Trees

A tree is a (possibly non-linear) data structure made up of nodes or vertices and edges without having any cycle. The tree with no nodes is called the null or empty tree. A tree that is not empty consists of a root node and potentially many levels of additional nodes that form a hierarchy.

Terminologies used in Trees

- Root – The top node in a tree.
- Child – A node directly connected to another node when moving away from the Root.
- Parent – The converse notion of a child.
- Siblings – Nodes with the same parent.
- Descendant – A node reachable by repeated proceeding from parent to child.
- Ancestor – A node reachable by repeated proceeding from child to parent.
- Leaf – A node with no children.
- Internal node – A node with at least one child.
- External node – A node with no children.
- Degree – Number of sub trees of a node.
- Edge – Connection between one node to another.
- Path – A sequence of nodes and edges connecting a node with a descendant.
- Level – The level of a node is defined by 1 + (the number of connections between the node and the root).
- Height of node – The height of a node is the number of edges on the longest downward path between that node and a leaf.
- Height of tree – The height of a tree is the height of its root node

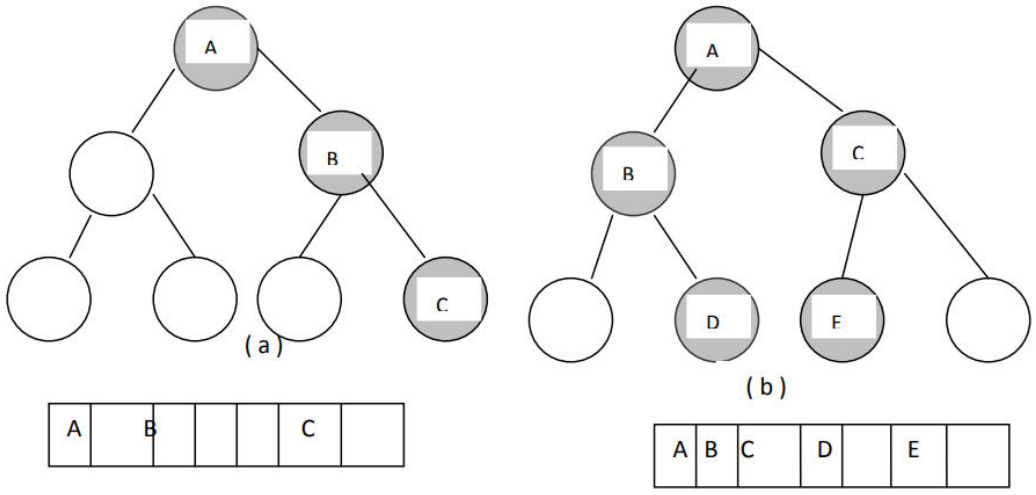
Representation of Binary Tree

There are two representations used to implement binary trees.

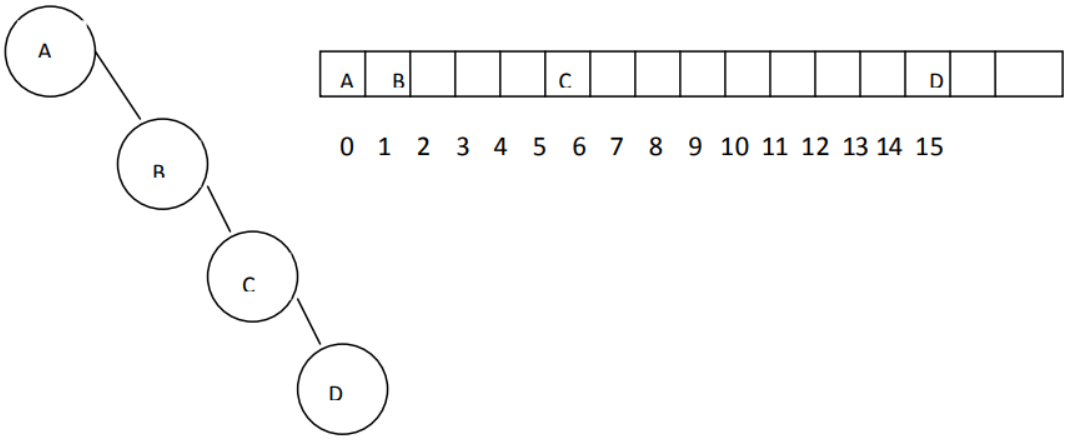
(i) Array Representation

(ii) Linked list Representation

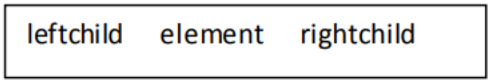
Array Representation: In this, the given binary trees even though are not complete binary trees, they are shown as complete binary trees in which missing elements are un shaded circles. The array representations for the following trees are shown in below.



In array, the elements of the binary are placed in the array according to their number assigned. The array starts indexing from 1. The main drawback of array representation is wasteful of memory when there are many missing elements. The binary tree with n elements requires array size up to 2n . Suppose array positions indexing from 0, then array size reduces to 2n -1. The right skewed binary trees have maximum waste of space. The following right skewed binary tree's array representation is shown as follows.



Linked list Representation: The most popular way to represent a binary tree is by using links or pointers. The node structure used in this representation consists of two pointers and an element for each node. The node structure is given as:



The first field leftchild pointer maintains left child of it. The middle field is the element of the node and last field is the right child pointer maintains right child of it.

Properties of a Binary Tree:

- The maximum number of nodes at level 'l' of a binary tree is 2^l:

Note: Here level is the number of nodes on the path from the root to the node (including root and node). The level of the root is 0
This can be proved by induction:

For root, l = 0, number of nodes = 2^0 = 1
Assume that the maximum number of nodes on level 'l' is 2^l
*Since in a Binary tree every node has at most 2 children, the next level would have twice nodes, i.e. 2 * 2^l*

- The Maximum number of nodes in a binary tree of height 'h' is $2^h - 1$:

Note: Here the height of a tree is the maximum number of nodes on the root-to-leaf path. The height of a tree with a single node is considered as 1
A tree has maximum nodes if all levels have maximum nodes. So the maximum number of nodes in a binary tree of height h is 1 + 2 + 4 + .. + 2^{h-1}. This is a simple geometric series with h terms and the sum of this series is $2^h - 1$.

If the height of the root is considered as 0. In this convention, the above formula becomes $2^{h+1} - 1$

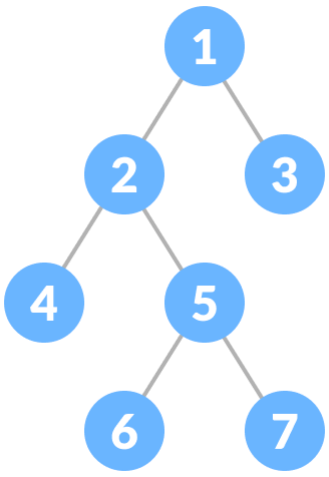
- In a Binary tree where every node has 0 or 2 children, the number of leaf nodes is always one more than nodes with two children.

- **Each node in a binary tree can have at most two child nodes:** In a binary tree, each node can have either zero, one, or two child nodes. If a node has zero children, it is called a leaf node. If a node has one child, it is called a unary node. If a node has two children, it is called a binary node.

Types of Binary Trees:

1. Full/Proper/Strictly Binary Tree

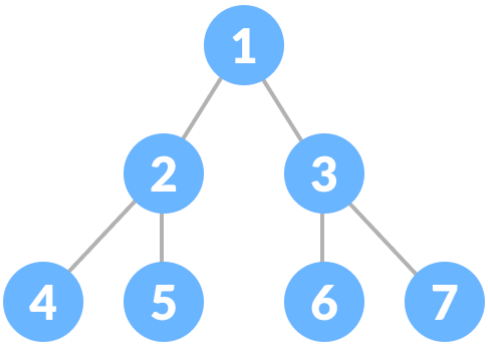
A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children.



Full Binary Tree

2. Perfect Binary Tree

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.

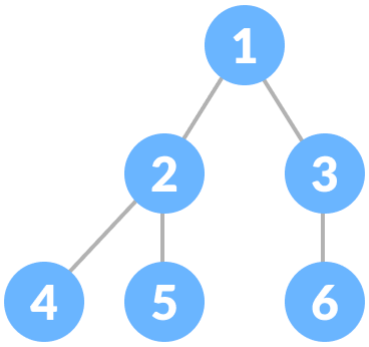


Perfect Binary Tree

3. **Complete Binary Tree**

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

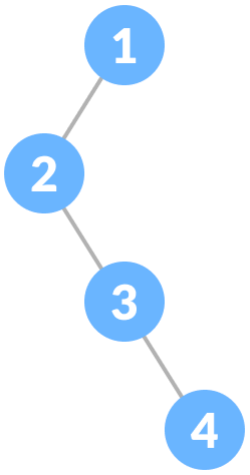
- Every level must be completely filled
- All the leaf elements must lean towards the left.
- The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.



Complete Binary Tree

4. **Degenerate or Pathological Tree**

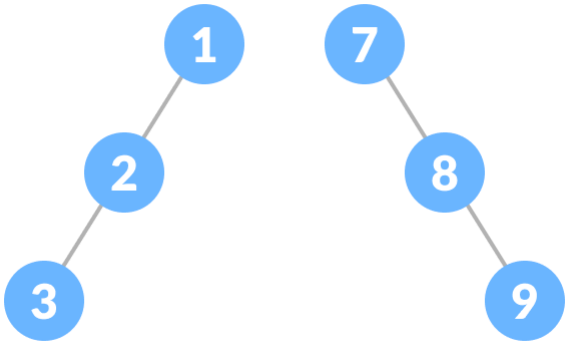
A degenerate or pathological tree is the tree having a single child either left or right.



Degenerate Binary Tree

5. **Skewed Binary Tree**

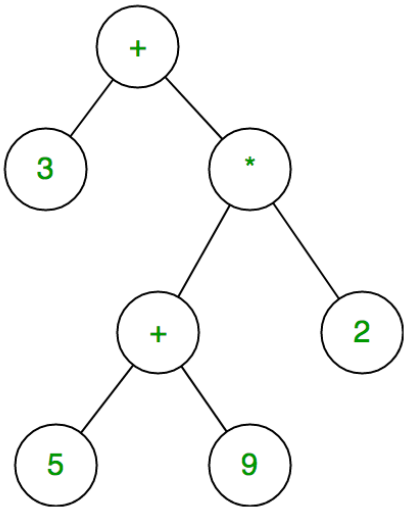
A skewed binary tree is a pathological/degenerate tree in which the tree is either dominated by the left nodes or the right nodes. Thus, there are two types of skewed binary tree: **left-skewed binary tree** and **right-skewed binary tree**.



Skewed Binary Tree

Expression Trees:

The expression tree is a tree used to represent the various expressions. The tree data structure is used to represent the expressional statements. In this tree, the internal node always denotes the operators and each leaf node corresponds to the operand so for example expression tree for 3 + ((5+9)*2) would be:



- The leaf nodes always denote the operands.
- The operations are always performed on these operands.
- The operator present in the depth of the tree is always at the highest priority.
- The operator, which is not much at the depth in the tree, is always at the lowest priority compared to the operators lying at the depth.
- The operand will always present at a depth of the tree; hence it is considered the **highest priority** among all the operators.
- In short, we can summarize it as the value present at a depth of the tree is at the highest priority compared with the other operators present at the top of the tree.
- The main use of these expression trees is that it is used to **evaluate, analyze** and **modify** the various expressions.
- It is also used to find out the associativity of each operator in the expression.

Tree Traversal:

Tree traversal is a popular form of graph traversal in the world of computer science and data structures, and it is a procedure of visiting each node of the tree. The sequence in which the nodes are visited is prefered to classify these traversals.

As we know, the tree is a non-linear data structure; hence its traversal is not the same as a linear data structure. In the case of linear data structures, we have

only a single way to visit each node, and that is to begin from the first element and traverse in a linear order. We have several ways to traverse a tree. In this article, we will learn more about those ways.

Tree traversal is a popular form of graph traversal in the world of computer science and data structures, and it is a procedure of visiting each node of the tree.

The sequence in which the nodes are visited is preferred to classify these traversals.

1. Preorder Traversal

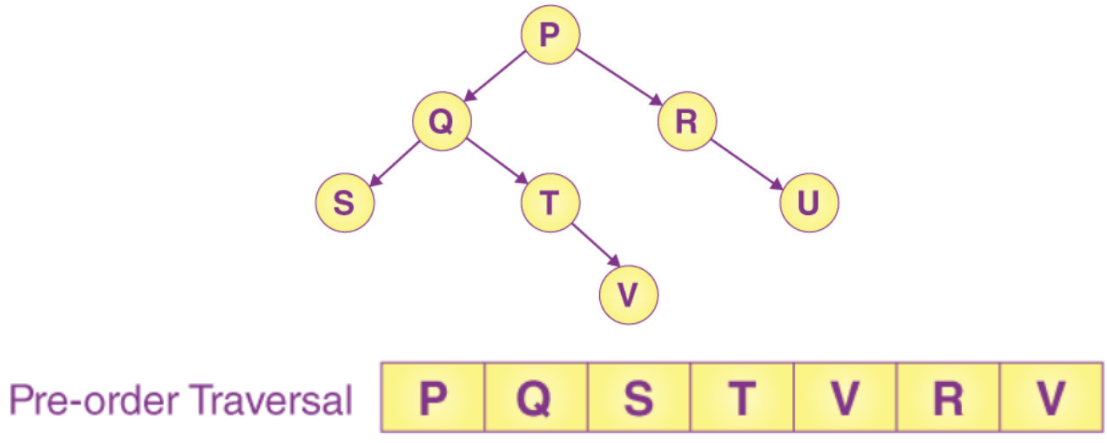
2. Inorder Traversal

3. Postorder Traversal

Preorder Traversal

In Preorder, we traverse from the root node to the left subtree and then we proceed to the right subtree.

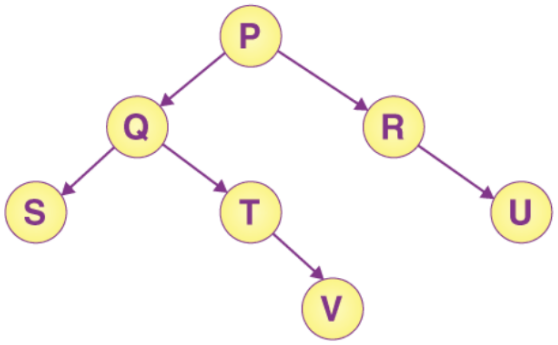
i) Visit Root node. ii) Visit Left sub tree. iii) Visit Right Sub tree



Inorder Traversal

In an inorder traversal, we rather process the left subtree, then move to the root node and then the right subtree.

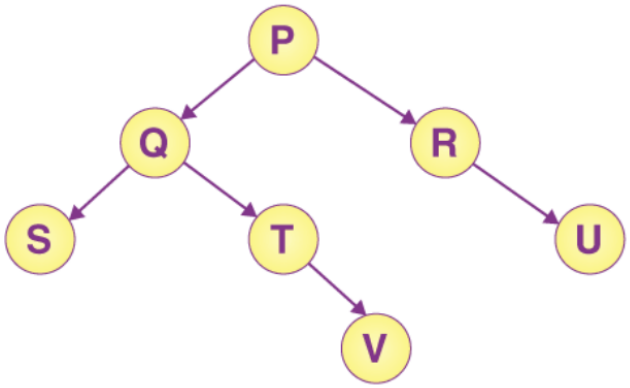
i) Visit Left sub tree. ii) Visit Root node. iii) Visit Right Sub tree



Postorder Traversal

In a postorder, we traverse from the left subtree to the right subtree and then proceed to the root node.

i) Visit Left sub tree. ii) Visit Root node. iii) Visit Right Sub tree



Recursive inorder traversal C function

```
Void inorder(Node temp)
{
if (temp!=NULL)
```

```
{
        inorder(temp->lptr)
        printf("%d",temp->data);
        inorder(temp->rptr)
}
```

Recursive postorder traversal C function

```
Void postorder(Node temp)
{
if (temp!=NULL)
{
        postorder(temp->lptr)
        postorder(temp->rptr
        printf("%d",temp->data);
}
}
```

Recursive preorder traversal C function

```
Void preorder(Node temp)
{
if (temp!=NULL)
{
        printf("%d",temp->data);
        preorder(temp->lptr)
        preorder(temp->rptr
}
}
```

Use of Expression tree

1. The main objective of using the expression trees is to make complex expressions and can be easily be evaluated using these expression trees.
2. It is also used to find out the associativity of each operator in the expression.
3. It is also used to solve the postfix, prefix, and infix expression evaluation.

To implement the expression tree and write its program, a stack data structure is used.