G H RAISONI INSTITUTE OF ENGINEERING & TECHNOLOGY

(Approved by AICTE, New Delhi and Recognized by DTE, Maharashtra)

An Autonomous Institute Affiliated to Rashtrasant Tukadoji Maharaj Nagpur University, Nagpur

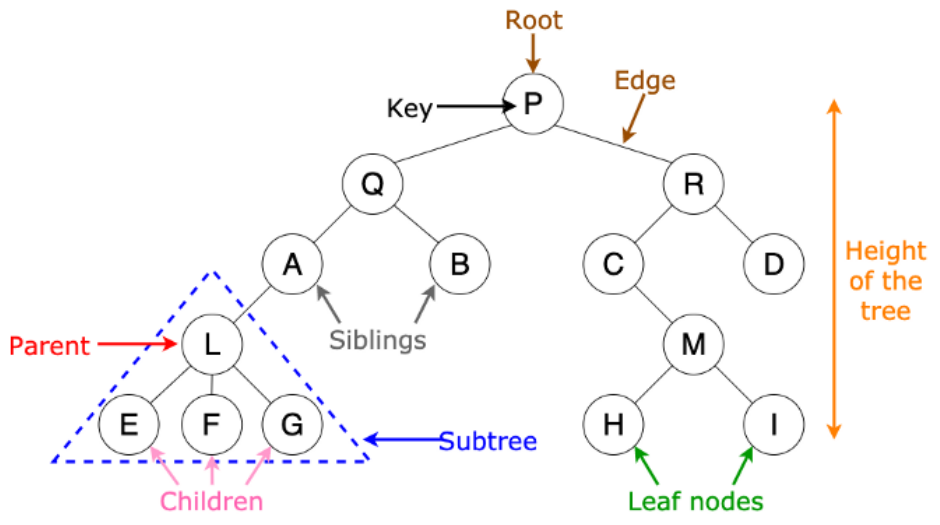
Accredited by NAAC with A+ Grade

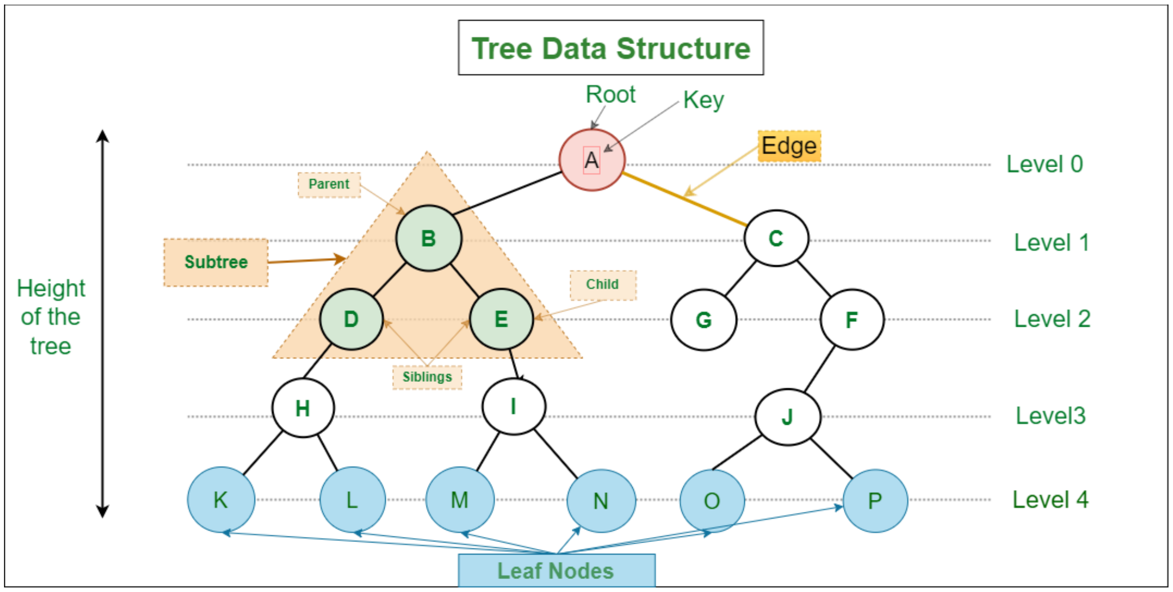
Session 2023-2024

Department of Artificial Intelligence

COURSE : DATA STRUCTURE AND ALGORITHM

UNIT : 4





Tree Terminologies

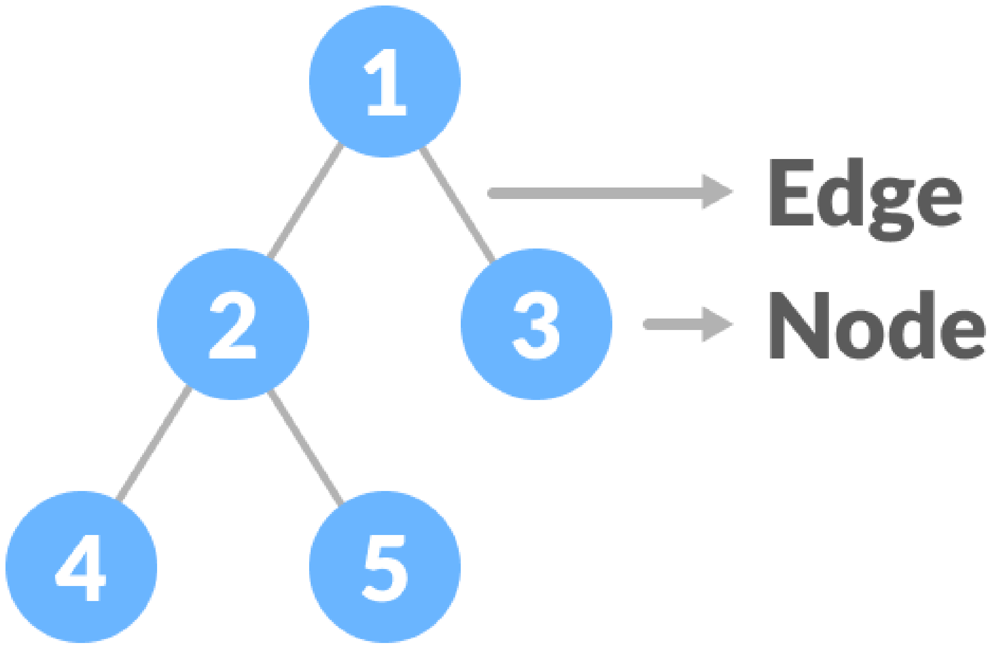
Node

A node is an entity that contains a key or value and pointers to its child nodes.

The last nodes of each path are called **leaf nodes or external nodes** that do not contain a link/pointer to child nodes.
The node having at least a child node is called an **internal node**.

Edge

It is the link between any two nodes.



Root

It is the topmost node of a tree.

Height of a Node

The height of a node is the number of edges from the node to the deepest leaf (ie. the longest path from the node to a leaf node).

Height of a Tree

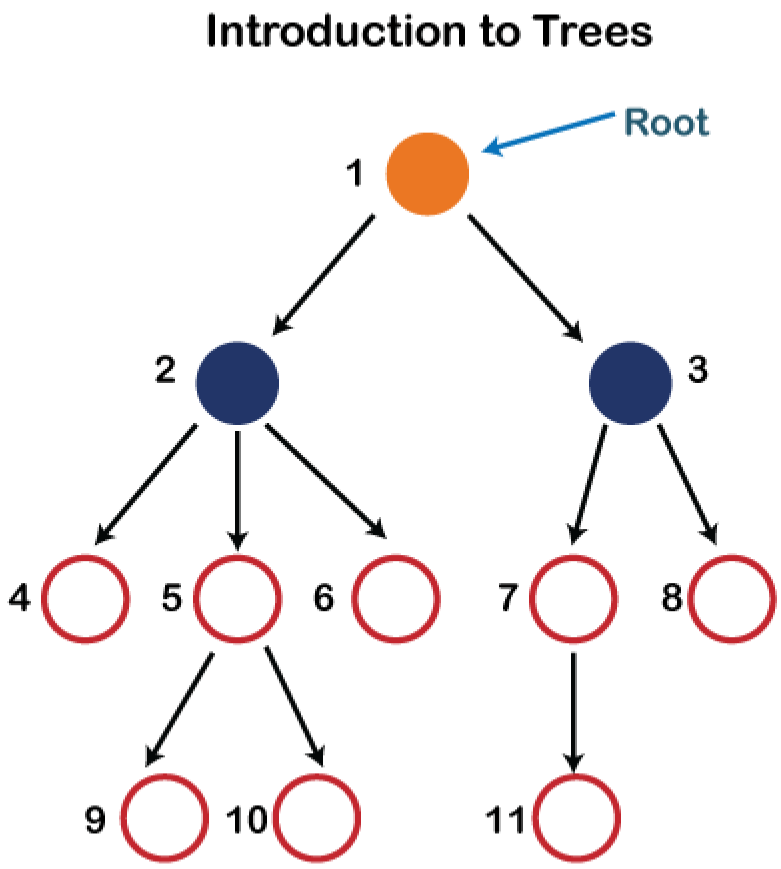
The height of a Tree is the height of the root node or the depth of the deepest node.

Degree of a Node

The degree of a node is the total number of branches of that node.

Forest

A collection of disjoint trees is called a forest.

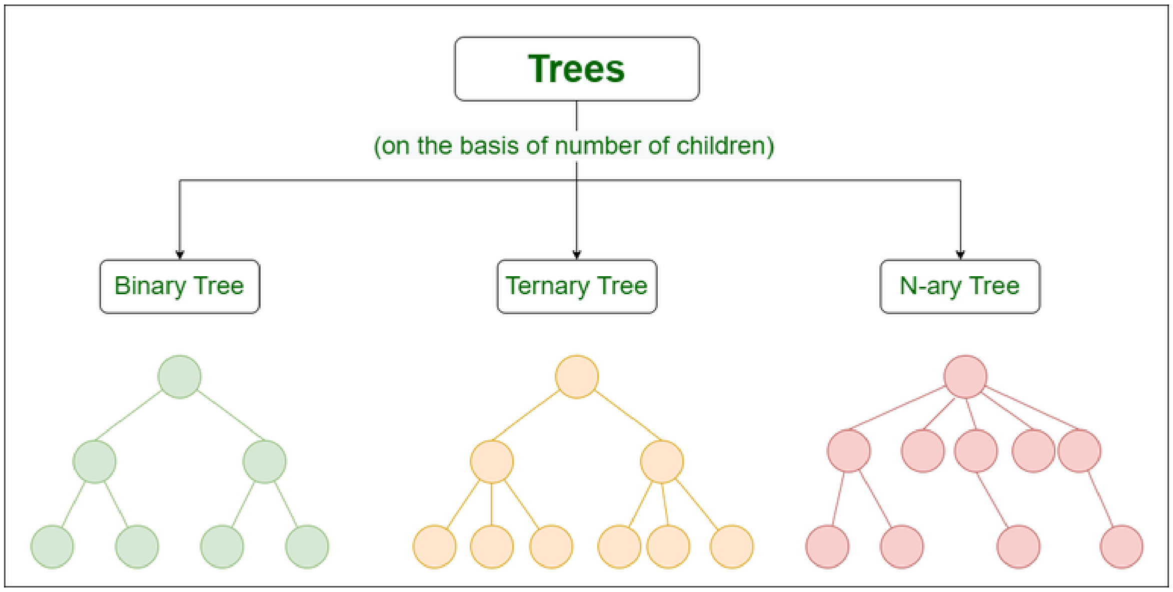


- **Root:** The root node is the topmost node in the tree hierarchy. In other words, the root node is the one that doesn't have any parent. In the above structure, node numbered 1 is **the root node of the tree.** A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree. If a node is directly linked to some other node, it would be called a parent-child relationship.

- **Child node:** If the node is a descendant of any node, then the node is known as a child node.
- **Parent:** If the node contains any sub-node, then that node is said to be the parent of that sub-node.
- **Sibling:** The nodes that have the same parent are known as siblings.
- **Leaf Node:-** The node of the tree, which doesn't have any child node, is called a leaf node. A leaf node is the bottom-most node of the tree. There can be any number of leaf nodes present in a general tree. Leaf nodes can also be called external nodes.
- **Internal nodes:** A node has atleast one child node known as an ***internal***
- **Ancestor node:-** An ancestor of a node is any predecessor node on a path from the root to that node. The root node doesn't have any ancestors. In the tree shown in the above image, nodes 1, 2, and 5 are the ancestors of node 10.
- **Descendant:** The immediate successor of the given node is known as a descendant of a node. In the above figure, 10 is the descendant of node 5.

Representation of a Node in Tree Data Structure:

```
struct Node
{
  int data;
  struct Node *first_child;
  struct Node *second_child;
  struct Node *third_child;
  .
  .
  .
  struct Node *nth_child;
};
```



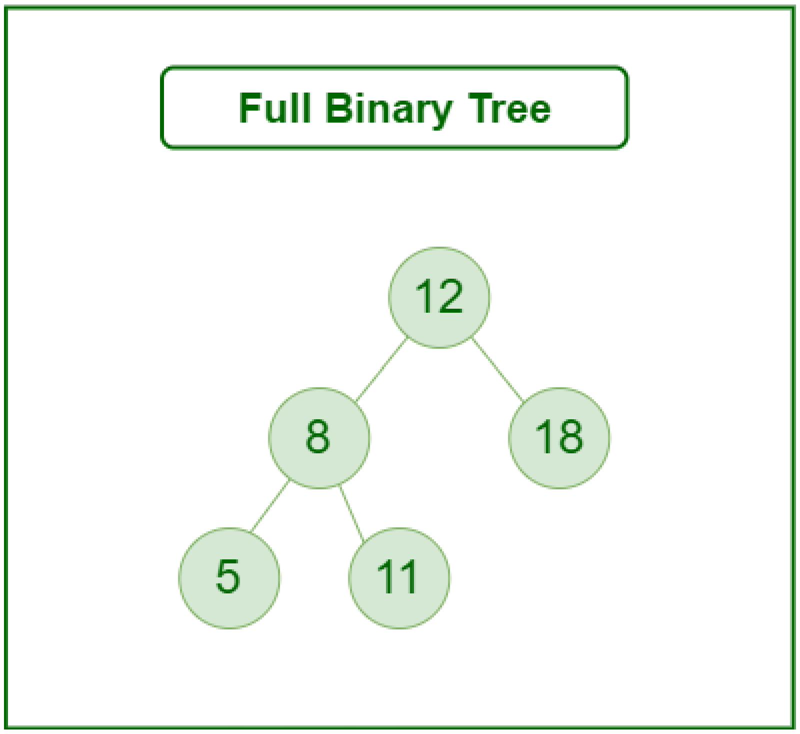
- **Binary tree:** In a binary tree, each node can have a maximum of two children linked to it. Some common types of binary trees include full binary trees, complete binary trees, balanced binary trees, and degenerate or pathological binary trees.
- **Ternary Tree:** A Ternary Tree is a tree data structure in which each node has at most three child nodes, usually distinguished as "left", "mid" and "right".
- **N-ary Tree or Generic Tree:** Generic trees are a collection of nodes where each node is a data structure that consists of records and a list of references to its children(duplicate references are not allowed).

Types of Binary Tree based on the number of children:
Following are the types of Binary Tree based on the number of children:

1. **Full Binary Tree**
2. **Degenerate Binary Tree**
3. **Skewed Binary Trees**

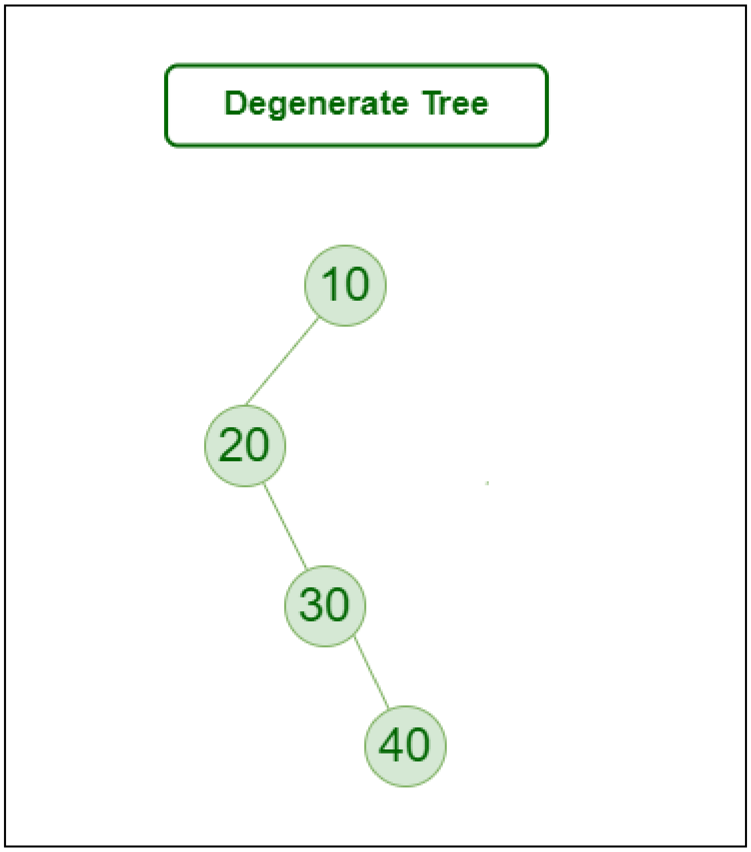
1. **Full Binary Tree**

A Binary Tree is a full binary tree if every node has 0 or 2 children. The following are examples of a full binary tree. We can also say a full binary tree is a binary tree in which all nodes except leaf nodes have two children.



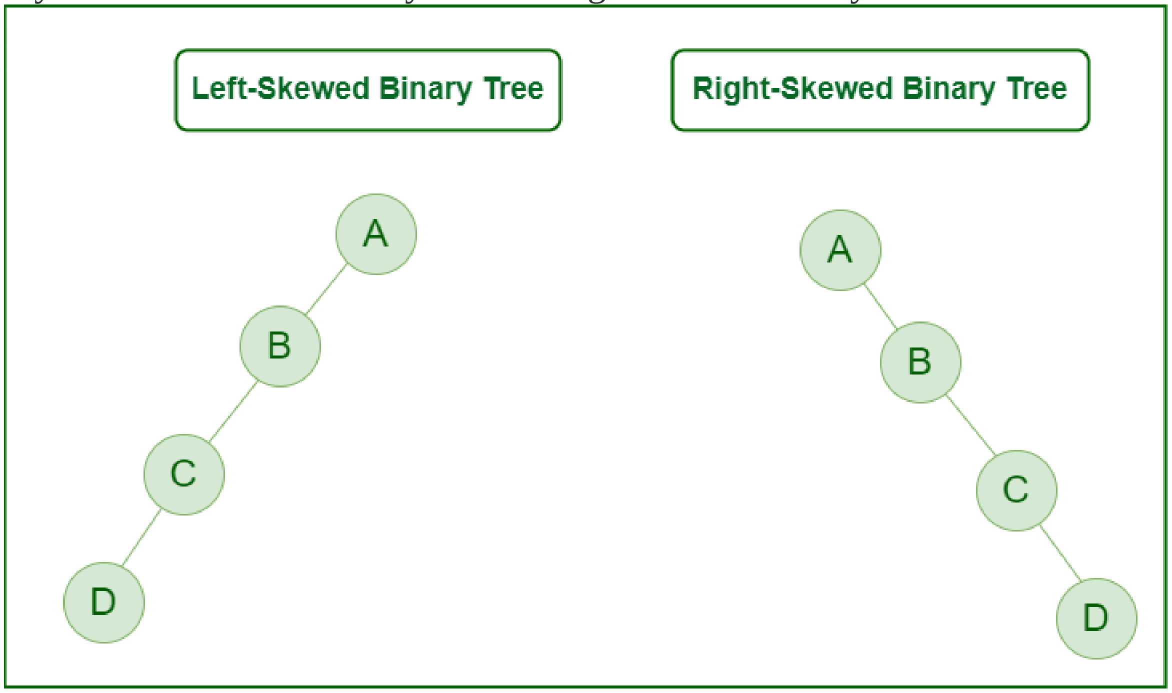
Degenerate (or pathological) tree

A Tree where every internal node has one child. Such trees are performance-wise same as linked list. A degenerate or pathological tree is a tree having a single child either left or right



. Skewed Binary Tree

A skewed binary tree is a pathological/degenerate tree in which the tree is either dominated by the left nodes or the right nodes. Thus, there are two types of skewed binary tree: left-skewed binary tree and right-skewed binary tree.



Types of Binary Tree On the basis of the completion of levels**:**

1. Complete Binary Tree
2. Perfect Binary Tree
3. Balanced Binary Tree

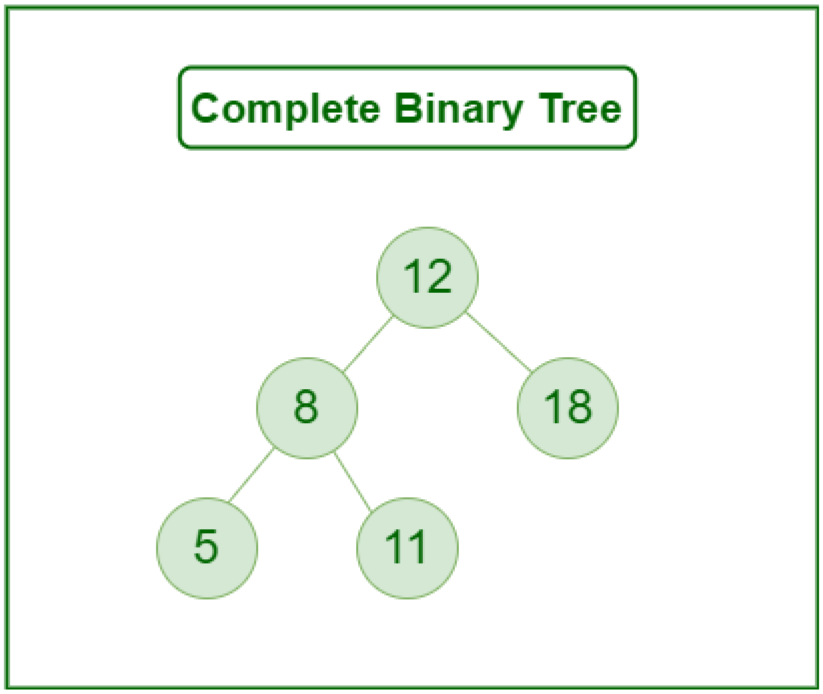
1. Complete Binary Tree

A Binary Tree is a Complete Binary Tree if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible.

A complete binary tree is just like a full binary tree, but with two major differences:

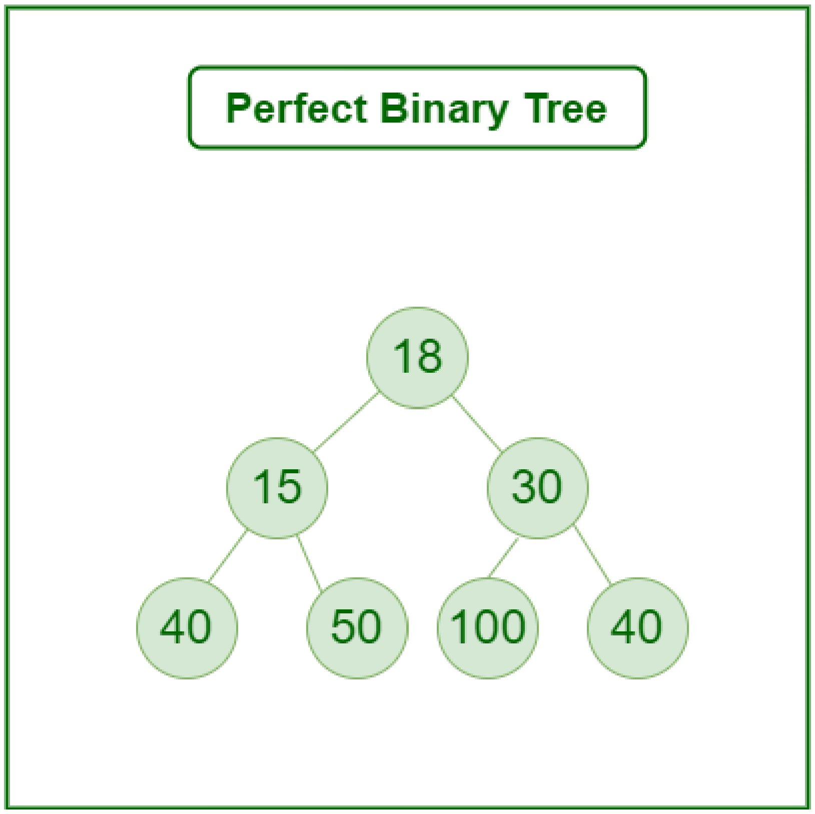
- Every level except the last level must be completely filled.

- All the leaf elements must lean towards the left.
- The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.



Perfect Binary Tree

A Binary tree is a Perfect Binary Tree in which all the internal nodes have two children and all leaf nodes are at the same level.
The following are examples of Perfect Binary Trees.
A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.



In a Perfect Binary Tree, the number of leaf nodes is the number of internal nodes plus 1

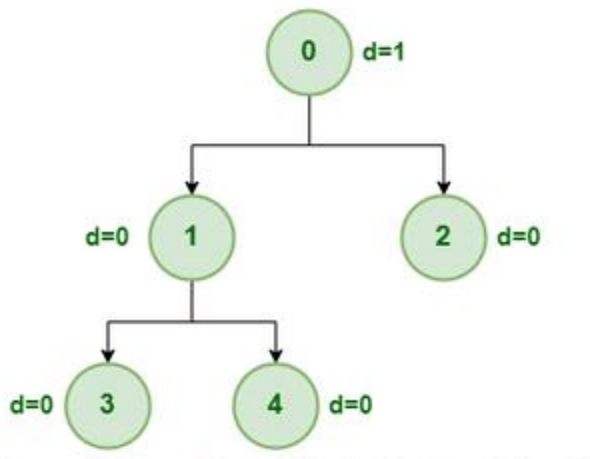
L = I + 1 Where L = Number of leaf nodes, I = Number of internal nodes.

A Perfect Binary Tree of height h (where the height of the binary tree is the number of edges in the longest path from the root node to any leaf node in the tree, height of root node is 0) has $2^{h+1} – 1$ node.

An example of a Perfect binary tree is ancestors in the family. Keep a person at root, parents as children, parents of parents as their children.

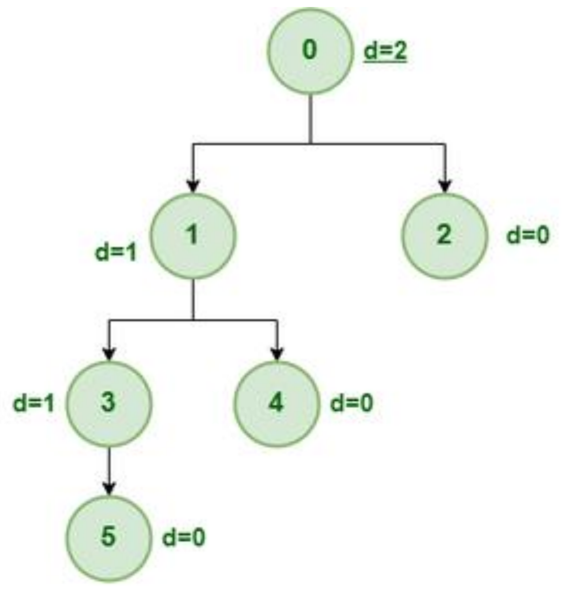
Balanced Binary Tree

A binary tree is balanced if the height of the tree is O(Log n) where n is the number of nodes. For Example, the AVL tree maintains O(Log n) height by making sure that the difference between the heights of the left and right subtrees is at most 1. Red-Black trees maintain O(Log n) height by making sure that the number of Black nodes on every root to leaf paths is the same and that there are no adjacent red nodes. Balanced Binary Search trees are performance-wise good as they provide O(log n) time for search, insert and delete.



Balanced Binary Tree with depth at each level indicated

Depth of a node = |height of left child - height of right child|



Unbalanced Binary Tree with depth at each level indicated

Depth of a node = |height of left child - height of right child|

- Unlike the linked list, each node stores the address of multiple nodes

Basic Operation Of Tree Data Structure:

- **Create** – create a tree in the data structure.
- **Insert** − Inserts data in a tree.
- **Search** − Searches specific data in a tree to check whether it is present or not.

Tree Data Structure can be traversed in following ways:

A)Depth First Search or DFS

1. ***Inorder Traversal***
2. ***Preorder Traversal***
3. ***Postorder Traversal***

- **Traversal**:
    - **Preorder Traversal** – perform Traveling a tree in a pre-order manner in the data structure.

- **In order Traversal** – perform Traveling a tree in an in-order manner.

Post-order Traversal –perform Traveling a tree in a post-order manner

B)Level Order Traversal or Breadth First Search or BFS

2. *Boundary Traversal*
3. *Diagonal Traversal*

-

Why Tree is considered a non-linear data structure?

The data in a tree are not stored in a sequential manner i.e., they are not stored linearly. Instead, they are arranged on multiple levels or we can say it is a hierarchical structure. For this reason, the tree is considered to be a non-linear data structure.

Need for Tree Data Structure

1. One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer:

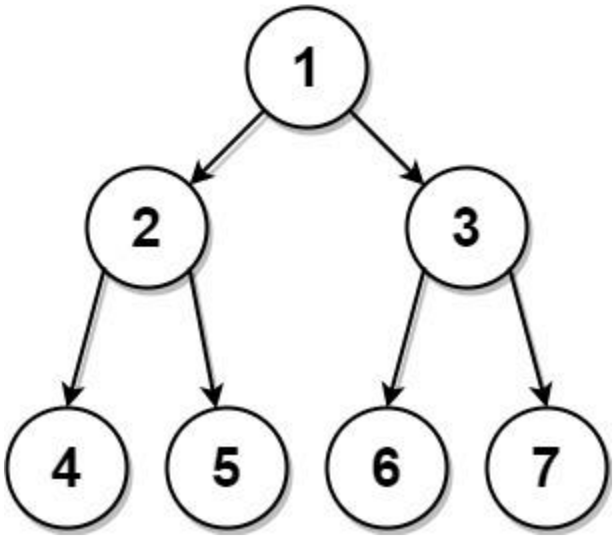
2. Trees (with some ordering e.g., BST) provide moderate access/search (quicker than Linked List and slower than arrays).
3. Trees provide moderate insertion/deletion (quicker than Arrays and slower than Unordered Linked Lists).
4. Like Linked Lists and unlike Arrays, Trees don't have an upper limit on the number of nodes as nodes are linked using pointers.

Application of Tree Data Structure:

- **File System:** This allows for efficient navigation and organization of files.
- **Data Compression**: Huffman coding is a popular technique for data compression that involves constructing a binary tree where the leaves represent characters and their frequency of occurrence. The resulting tree is used to encode the data in a way that minimizes the amount of storage required.
- **Compiler Design:** In compiler design, a syntax tree is used to represent the structure of a program.
- **Database Indexing**: B-trees and other tree structures are used in database indexing to efficiently search for and retrieve data.

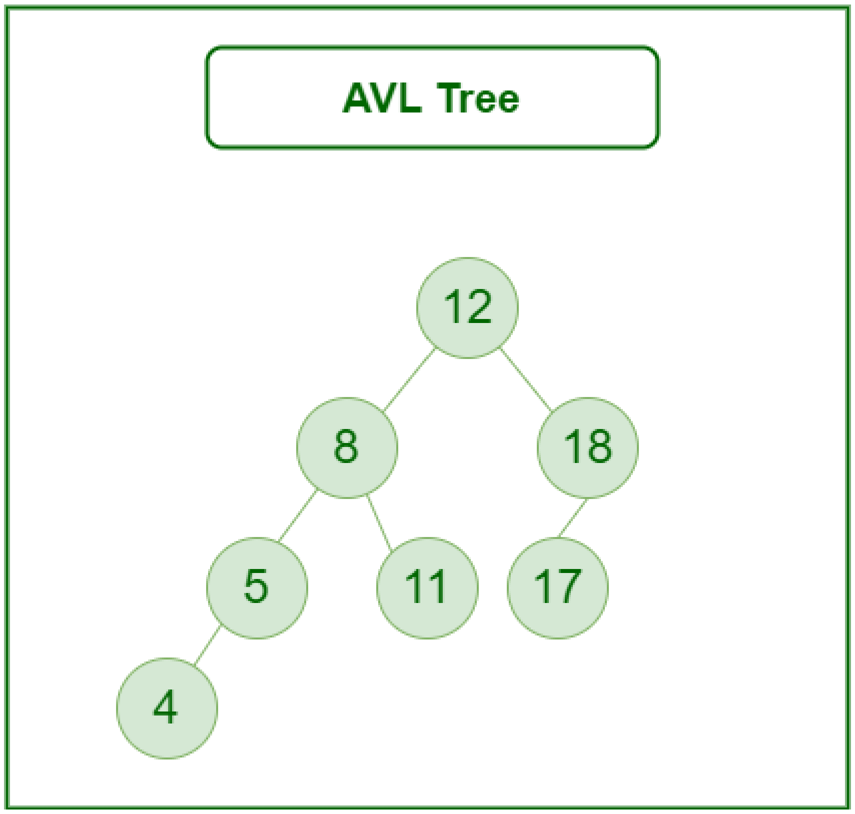
. Construct Binary Tree from Preorder and Postorder Traversal

Input: **preorder = [1,2,4,5,3,6,7], postorder = [4,5,2,6,7,3,1]**
Output: **[1,2,3,4,5,6,7]**



AVL Tree Data Structure

*An **AVL tree** defined as a self-balancing **Binary Search Tree** (BST) where the difference between heights of left and right subtrees for any node cannot be more than one.*
The difference between the heights of the left subtree and the right subtree for any node is known as the **balance factor** of the node.
The AVL tree is named after its inventors, Georgy Adelson-Velsky and Evgenii Landis, who published it in their 1962 paper “An algorithm for the organization of information”.



Operations on an AVL Tree:

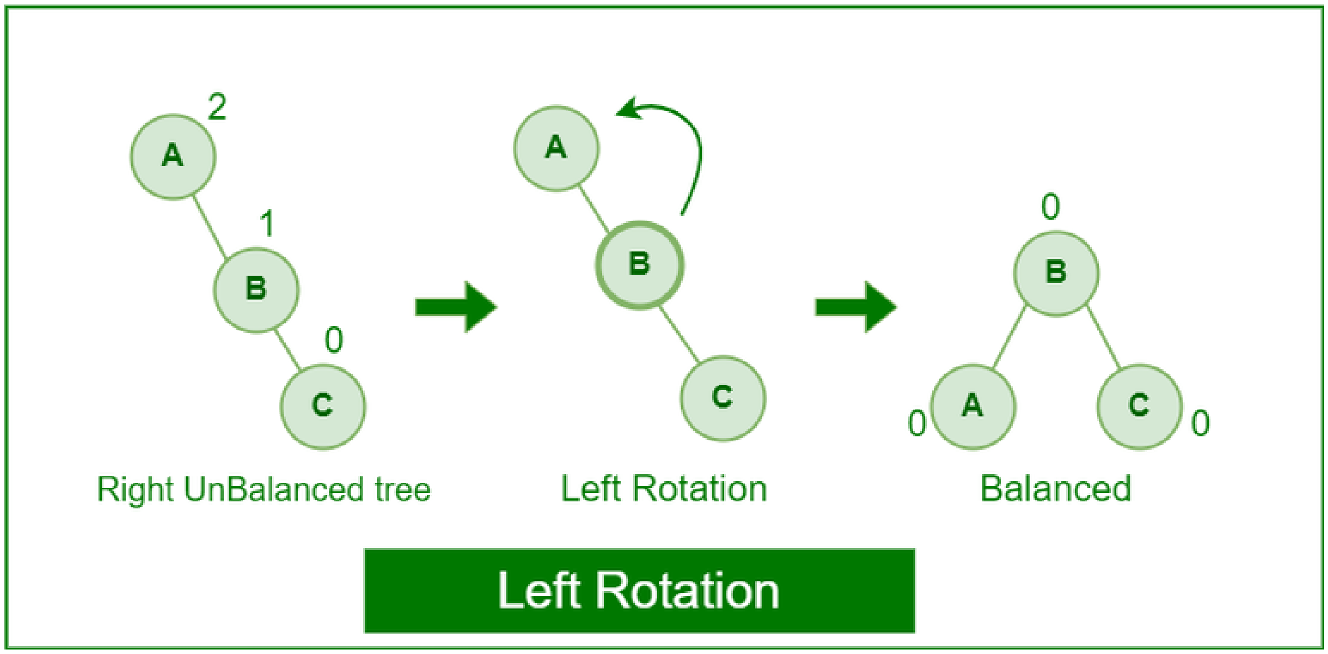
- Insertion

- Deletion
- Searching [It is similar to performing a search in BST]

Rotating the subtrees in an AVL Tree:

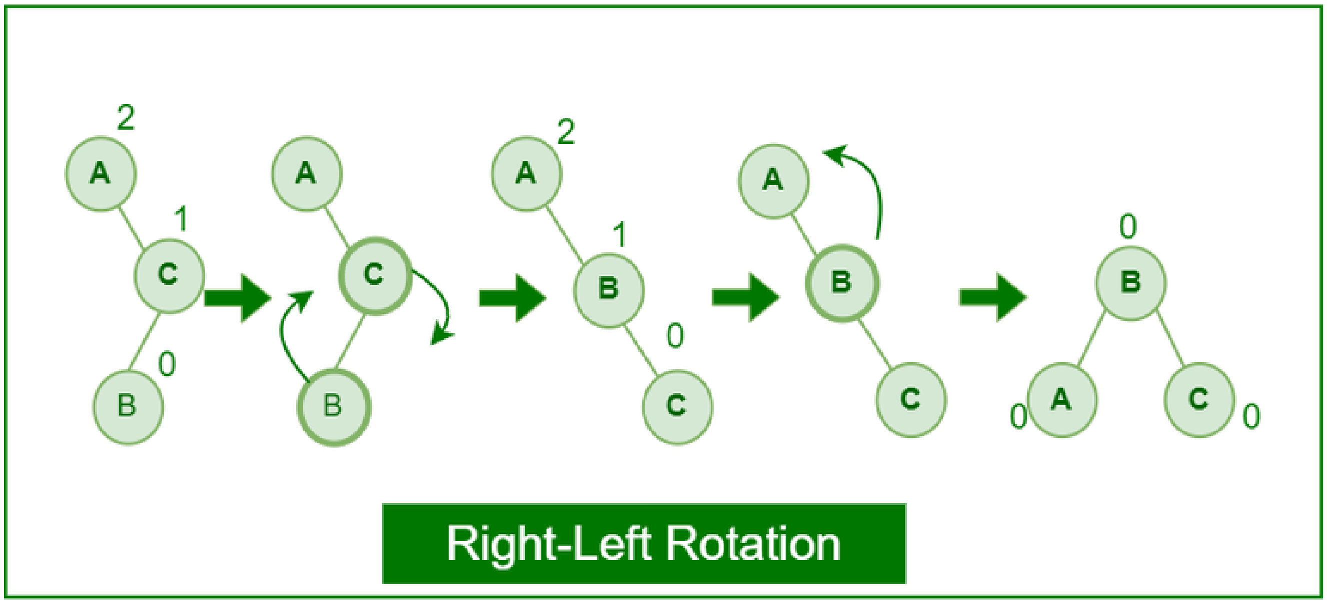
An AVL tree may rotate in one of the following four ways to keep itself balanced:

Left Rotation:
When a node is added into the right subtree of the right subtree, if the tree gets out of balance, we do a single left rotation.

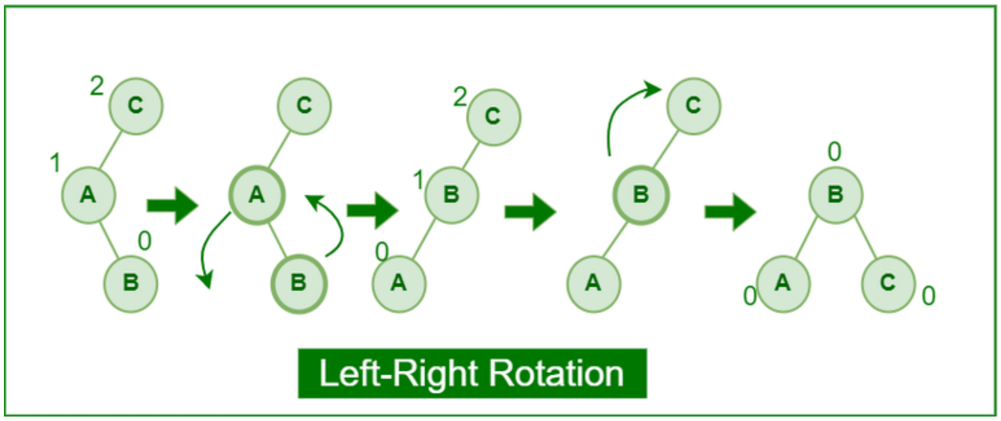


Right Rotation:
If a node is added to the left subtree of the left subtree, the AVL tree may get out of balance, we do a single right rotation.

Right-Left Rotation:
A right-left rotation is a combination in which first right rotation takes place after that left rotation executes.



Left-Right Rotation:
A left-right rotation is a combination in which first left rotation takes place after that right rotation executes.



Applications of AVL Tree:

1. It is used to index huge records in a database and also to efficiently search in that.
2. For all types of in-memory collections, including sets and dictionaries, AVL Trees are used.
3. Database applications, where insertions and deletions are less common but frequent data lookups are necessary
4. Software that needs optimized search.
5. It is applied in corporate areas and storyline games.

Spanning Tree

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..

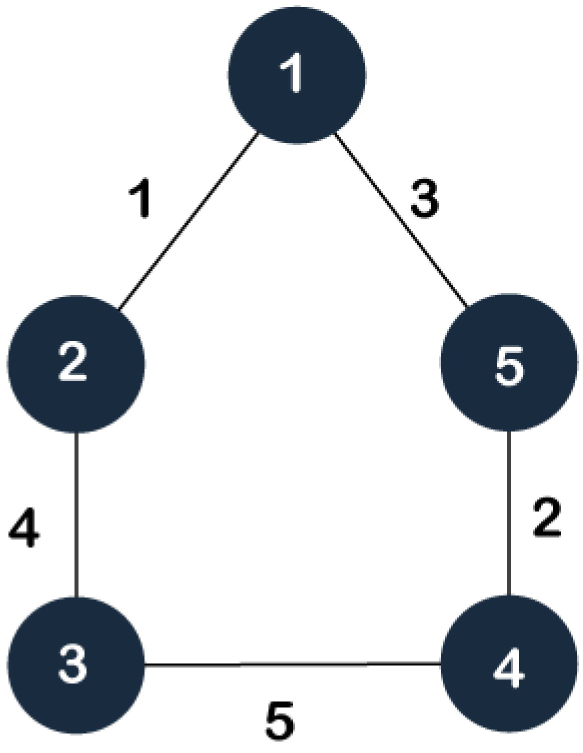
By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.

General Properties of Spanning Tree

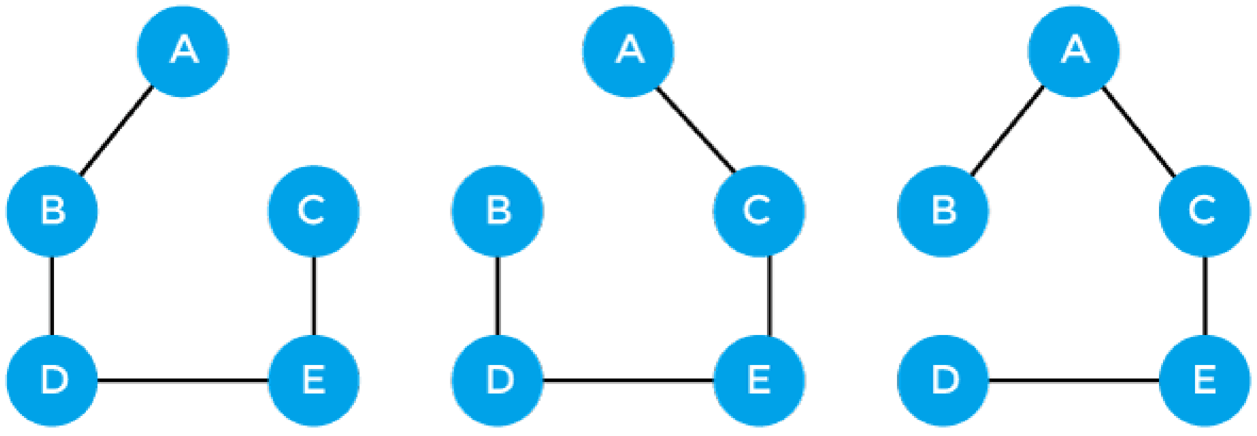
We now understand that one graph can have more than one spanning tree. Following are a few properties of the spanning tree connected to graph G −

- A connected graph G can have more than one spanning tree.

- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is **minimally connected**.
- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is **maximally acyclic**.



Some of the possible spanning trees that will be created from the above graph are given as follows -



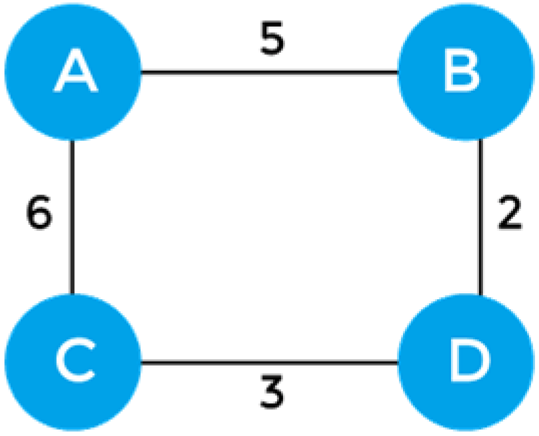
Spanning tree 1 Spanning tree 2 Spanning tree 3

Minimum Spanning tree

A minimum spanning tree can be defined as the spanning tree in which the sum of the weights of the edge is minimum. The weight of the spanning tree is the sum of the weights given to the edges of the spanning tree. In the real world, this weight can be considered as the distance, traffic load, congestion, or any random value.

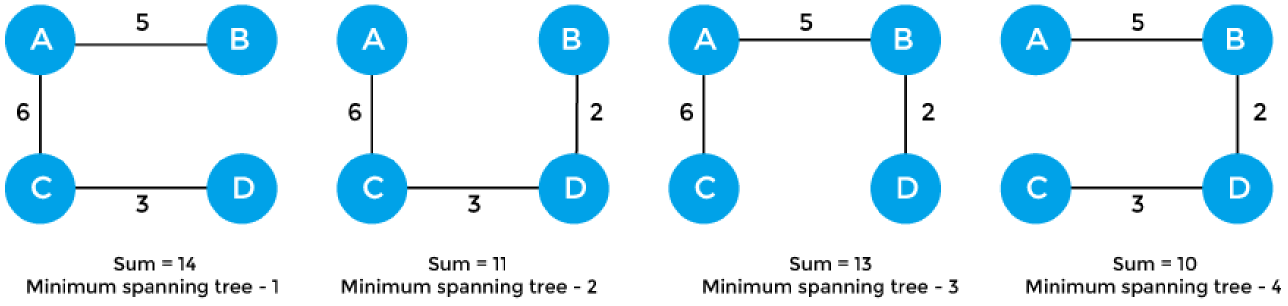
Example of minimum spanning tree

Let's understand the minimum spanning tree with the help of an example.

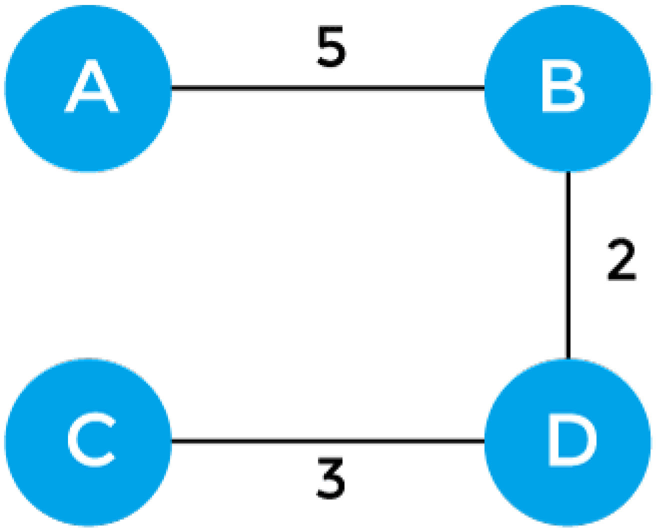


Weighted graph

The sum of the edges of the above graph is 16. Now, some of the possible spanning trees created from the above graph are –


Sum = 14
Minimum spanning tree - 1
Sum = 11
Minimum spanning tree - 2
Sum = 13
Minimum spanning tree - 3
Sum = 10
Minimum spanning tree - 4

o, the minimum spanning tree that is selected from the above spanning trees for the given weighted graph is –



Sum = 10

Applications of minimum spanning tree

The applications of the minimum spanning tree are given as follows -

- Minimum spanning tree can be used to design water-supply networks, telecommunication networks, and electrical grids.
- It can be used to find paths in the map.

Algorithms for Minimum spanning tree

A minimum spanning tree can be found from a weighted graph by using the algorithms given below -

- Prim's Algorithm
- Kruskal's Algorithm

Inorder Traversal:

Algorithm Inorder(tree)

1. *Traverse the left subtree, i.e., call Inorder(left->subtree)*
2. *Visit the root.*
3. *Traverse the right subtree, i.e., call Inorder(right->subtr*

```
program for different tree traversals
#include <stdio.h>
#include <stdlib.h>

// A binary tree node has data, pointer to left child
// and a pointer to right child
struct node {
    int data;
    struct node* left;
    struct node* right;
};

// Helper function that allocates a new node with the
// given data and NULL left and right pointers.
struct node* newNode(int data)
{
    struct node* node
        = (struct node*)malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return (node);
}

```

```
// Given a binary tree, print its nodes in inorder
void printInorder(struct node* node)
{
    if (node == NULL)
        return;

    // First recur on left child
    printInorder(node->left);

    // Then print the data of node
    printf("%d ", node->data);

    // Now recur on right child
    printInorder(node->right);
}

// Driver code
int main()
{
    struct node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);

    // Function call
    printf("Inorder traversal of binary tree is \n");
    printInorder(root);

    getchar();
    return 0;
}
```

Output

```
Inorder traversal of binary tree is

4 2 5 1 3
```

Preorder Traversal:

Algorithm Preorder(tree)

1. ***Visit the root.***
2. ***Traverse the left subtree, i.e., call Preorder(left->subtree)***
3. ***Traverse the right subtree, i.e., call Preorder(right->subtree)***

DIFFERENCE BETWEEN TREE AND GRAPH