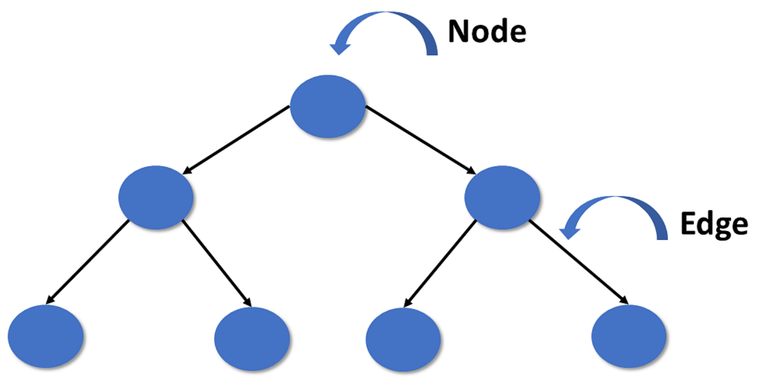
Introduction to Tree in Data Structure

The tree is a nonlinear hierarchical data structure and comprises a collection of entities known as nodes. It connects each node in the tree data structure using "edges", both directed and undirected. The image below represents the tree data structure. The blue-colored circles depict the nodes of the tree and the black lines connecting each node with another are called edges.



The Necessity for a Tree in Data Structures

Other data structures like arrays, linked-list, stacks, and queues are linear data structures, and all these data structures store data in sequential order. Time complexity increases with increasing data size to perform operations like insertion and deletion on these linear data structures. But it is not acceptable for today's world of computation.

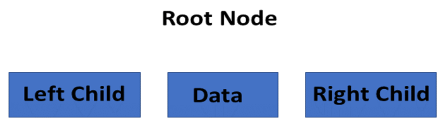
The non-linear structure of trees enhances the data storing, data accessing, and manipulation processes by employing advanced control methods traversal through it. You will learn about tree traversal in the upcoming section.

Tree terminologies

Tree Node

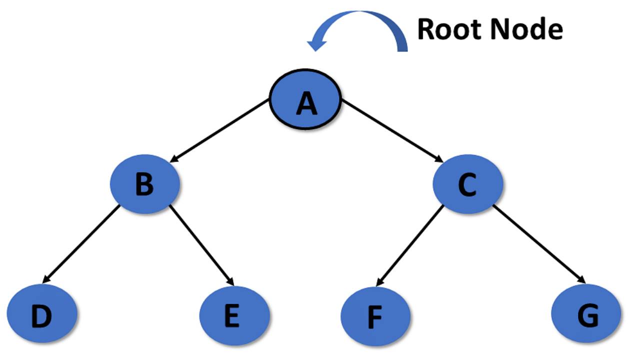
A node is a structure that contains a key or value and pointers in its child node in the tree data structure.
In the tree data structure, you can define the tree node as follows.

```
struct node
{
 int data;
 struct node *leftchild;
 struct node *rightchild;
}
```



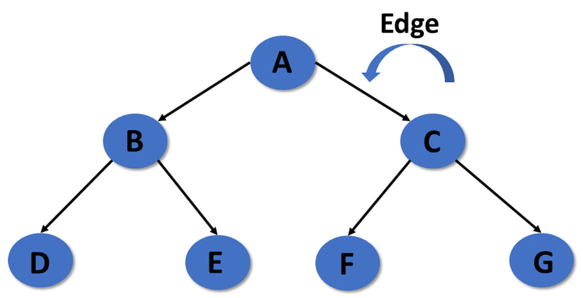
Root

- In a tree data structure, the root is the first node of the tree. The root node is the initial node of the tree in data structures.
- In the tree data structure, there must be only one root node.



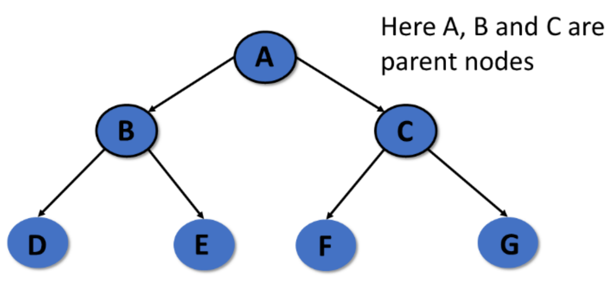
Edge

- In a tree in data structures, the connecting link of any two nodes is called the edge of the tree data structure.
- In the tree data structure, N number of nodes connecting with N -1 number of edges.



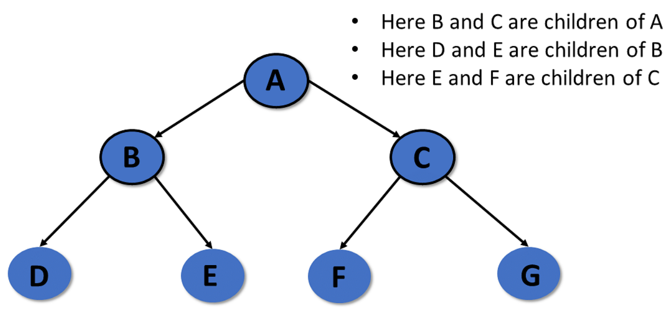
Parent

In the tree in data structures, the node that is the predecessor of any node is known as a parent node, or a node with a branch from itself to any other successive node is called the parent node.



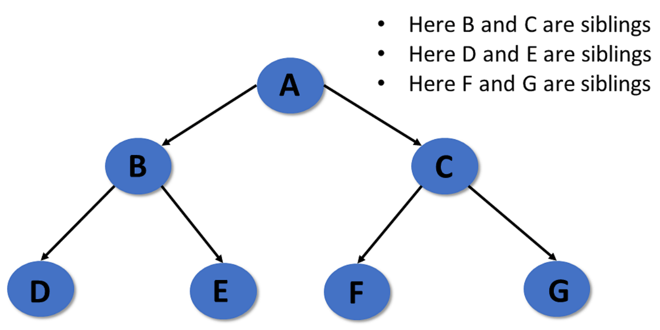
Child

- The node, a descendant of any node, is known as child nodes in data structures.
- In a tree, any number of parent nodes can have any number of child nodes.
- In a tree, every node except the root node is a child node.


Here B and C are children of A
Here D and E are children of B
Here E and F are children of C
A
B
C
D
E
F
G

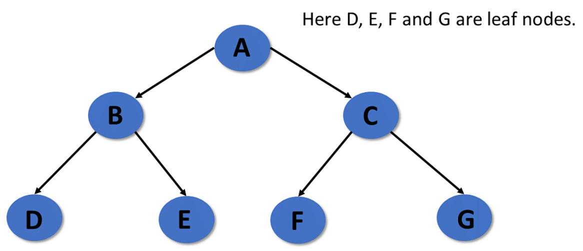
Siblings

In trees in the data structure, nodes that belong to the same parent are called siblings.


Here B and C are siblings
Here D and E are siblings
Here F and G are siblings
A
B
C
D
E
F
G

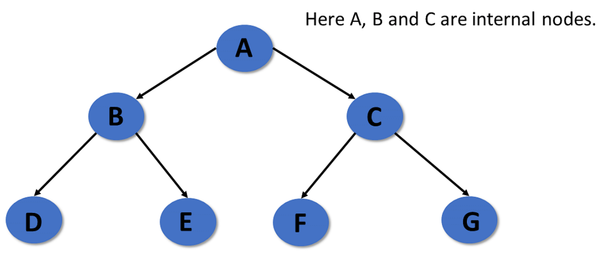
Leaf

- Trees in the data structure, the node with no child, is known as a leaf node.
- In trees, leaf nodes are also called external nodes or terminal nodes.



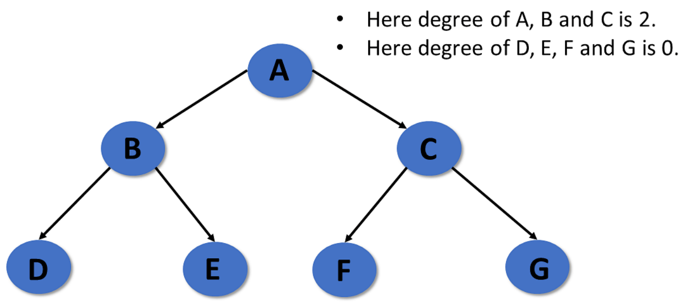
Internal nodes

- Trees in the data structure have at least one child node known as internal nodes.
- In trees, nodes other than leaf nodes are internal nodes.
- Sometimes root nodes are also called internal nodes if the tree has more than one node.



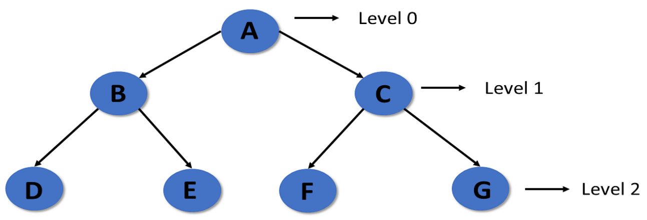
Degree

- In the tree data structure, the total number of children of a node is called the degree of the node.
- The highest degree of the node among all the nodes in a tree is called the Degree of Tree.


Here degree of A, B and C is 2.
Here degree of D, E, F and G is 0.
A
B
C
D
E
F
G

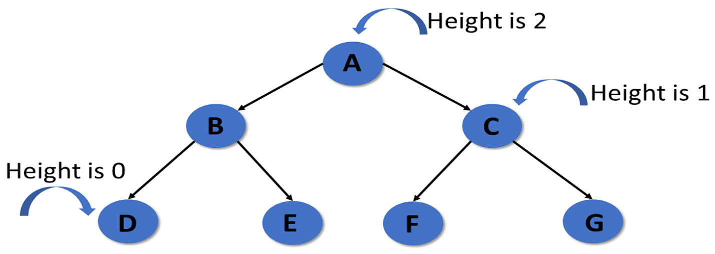
Level

In tree data structures, the root node is said to be at level 0, and the root node's children are at level 1, and the children of that node at level 1 will be level 2, and so on.



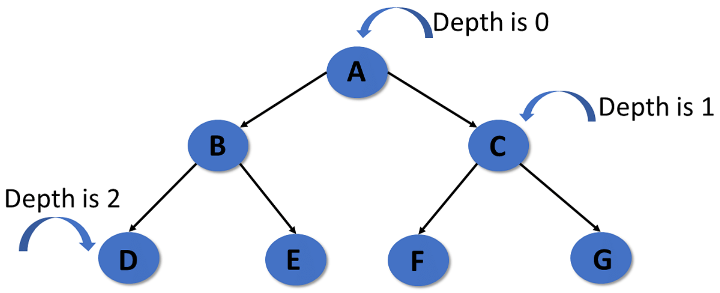
Height

- In a tree data structure, the number of edges from the leaf node to the particular node in the longest path is known as the height of that node.
- In the tree, the height of the root node is called "Height of Tree".
- The tree height of all leaf nodes is 0.



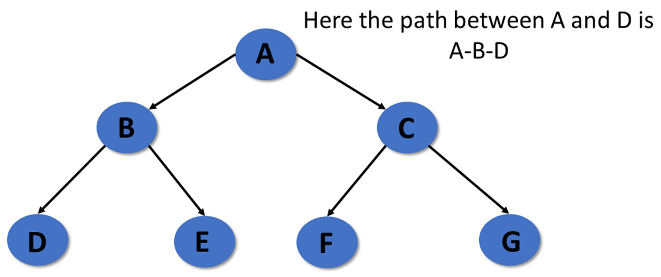
Depth

- In a tree, many edges from the root node to the particular node are called the depth of the tree.
- In the tree, the total number of edges from the root node to the leaf node in the longest path is known as "Depth of Tree".
- In the tree data structures, the depth of the root node is 0.



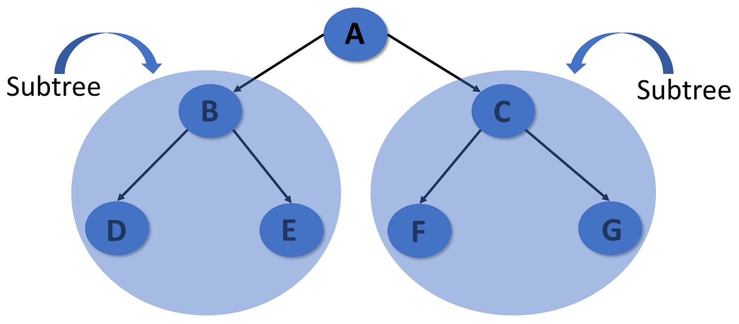
Path

- In the tree in data structures, the sequence of nodes and edges from one node to another node is called the path between those two nodes.
- The length of a path is the total number of nodes in a path.



Subtree

In the tree in data structures, each child from a node shapes a sub-tree recursively and every child in the tree will form a sub-tree on its parent node.



Types of Tree in Data Structures

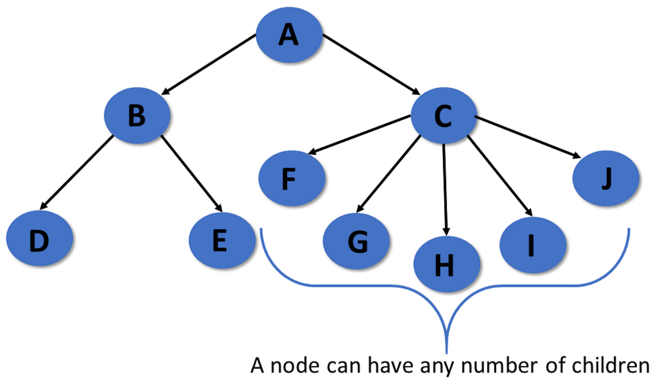
Here are the different kinds of tree in data structures:

General Tree

The general tree is the type of tree where there are no constraints on the hierarchical structure.

Properties

- The general tree follows all properties of the tree data structure.
- A node can have any number of nodes.

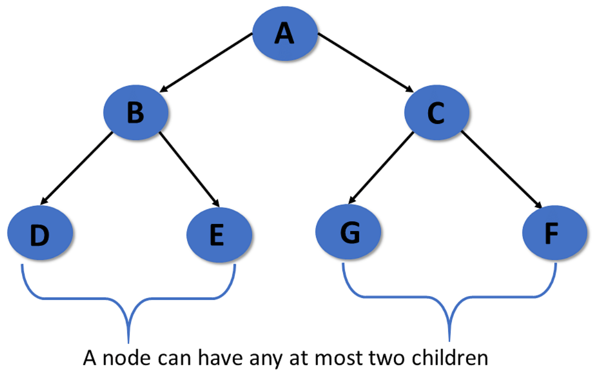


Binary Tree

A binary tree has the following properties:

Properties

- Follows all properties of the tree data structure.
- Binary trees can have at most two child nodes.
- These two children are called the left child and the right child.

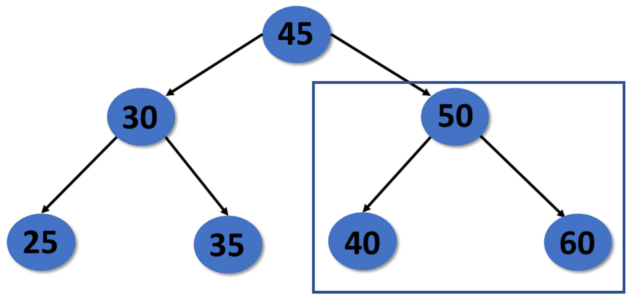


Binary Search Tree

A binary search tree is a type of tree that is a more constricted extension of a binary tree data structure.

Properties

- Follows all properties of the tree data structure.
- The binary search tree has a unique property known as the binary search property. This states that the value of a left child node of the tree should be less than or equal to the parent node value of the tree. And the value of the right child node should be greater than or equal to the parent value.



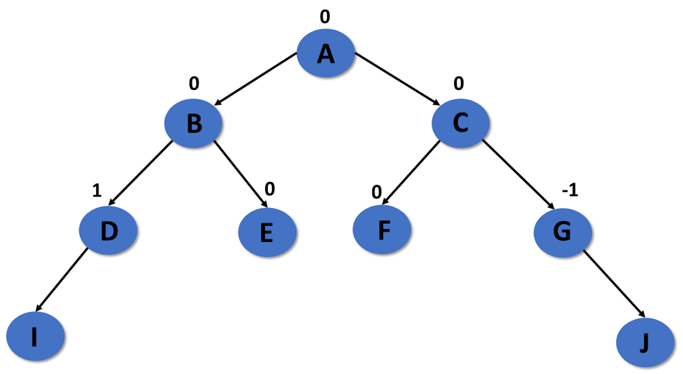
Left node value<= root node <= right node value

AVL Tree

An AVL tree is a type of tree that is a self-balancing binary search tree.

Properties

- Follows all properties of the tree data structure.
- Self-balancing.
- Each node stores a value called a balanced factor, which is the difference in the height of the left sub-tree and right sub-tree.
- All the nodes in the AVL tree must have a balance factor of -1, 0, and 1.

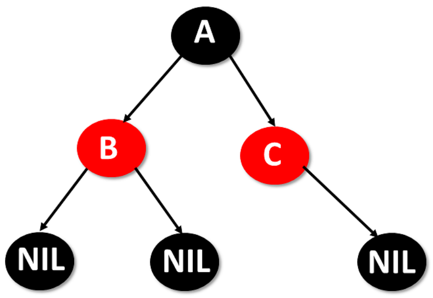


Red-Black Tree

- A red-black tree is a self-balancing binary search tree, where each node has either the color of red or black.
- The colors of the nodes are used to make sure that the tree remains approximately balanced during insertion and deletion.

Properties

- Follow all properties of binary tree data structure.
- Self-balancing.
- Each node is either red or black.
- The root and leaves nodes are black.
- If the node is red, then both children are black.
- Every path from a given node to any of its nodes must go through the same number of black nodes.



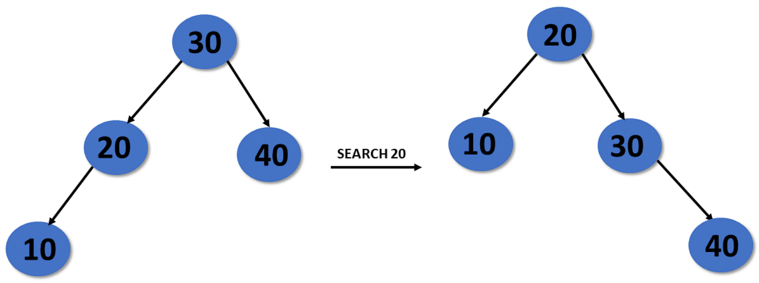
Splay Tree

A splay tree is a self-balancing binary search tree.

Properties

- Follows properties of binary tree data structure.
- Self-balancing.
- Recently accessed elements are quicker to access again.

After you perform operations such as insertion and deletion, the splay tree acts, which is called splaying. Here it rearranges the tree so that the particular elements are placed at the root of the tree.

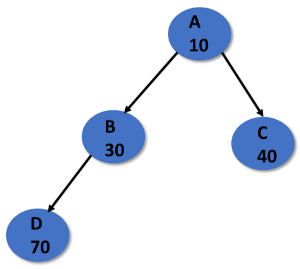


Treap Tree

The Treap tree is made up of a tree, and the heap is a binary search tree.

Properties

- Each node has two values: a key and a priority.
- Follows a binary search tree property.
- Priority of the treap tree follows the heap property.



Tree Traversal

Traversal of the tree in data structures is a process of visiting each node and prints their value. There are three ways to traverse tree data structure.

- Pre-order Traversal
- In-Order Traversal
- Post-order Traversal

In-Order Traversal

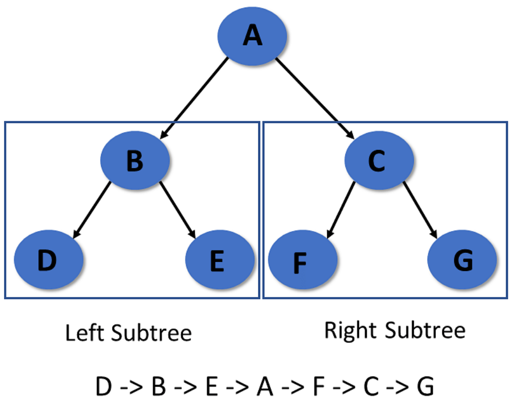
In the in-order traversal, the left subtree is visited first, then the root, and later the right subtree.

Algorithm:

Step 1- Recursively traverse the left subtree

Step 2- Visit root node

Step 3- Recursively traverse right subtree



Pre-Order Traversal

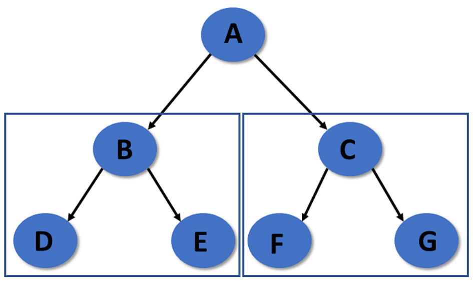
In pre-order traversal, it visits the root node first, then the left subtree, and lastly right subtree.

Algorithm:

Step 1- Visit root node

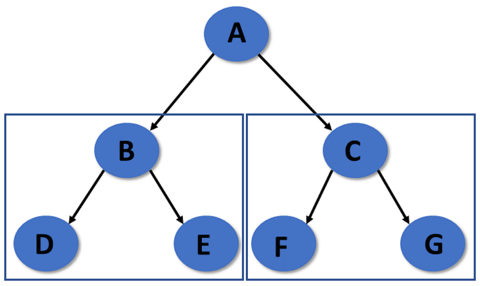
Step 2- Recursively traverse the left subtree

Step 3- Recursively traverse right subtree



A -> B -> D -> E -> C -> F -> G

Post-Order Traversal
It visits the left subtree first in post-order traversal, then the right subtree, and finally the root node.
Algorithm:
Step 1- Recursively traverse the left subtree
Step 2- Visit root node
Step 3- Recursively traverse right subtree



D -> E -> B -> F -> G -> C -> A

Code:
```
#include <stdio.h>
#include <stdlib.h>
struct node {
  int data;
  struct node* left;
  struct node* right;
};
void inorder(struct node* root) {
  if (root == NULL) return;
  inorder(root->left);
  printf("%d ->", root->data);
  inorder(root->right);
}
void preorder(struct node* root) {
  if (root == NULL) return;
  printf("%d ->", root->data);
  preorder(root->left);
  preorder(root->right);
}
void postorder(struct node* root) {
  if (root == NULL) return;
  postorder(root->left);
  postorder(root->right);
  printf("%d ->", root->data);
}
struct node* createNode(item) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->data = item;
  newNode->left = NULL;
  newNode->right = NULL;
  return newNode;
}
struct node* insertLeft(struct node* root, int item) {
  root->left = createNode(item);
  return root->left;
}
struct node* insertRight(struct node* root, int item) {
  root->right = createNode(item);
  return root->right;
}
int main() {
  struct node* root = createNode(7);
  insertLeft(root, 12);
  insertRight(root, 29);
  insertLeft(root->left, 15);
  insertRight(root->left, 26);
  printf("Inorder  \n");
  inorder(root);
  printf("\nPreorder \n");
  preorder(root);
  printf("\nPostorder  \n");
  postorder(root);
}
```

Application of Tree in Data Structures

- Binary Search Tree (BST) is used to check whether elements present or not.
- Heap is a type of tree that is used to heap sort.
- Tries are the modified version of the tree used in modem routing to information of the router.
- The widespread database uses B-tree.
- Compilers use syntax trees to check every syntax of a program.