binary tree

• Exercise:
TRAVERSING A BINARY TREE

• Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way.

• Unlike linear data structures in which the elements are traversed sequentially, tree is a nonlinear data structure in which the elements can be traversed in many different ways.

• **Three different algorithms for tree traversals.**

  1. Pre-Order Traversal
  2. In-Order Traversal
  3. Post-Order Traversal

Prepare By: Kaushik Vaghani
Pre-order Traversal:

- To traverse a non-empty binary tree in pre-order, the following operations are performed recursively at each node. The algorithm works by:

1. Visiting the root node,
2. Traversing the left sub-tree, and finally
3. Traversing the right sub-tree.

Note: Pre-order algorithm is also known as the NLR traversal algorithm (Node-Left-Right).

Pre-order: A-B-C
Algorithm for Pre-order Traversal:

**PREORDER(ROOT)**

*Here ROOT is the pointer to the root node of the binary tree.*

• Step 1: Repeat Steps 2 to 4 while ROOT != NULL
• Step 2: Write ROOT → DATA
• Step 3: PREORDER(ROOT → LEFT)
• Step 4: PREORDER(ROOT → RIGHT)
• [END OF LOOP]
• Step 5: END

**Note:** Pre-order traversal algorithms are used to extract a prefix notation from an expression tree.

*(Examples from prev. slides)*
• Examples:


b) TRAVERSAL PREORDER: A, B, D, C, E, F, G, H, I
In-order Traversal:

• To traverse a non-empty binary tree in In-order, the following operations are performed recursively at each node. The algorithm works by:

1. Traversing the left sub-tree,
2. Visiting the root node, and finally
3. Traversing the right sub-tree.

Note: In-order algorithm is also known as the LNR traversal algorithm (Left-Node-Right) or (symmetric traversal).

In-order: B-A-C
Algorithm for In-order Traversal:

**INORDER(ROOT)**

Here *ROOT* is the pointer to the root node of the binary tree.

- Step 1: Repeat Steps 2 to 4 while *ROOT* != NULL
- Step 2: INORDER(*ROOT* → LEFT)
- Step 3: Write *ROOT* → DATA
- Step 4: INORDER(*ROOT* → RIGHT)
- [END OF LOOP]
- Step 5: END

**Note:** In-order traversal algorithms are used to extract a infix notation from an expression tree.

*(Examples from prev. slides)*
• Examples:

a) TRAVERSAL INORDER: G, D, H, L, B, E, A, C, I, F, K, J

b) TRAVERSAL INORDER: B, D, A, E, H, G, I, F, C
Post-order Traversal:

- To traverse a non-empty binary tree in Post-order, the following operations are performed recursively at each node. The algorithm works by:

1. Traversing the left sub-tree,
2. Traversing the right sub-tree, and finally
3. Visiting the root node.

Note: Post-order algorithm is also known as the LRN traversal algorithm (Left-Right-Node).

In-order: B-C-A
Algorithm for Post-order Traversal:

**POSTORDER(ROOT)**

*Here ROOT is the pointer to the root node of the binary tree.*

- Step 1: Repeat Steps 2 to 4 while ROOT != NULL
- Step 2: POSTORDER(ROOT \(\rightarrow\) LEFT)
- Step 3: POSTORDER(ROOT \(\rightarrow\) RIGHT)
- Step 4: Write ROOT \(\rightarrow\) DATA
- [END OF LOOP]
- Step 5: END

**Note:** Post-order traversal algorithms are used to extract a postfix notation from an expression tree.

*(Examples from prev. slides)*
• Examples:

a) TRAVERSAL POSTORDER: G, L, H, D, E, B, I, K, J, F, C, A

b) TRAVERSAL POSTORDER: D, B, H, I, G, F, E, C, A
Constructing a Binary Tree from Traversal Results

• Construct Binary tree from following traversal:

In–order Traversal: D B E A F C G
Pre–order Traversal: A B D E C F G

Step 1 Use the **pre-order sequence** to determine the root node of the tree. The first element would be the root node.

Step 2 Elements on the left side of the root node in the **in-order traversal** sequence form the left sub-tree of the root node. Similarly, elements on the right side of the root node in the **in-order traversal** sequence form the right sub-tree of the root node.
**Step 3** Recursively select each element from pre-order traversal sequence and create its left and right sub-trees from the in-order traversal sequence.
• Construct Binary tree from following traversal:

In–order Traversal: D B H E I A F J C G
Post–order Traversal: D H I E B J F G C A

Step 1 Use the post-order sequence to determine the root node of the tree. The last element would be the root node.
Step 2 Elements on the left side of the root node in the in-order traversal sequence form the left sub-tree of the root node. Similarly, elements on the right side of the root node in the in-order traversal sequence form the right sub-tree of the root node.
Step 3 Recursively select each element from post-order traversal sequence and create its left and right sub-trees from the in-order traversal sequence.
• **Examples 1:**

Inorder : D  B  H  E  A  I  F  J  C  G

Preorder : A  B  D  E  H  C  F  I  J  G

• **Examples 2:**

Inorder : n1  n2  n3  n4  n5  n6  n7  n8  n9

Postorder : n1  n3  n5  n4  n2  n8  n7  n9  n6
• Examples 3:
Preorder : 1 2 4 8 9 5 3 6 7
Postorder : 8 9 4 5 2 6 7 3 1

• Examples 4:
Preorder : u1 u2 u3 u4 u10 u8 u5 u9 u6 u11 u7
Postorder : u4 u10 u3 u8 u2 u9 u11 u7 u6 u5 u1
GTU Questions

• **Examples 1:**
Construct a binary tree from the traversals given below:
Inorder: 1 3 4 6 7 8 10 13 14
Preorder: 8 3 1 6 4 7 10 14 13

• **Examples 2:**
Construct a binary tree from the traversals given below:
Inorder: 1, 10, 11, 12, 13, 14, 15, 17, 18, 21
Postorder: 1, 11, 12, 10, 14, 18, 21, 17, 15, 13
GTU Questions

• Examples 3:
  Given the following traversals create a binary tree from that. Also give the postorder traversal for the same.
  
  preorder = \{7,10,4,3,1,2,8,11\}
  inorder = \{4,10,3,1,7,11,8,2\}

• Examples 4:
  The inorder and preorder traversal of a binary tree are
d b e a f c g
a b d e c f g respectively
Construct binary tree and find its postorder traversal.
APPLICATIONS OF TREES

• Trees are used to store simple as well as complex data. Here simple means an integer value, character value and complex data means a structure or a record.
• Trees are often used for implementing other types of data structures like hash tables, sets and maps.
• A self-balancing tree, Red-black tree is used in kernel scheduling, to preempt massively multiprocessor computer operating system use.
• Another variation of tree, B-trees are prominently used to store tree structures on disc. They are used to index a large number of records.
• B-trees are also used for secondary indexes in databases, where the index facilitates a select operation to answer some range criteria.
• Trees are an important data structure used for compiler construction.
• Trees are also used in database design.
• Trees are used in file system directories.
• Trees are also widely used for information storage and retrieval in symbol tables.
A binary search tree, also known as an ordered binary tree, is a variant of binary trees in which the nodes are arranged in an order.

A tree is called binary search tree if it satisfy following two conditions:

1. All nodes must have at most two children. (Binary tree)
2. In a binary search tree, all the nodes in the left sub-tree have a value less than that of the root node. Correspondingly, all the nodes in the right sub-tree have a value either equal to or greater than the root node.
Since the nodes in a binary search tree are ordered, the time needed to search an element in the tree is greatly reduced.
a binary search tree is a binary tree with the following properties:

1. The left sub-tree of a node N contains values that are less than N’s value.

2. The right sub-tree of a node N contains values that are greater than N’s value.

3. Both the left and the right binary trees also satisfy these properties and, thus, are binary search trees.
State whether the binary trees in below Fig. are binary search trees or not?

- Tree 1: No
- Tree 2: Yes
- Tree 3: No
Create a binary search tree (sequential order binary tree) using the following data elements:
45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81
BINARY SEARCH TREES

45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81

Prepare By: Kaushik Vaghani
GTU Questions:

Example 1: Generate a binary search tree for following numbers and perform in-order and post-order traversals:

50, 40, 80, 20, 0, 30, 10, 90, 60, 70

Example 2: What is a binary search tree? Create a binary search tree:

50, 45, 100, 25, 49, 120, 105, 46, 90, 95

Example 3: Construct a binary search tree. Also do the inorder and postorder traversal for the same:

45, 56, 39, 12, 34, 78, 54, 67, 10, 32, 89, 81

Example 4: Briefly explain advantages of binary search tree. Construct binary search tree for the following elements:

8, 3, 11, 5, 9, 12, 13, 4, 6, 20

Example 5: 68, 85, 23, 38, 44, 80, 30, 108, 26, 5, 92, 60

Example 6: 10, 15, 17, 8, 7, 9, 11, 12, 13, 4, 14, 5
BINARY SEARCH TREES

Operations on Binary Search Tree

• Searching Data

• Inserting Data

• Deleting Data

• Traversing the Tree
Searching for node in Binary Search Tree

Searching a node with value 12 in the given binary search tree
Searching for node in Binary Search Tree

Binsary Trees

(Step 1)
45
39
56
12
54
78
10
34
67
89
32

(Step 2)
45
39
56
12
54
78
10
34
67
89
32

(Step 3)
45
39
56
12
54
78
10
34
67
89
32

(Step 4)
45
39
56
12
54
78
10
34
67
89
32

Searching a node with value 67 in the given binary search tree

Prepared By: Kaushik Vaghani
The search would terminate after reaching node 39 as it does not have any right child.
Algorithm **Searching** for node in Binary Search Tree

```plaintext
searchElement (TREE, VAL)

Step 1: IF TREE -> DATA = VAL OR TREE = NULL
    Return TREE
ELSE
    IF VAL < TREE -> DATA
        Return searchElement(TREE -> LEFT, VAL)
    ELSE
        Return searchElement(TREE -> RIGHT, VAL)
[END OF IF]
[END OF IF]

Step 2: END
```
BINARY SEARCH TREES

Insert node in Binary Search Tree

(Step 1)  
45  
39  56  
54  78

(Step 2)  
45  
39  56  
54  78

(Step 3)  
45  
39  56

(Step 4)  
45  
39  56

(Step 5)  
45  
39  56

(Step 6)  
45  
39  56

(Step 7)  
45  
39  56

Inserting nodes with values 12 and 55 in the given binary search tree

Prepare By: Kaushik Vaghani
Algorithm to **Insert** node in Binary Search Tree

```plaintext
Insert (TREE, VAL)

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
```
Case 1: Deleting a Node that has No Children

Deleting node 78 from the given binary search tree
Case 2: Deleting a Node with One Child

Deleting node 54 from the given binary search tree

To handle this case, the node’s child is set as the child of the node’s parent. In other words, replace the node with its child. Now, if the node is the left child of its parent, the node’s child becomes the left child of the node’s parent. Correspondingly, if the node is the right child of its parent, the node’s child becomes the right child of the node’s parent.
Case 3: Deleting a Node with Two Children

Deleting node 56 from the given binary search tree

To handle this case, replace the node’s value with its *in-order predecessor* (largest value in the left sub-tree) or *in-order successor* (smallest value in the right sub-tree).

The in-order predecessor or the successor can then be deleted using any of the above cases.
Case 3: Deleting a Node with Two Children

Deleting node 56 from the given binary search tree
Algorithm to **Delete** node from Binary Search Tree

Delete (TREE, VAL)

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 = findLargestNode(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
Example 1: First insert 10 and then insert 24. After these insertions, delete 37 and then delete 22 from the following binary search tree. Draw the tree after each operation.
• All the traversal operations on binary search tree are same as binary tree traversal methods: inorder, preorder, postorder.

• Inorder traversal on a binary search tree will give the sorted order of data in ascending order.
THREADED BINARY TREES

• A threaded binary tree is the same as that of a binary tree but with a difference in storing the NULL pointers.

• In the linked representation, a number of nodes contain a NULL pointer, either in their left or right fields or in both. This space that is wasted in storing a NULL pointer can be efficiently used to store some other useful piece of information.

• For example, the NULL entries can be replaced to store a pointer to the in-order predecessor or the in-order successor of the node.

• These special pointers are called *threads and binary trees containing* threads are called *threaded binary trees*.

• *In the* linked representation of a threaded binary tree, threads will be denoted using arrows.
There are many ways of threading a binary tree and each type may vary according to the way the tree is traversed.

The threaded binary tree corresponding to inorder traversal is called inorder threading, and corresponding to preorder traversal is called preorder threading. Similarly with postorder threading. Here we will discuss inorder threading.
Apart from this, a threaded binary tree may correspond to one-way threading or a two-way threading.

**Threaded binary tree**

- One way threading
  - left threaded binary tree
  - right threaded binary tree

- Two way threading
  - fully threaded binary tree

**Prepare By: Kaushik Vaghani**
In **one-way threading**, a thread will appear either in the right field or the left field of the node. A one-way threaded tree is also called a **single-threaded tree**.

If the thread appears in the left field, then the left field will be made to point to the in-order predecessor of the node. Such a one-way threaded tree is called a **left-threaded binary tree**.

On the contrary, if the thread appears in the right field, then it will point to the in-order successor of the node. Such a one-way threaded tree is called a **right threaded binary tree**.

In a **two-way threaded tree**, also called a **double-threaded tree**, threads will appear in both the left and the right field of the node. While the left field will point to the in-order predecessor of the node, the right field will point to its successor. A two-way threaded binary tree is also called a **fully threaded binary tree**.
The in-order traversal of the tree is given as 8, 4, 9, 2, 5, 1, 10, 6, 11, 3, 7, 12

Linked representation of the binary tree (without threading)

• One-way Threading (Right threaded binary tree)

Linked representation of the binary tree with one-way threading (Right)
The in-order traversal of the tree is given as
8, 4, 9, 2, 5, 1, 10, 6, 11, 3, 7, 12

Linked representation of the binary tree (without threading)

• **Two-way Threading  (Fully threaded binary tree)**

Linked representation of the binary tree with two-way threading (Fully)
THREADED BINARY TREES

Memory representation of binary trees: (a) without threading, (b) with one-way, and (c) two-way threading
Advantages of Threaded Binary Tree

• It enables linear traversal of elements in the tree.
• Linear traversal eliminates the use of stacks which in turn consume a lot of memory space and computer time.
• It enables to find the parent of a given element without explicit use of parent pointers.
• Since nodes contain pointers to in-order predecessor and successor, the threaded tree enables forward and backward traversal of the nodes as given by in-order fashion.
GTU Questions:

Example 1: Explain Right-in-threaded, left-in-threaded and full-in-threaded binary trees.

Example 2: Write a short note on Threaded binary tree.
AVL tree is a self-balancing binary search tree invented by G.M. Adelson-Velsky and E.M. Landis in 1962. The tree is named AVL in honour of its inventors.

**Def.**: In an AVL tree, the heights of the two sub-trees of a node may differ by at most one. Due to this property, the AVL tree is also known as a *height-balanced tree*.

The key advantage of using an AVL tree is that it takes $O(\log n)$ time to perform search, insert, and delete operations in an average case as well as the worst case because the height of the tree is limited to $O(\log n)$. 
AVL TREES

• The structure of an AVL tree is the same as that of a binary search tree but with a little difference. In its structure, it stores an additional variable called the **BalanceFactor**. Thus, every node has a balance factor associated with it.

• The **balance factor of a node** is calculated by subtracting the height of its right sub-tree from the height of its left sub-tree.

\[
\text{Balance factor} = \text{Height (left sub-tree)} - \text{Height (right sub-tree)}
\]

• A binary search tree in which every node has a balance factor of –1, 0, or 1 is said to be height balanced.

• A node with any other balance factor is considered to be unbalanced and requires rebalancing of the tree.
AVL TREES

• If the balance factor of a node is 1, then it means that the left sub-tree of the tree is one level higher than that of the right sub-tree. Such a tree is therefore called as a **left-heavy tree**.

• If the balance factor of a node is 0, then it means that the height of the left sub-tree (longest path in the left sub-tree) is equal to the height of the right sub-tree.

• If the balance factor of a node is −1, then it means that the left sub-tree of the tree is one level lower than that of the right sub-tree. Such a tree is therefore called as a **right-heavy tree**.
AVL TREES

(a) Left-heavy AVL tree, (b) right-heavy tree, (c) balanced tree
• In previous slide, all are AVL trees because the balancing factor of every node is either 1, 0, or –1.

• However, insertions and deletions from an AVL tree may disturb the balance factor of the nodes and, thus, rebalancing of the tree may have to be done.

• The tree is rebalanced by performing rotation at the critical node.

• There are four types of rotations: **LL rotation, RR rotation, LR rotation, and RL rotation.**

• The type of rotation that has to be done will vary depending on the particular situation.
AVL TREES

Operations on AVL Trees

1. Searching for a Node in an AVL Tree

• Searching in an AVL tree is performed exactly the same way as it is performed in a binary search tree. Due to the height-balancing of the tree, the search operation takes $O(\log n)$ time to complete.

• Since the operation does not modify the structure of the tree, no special provisions are required.
Inserting a New Node in an AVL Tree

• Insertion in an AVL tree is also done in the same way as it is done in a binary search tree. In the AVL tree, the new node is always inserted as the leaf node.

• But the step of insertion is usually followed by an additional step of rotation. Rotation is done to restore the balance of the tree.

• However, if insertion of the new node does not disturb the balance factor, that is, if the balance factor of every node is still –1, 0, or 1, then rotations are not required.

• Cont.....
AVL TREES

• During insertion, the new node is inserted as the leaf node, so it will always have a balance factor equal to zero. The only nodes whose balance factors will change are those which lie in the path between the root of the tree and the newly inserted node.

The possible changes which may take place in any node on the path are as follows:
1. Initially, the node was either left- or right-heavy and after insertion, it becomes balanced.
2. Initially, the node was balanced and after insertion, it becomes either left- or right-heavy.
3. Initially, the node was heavy (either left or right) and the new node has been inserted in the heavy sub-tree, thereby creating an unbalanced sub-tree. Such a node is said to be a critical node.

• Cont.....
If we insert a new node with the value 30, then the new tree will still be balanced and no rotations will be required in this case.

• Cont.....
AVL TREES

Note that there are three nodes in the tree that have their balance factors 2, –2, and –2, thereby disturbing the AVLness of the tree. So, here comes the need to perform rotation.

•Cont.....
• To perform rotation, our first task is to find the critical node. Critical node is the nearest ancestor node on the path from the inserted node to the root whose balance factor is neither –1, 0, nor 1.
• Here, the critical node is 72.

• The second task in rebalancing the tree is to determine which type of rotation has to be done.

• There are four types of rebalancing rotations and application of these rotations depends on the position of the inserted node with reference to the critical node.

• Cont.....
The four categories of rotations are:

1. **LL rotation**: The new node is inserted in the left sub-tree of the left sub-tree of the critical node.

2. **RR rotation**: The new node is inserted in the right sub-tree of the right sub-tree of the critical node.

3. **LR rotation**: The new node is inserted in the right sub-tree of the left sub-tree of the critical node.

4. **RL rotation**: The new node is inserted in the left sub-tree of the right sub-tree of the critical node.

•Cont.....
**AVL TREES**

**LL Rotation**

- Tree (a) is an AVL tree. In tree (b), a new node is inserted in the left sub-tree of the left sub-tree of the critical node A.
- So we apply LL rotation as shown in tree (c).
- While rotation, node B becomes the root, with T1 and A as its left and right child. T2 and T3 become the left and right sub-trees of A.

**Cont.....**
**LL Rotation**

**Example:**
Consider the AVL tree given below and insert 18 into it.

![AVL Tree Diagram]

1. **Step 1:** Insert 18 into the tree.
2. **Step 2:** After insertion, the tree is rebalanced if necessary.
Tree (a) is an AVL tree. In tree (b), a new node is inserted in the right sub-tree of the right sub-tree of the critical node A (node A is the critical node because it is the closest ancestor whose balance factor is not –1, 0, or 1), so we apply RR rotation as shown in tree (c). Note that the new node has now become a part of tree T3. While rotation, node B becomes the root, with A and T3 as its left and right child. T1 and T2 become the left and right sub-trees of A.

•Cont.....
AVL TREES

**RR Rotation**

**Example:**
Consider the AVL tree given below and insert 89 into it.

```
        45
      /   \
     36    63
      /     /
     54    72
```

```
        45
      /   \
     36    63
      /     /
     54    72
        /  \
      89   0
```

(Step 1)

```
        45
      /   \
     36    72
      /     /
     54    89
```

(Step 2)

**Cont.....**
LR Rotations

(a)  

(b)  

Height = h

(c)
**LR Rotation**

**Example:**
Consider the AVL tree given below and insert 37 into it.

![AVL Tree Diagram]

- **Step 1:** Insert 37 into the tree.
- **Step 2:** The tree is balanced again.

• Cont.....
AVL TREES

**RL Rotations**

- **(a)**
  - Node A with a balance factor of -1.
  - Nodes B and C with balance factors of 0.
  - Node T₁ with height h.
  - Node T₄ with height h.
  - Nodes T₂ and T₃ with height h-1.

- **(b)**
  - New node formed.
  - Height = h.

- **(c)**
  - Node A with a balance factor of 0.
  - Nodes B and C with balance factors of -1.
  - Node T₁ with height h.
  - Node T₄ with height h.
  - Nodes T₂ and T₃ with height h-1.

**Cont.....**

Prepare By: Kaushik Vaghani
Example: 1
Construct an AVL tree by inserting the following elements in the given order.
63, 9, 19, 27, 18, 108, 99, 81.
AVL TREES

63, 9, 19, 27, 18, 108, 99, 81.

•Cont.....
AVL TREES

Example: 2
Construct an AVL tree by inserting the following elements in the given order.
Set 1: 1,2,3,4,5,6,7,8,9,10,11,12

Set 2: 8,9,10,2,1,5,6,4,7,11,12,3
AVL TREES

GTU Questions:

1) Insert 1, 29, 32 and 13 in the following Height balanced tree. For each insertion, draw the balanced tree using AVL rotation.

2) Define an AVL tree. Obtain an AVL tree by inserting one integer at a time in the following sequence. 150, 155, 160, 115, 110, 140, 120, 145, 130, 147, 170, 180. Show all the steps.

•Cont.....
AVL TREES

GTU Questions:

3) Define height balanced tree. Construct a height balanced binary tree (AVL tree) for the following data 32, 16, 44, 52, 78, 40, 12, 22, 02, 23

4) Construct the AVL search tree by inserting the following elements in the order of their occurrence. 64, 1, 44, 26, 13, 110, 98, 85
3. **Deleting a Node from an AVL Tree**

• Deletion of a node in an AVL tree is similar to that of binary search trees. But it goes one step ahead. Deletion may disturb the AVLness of the tree, so to rebalance the AVL tree, we need to perform rotations.

• There are two classes of rotations that can be performed on an AVL tree after deleting a given node.

• These rotations are **R rotation and L rotation**.

• Cont.....
AVL TREES

• On deletion of node X from the AVL tree, if node A becomes the critical node (closest ancestor node on the path from X to the root node that does not have its balance factor as 1, 0, or –1), then the type of rotation depends on whether X is in the left sub-tree of A or in its right sub-tree.

• If the node to be deleted is present in the left sub-tree of A, then L rotation is applied, else if X is in the right sub-tree, R rotation is performed.

• Further, there are three categories of L and R rotations. The variations of L rotation are L−1, L0, and L1 rotation. Correspondingly for R rotation, there are R0, R−1, and R1 rotations.

• Cont.....
AVL TREES

R0 Rotation

• Let B be the root of the left or right sub-tree of A (critical node). R0 rotation is applied if the balance factor of B is 0.

- Tree (a) is an AVL tree. In tree (b), the node X is to be deleted from the right sub-tree of the critical node A. Since the balance factor of node B is 0, we apply R0 rotation as shown in tree (c).

• Cont.....
Example:

Consider the AVL tree given in below fig. and delete 72 from it.
AVL TREES

R1 Rotation

• Let B be the root of the left or right sub-tree of A (critical node). R1 rotation is applied if the balance factor of B is 1. **Observe that R0 and R1 rotations are similar to LL rotations.**

![Diagram of AVL Tree R1 Rotation](image)

• Tree (a) is an AVL tree. In tree (b), the node X is to be deleted from the right sub-tree of the critical node A. Since the balance factor of node B is 1, we apply R1 rotation as shown in tree (c).

• *Continued...*
Consider the AVL tree given in the below figure and delete 72 from it.

• Cont.....
AVL TREES

R–1 Rotation

• Let B be the root of the left or right sub-tree of A (critical node). R–1 rotation is applied if the balance factor of B is –1. Observe that R–1 rotation is similar to LR rotation.

![Diagram of AVL Trees]

Cont.....
Example:

Consider the AVL tree given in below Fig. and delete 72 from it.

\[ \text{Cont.....} \]
Delete nodes 52, 36, and 61 from the AVL tree given in below Fig..
Multi-way Search Trees

• binary search tree contains one value and two pointers, left and right, which point to the node’s left and right sub-trees, respectively. The structure of a binary search tree node is shown in below Fig.

| Pointer to left sub-tree | Value or Key of the node | Pointer to right sub-tree |

• The same concept is used in an M-way search tree which has \( M - 1 \) values per node and \( M \) subtrees. In such a tree, \( M \) is called the degree of the tree. Eg. - B tree, B+ tree, 2-3 tree.

• Note that in a binary search tree \( M = 2 \), so it has one value and two sub-trees. In other words, every internal node of an M-way search tree consists of pointers to \( M \) sub-trees and contains \( M - 1 \) keys, where \( M > 2 \).
Multi-way Search Trees

• The structure of an M-way search tree node is shown in below Fig.

| P₀ | K₀ | P₁ | K₁ | P₂ | K₂ | ……… | Pₙ₋₁ | Kₙ₋₁ | Pₙ |

In the structure shown, P₀, P₁, P₂, ..., Pₙ are pointers to the node’s sub-trees and K₀, K₁, K₂, ..., Kₙ₋₁ are the key values of the node. All the key values are stored in ascending order. That is, Kᵢ < Kᵢ₊₁ for 0 <= i <= n–2.
Multi-way Search Trees

Some of the basic properties of an M-way search tree

• Note that the key values in the sub-tree pointed by P0 are less than the key value K0. Similarly, all the key values in the sub-tree pointed by P1 are less than K1, so on and so forth. Thus, the generalized rule is that all the key values in the sub-tree pointed by Pi are less than Ki, where 0 ≤ i ≤ n–1.

• Note that the key values in the sub-tree pointed by P1 are greater than the key value K0. Similarly, all the key values in the sub-tree pointed by P2 are greater than K1, so on and so forth. Thus, the generalized rule is that all the key values in the sub-tree pointed by Pi are greater than Ki–1, where 0 ≤ i ≤ n–1.
A B tree is designed to store sorted data and allows search, insertion, and deletion operations to be performed in logarithmic amortized time. A B tree of order m (the maximum number of children that each node can have) is a tree with all the properties of an M-way search tree. In addition it has the following properties:

1. Every node in the B tree has at most (maximum) m children.

2. Every node in the B tree except the root node and leaf nodes has at least (minimum) m/2 children. This condition helps to keep the tree bushy so that the path from the root node to the leaf is very short, even in a tree that stores a lot of data.

3. The root node has at least two children if it is not a terminal (leaf) node.

4. All leaf nodes are at the same level.
An internal node in the B tree can have \( n \) number of children, where \( 0 \leq n \leq m \). It is not necessary that every node has the same number of children, but the only restriction is that the node should have at least \( m/2 \) children. As B tree of order 4 is given in below Fig.
1) Searching for an Element in a B Tree

B tree of order 4

Searching for an element in a B tree is similar to that in binary search trees. Consider the B tree given in above Fig. To search for 59, we begin at the root node. The root node has a value 45 which is less than 59. So, we traverse in the right sub-tree. The right sub-tree of the root node has two key values, 49 and 63. Since 49 <= 59 <= 63, we traverse the right sub-tree of 49, that is, the left sub-tree of 63. This sub-tree has three values, 54, 59, and 61. On finding the value 59, the search is successful.
2) **Inserting a New Element in a B Tree**

In a B tree, all insertions are done at the leaf node level. A new value is inserted in the B tree using the algorithm given below.

1. Search the B tree to find the leaf node where the new key value should be inserted.
2. If the leaf node is not full, that is, it contains less than \( m-1 \) key values, then insert the new element in the node keeping the node’s elements ordered.
3. If the leaf node is full, that is, the leaf node already contains \( m-1 \) key values, then
   - (a) insert the new value in order into the existing set of keys,
   - (b) split the node at its median into two nodes (note that the split nodes are half full), and
   - (c) push the median element up to its parent’s node. If the parent’s node is already full, then split the parent node by following the same steps.
Example: Look at the B tree of order 5 given below and insert 8, 9, 39, and 4 into it.
Multi-way Search Trees

B TREES

Step 2: Insert 9

Step 3: Insert 39

Prepare By: Kaushik Vaghani
Multi-way Search Trees

B TREES

Step 4: Insert 4

<table>
<thead>
<tr>
<th>4</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>21</td>
<td>27</td>
</tr>
<tr>
<td>39</td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>45</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>63</td>
</tr>
<tr>
<td>81</td>
<td>89</td>
</tr>
</tbody>
</table>

Prepare By: Kaushik Vaghani
3) Deleting an Element from a B Tree

Like insertion, deletion is also done from the leaf nodes. There are two cases of deletion. In the first case, a leaf node has to be deleted. In the second case, an internal node has to be deleted. Let us first see the steps involved in deleting a leaf node.

Case 1: Deleting leaf nodes

1. Locate the leaf node which has to be deleted.

2. If the leaf node contains more than the minimum number of key values (more than \( \frac{m}{2} \) elements), then delete the value.

3. Else if the leaf node does not contain \( \frac{m}{2} \) elements, then fill the node by taking an element either from the left or from the right sibling.
Multi-way Search Trees

B TREES

(a) If the left sibling has more than the minimum number of key values, push its largest key into its parent’s node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

(b) Else, if the right sibling has more than the minimum number of key values, push its smallest key into its parent node and pull down the intervening element from the parent node to the leaf node where the key is deleted.

4. Else, if both left and right siblings contain only the minimum number of elements, then create a new leaf node by combining the two leaf nodes and the intervening element of the parent node (ensuring that the number of elements does not exceed the maximum number of elements a node can have, that is, m). If pulling the intervening element from the parent node leaves it with less than the minimum number of keys in the node, then propagate the process upwards, thereby reducing the height of the B tree.
Case 2: Deleting internal nodes

To delete an internal node, promote the successor or predecessor of the key to be deleted to occupy the position of the deleted key. This predecessor or successor will always be in the leaf node. So the processing will be done as if a value from the leaf node has been deleted.
Deleting an Element from a B Tree

Example: Consider the following B tree of order 5 and delete values 93, 201, 180, and 72 from it.
Multi-way Search Trees

B TREES

Step 1: Delete 93

Step 2: Delete 201

Prepare By: Kaushik Vaghani
Multi-way Search Trees

B TREES

Step 3: Delete 180

Step 4: Delete 72

Prepare By: Kaushik Vaghani
Multi-way Search Trees

B TREES

Example: Consider the B tree of order 3 given below and perform the following operations:
(a) insert 121, 87 and then (b) delete 36, 109.
Multi-way Search Trees

B TREES

Step 2: Insert 87

Step 3: Delete 36

Prepare By: Kaushik Vaghani
Multi-way Search Trees

B TREES

Step 3: Delete 36

Step 4: Delete 109

Prepare By: Kaushik Vaghani
Example: Create a B tree of order 5 by inserting the following elements: 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, and 19.
Multi-way Search Trees

B TREES

Step 4: Insert 13

Step 5: Insert 6, 23, 12, 20

Prepare By: Kaushik Vaghani
Multi-way Search Trees

B TREES

Step 6: Insert 26

Step 7: Insert 4

Prepare By: Kaushik Vaghani
Multi-way Search Trees

B TREES

Step 8: Insert 16, 18, 24, 25

Step 9: Insert 19
GTU Questions:

Example 1: Insert the following elements in a B-Tree.
a, g, f, b, k, c, h, n, j

Example 2: Create a B-tree of order 5 by inserting the following data values.
Multi-way Search Trees

2-3 Trees

• In a 2-3 tree, each interior node has either two or three children.

1. Nodes with two children are called 2-nodes. The 2-nodes have one data value and two children
2. Nodes with three children are called 3-nodes. The 3-nodes have two data values and three children (left child, middle child, and a right child)

• This means that a 2-3 tree is not a binary tree. In this tree, all the leaf nodes are at the same level (bottom level).
Inserting a New Element in a 2-3 Tree

• To insert a new value in the 2-3 tree, an appropriate position of the value is located in one of the leaf nodes. If after insertion of the new value, the properties of the 2-3 tree do not get violated then insertion is over. Otherwise, if any property is violated then the violating node must be split.

Splitting a node: A node is split when it has three data values and four children. Here, P is the parent and L, M, R denote the left, middle, and right children.
Inserting a New Element in a 2-3 Tree

Example: Consider the 2-3 tree given below and insert the following data values into it: 39, 37, 42, 47.

Step 1: Insert 39 in the leaf node

The tree after insertion can be given as
2-3 Trees

Step 2: Insert 37 in the leaf node The tree after insertion can be given as below. Note that inserting 37 violates the property of 2-3 trees. Therefore, the node with values 37 and 39 must be split.

After splitting the leaf node, the tree can be given as below.
Multi-way Search Trees

2-3 Trees

Step 3: Insert 42 in the leaf node. The tree after insertion can be given as follows.

Step 4: Insert 47 in the leaf node. The tree after insertion can be given as follows.
Multi-way Search Trees

2-3 Trees

The leaf node has three data values. Therefore, the node is violating the properties of the tree and must be split.

The parent node has three data values. Therefore, the node is violating the properties of the tree and must be split.
Multi-way Search Trees

2-3 Trees

Deleting an Element from a 2-3 Tree

• In the deletion process, a specified data value is deleted from the 2-3 tree. If deleting a value from a node violates the property of a tree, that is, if a node is left with less than one data value then two nodes must be merged together to preserve the general properties of a 2-3 tree.

• In insertion, the new value had to be added in any of the leaf nodes but in deletion it is not necessary that the value has to be deleted from a leaf node. The value can be deleted from any of the nodes. To delete a value x, it is replaced by its in-order successor and then removed. If a node becomes empty after deleting a value, it is then merged with another node to restore the property of the tree.
Deleting an Element from a 2-3 Tree

Example: Consider the 2-3 tree given below and delete the following values from it: 69, 72, 99, 81.
To delete 69, swap it with its in-order successor, that is, 72. 69 now comes in the leaf node. Remove the value 69 from the leaf node.

72 is an internal node. To delete this value swap 72 with its in-order successor 81 so that 72 now becomes a leaf node. Remove the value 72 from the leaf node.
**2-3 Trees**

Now there is a leaf node that has less than 1 data value thereby violating the property of a 2-3 tree. So the node must be merged. To merge the node, pull down the lowest data value in the parent’s node and merge it with its left sibling.

99 is present in a leaf node, so the data value can be easily removed.
2-3 Trees

Now there is a leaf node that has less than 1 data value, thereby violating the property of a 2-3 tree. So the node must be merged. To merge the node, pull down the lowest data value in the parent’s node and merge it with its left sibling.
2-3 Trees

81 is an internal node. To delete this value swap 81 with its in-order successor 90 so that 81 now becomes a leaf node. Remove the value 81 from the leaf node.