Practical – 9
Implementation of Tree and Searching

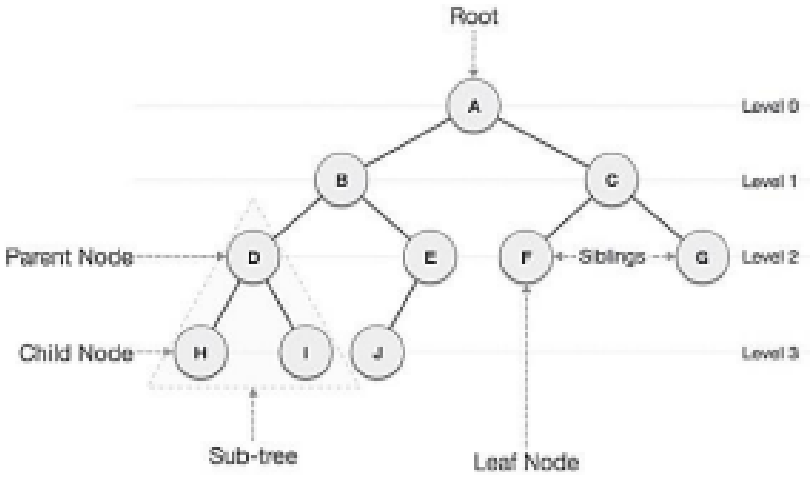
Tree

Tree represents the nodes connected by edges. We will discuss binary trees or binary search trees specifically.

The tree is a non-linear data structure, i.e. The elements are not stored sequentially, and in the tree, they are present in levels. A binary tree is a data structure in which every node can have a maximum of two children nodes. Children nodes are labeled as right and left child. Each node in the binary tree contains a value and two pointers pointing to the children. The topmost node of the binary tree is called the root node and the bottom-most nodes of the binary tree are called leaf nodes. The height of a binary tree is calculated by taking the longest path between the root and the leaf node.

Binary Tree is a special data structure used for data storage purposes. A binary tree has a special condition that each node can have a maximum of two children. A binary tree has the benefits of both an ordered array and a linked list as search is as quick as in a sorted array and insertion or deletion operations are as fast as in linked list.

In the i'th level, the maximum number of nodes can be 2 raised to the power i, and the minimum number of nodes in a binary tree of height H is H+1.



Tree Node

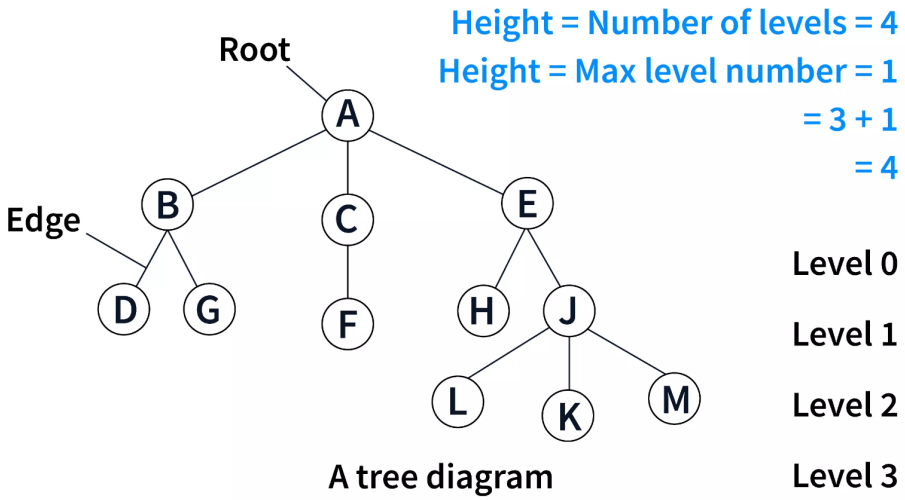
```
struct node {
     int data;
     struct node *leftChild;
     struct node *rightChild;
};
```

The binary trees are implemented using pointers in C, usually, we create a structure that contains a data variable that is used to store the value of that node and two pointer

variables(named left and right), the left pointer is used to point to the left child similarly, the right pointer is used to point to the right child of the node.

The binary tree in C can also be implemented using the array data structure. If P is the index of the parent element then the left child will be stored in their index (2p)+1, and the right child will be stored in the index (2p)+2.

The tree is a non-linear data structure in C that contains nodes that are not connected linearly, rather they are connected in a hierarchical manner. The nodes are present at different levels and are connected by edges. Between the set of two connected nodes, the one which is at the upper level is considered a parent, and the lower one is considered a child node.

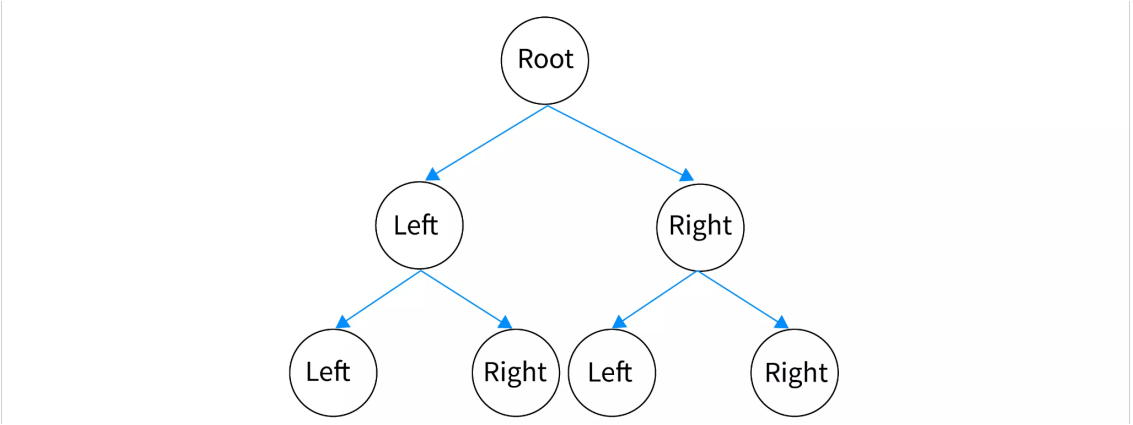


For above tree

Number of level = 4

Max level number = 3

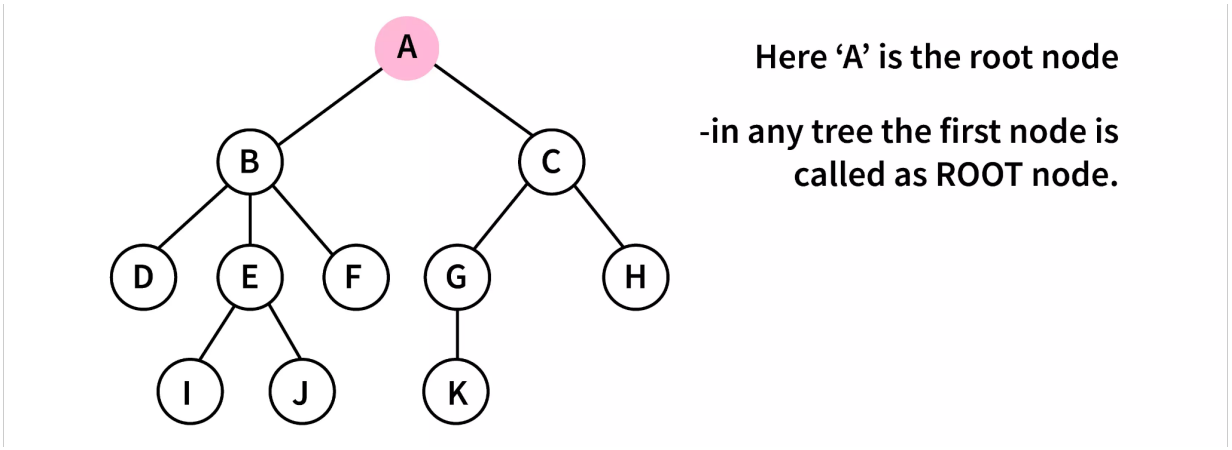
In the above image of the tree, A is the root node. B, C, and E are the children of node A, and they have 2, 1, and 2 children respectively. Here, D, G, F, H, L, K, and M are the leaf nodes. A binary tree in C is a special case of the tree, which can have only two children nodes (left child and right child).



Terminologies

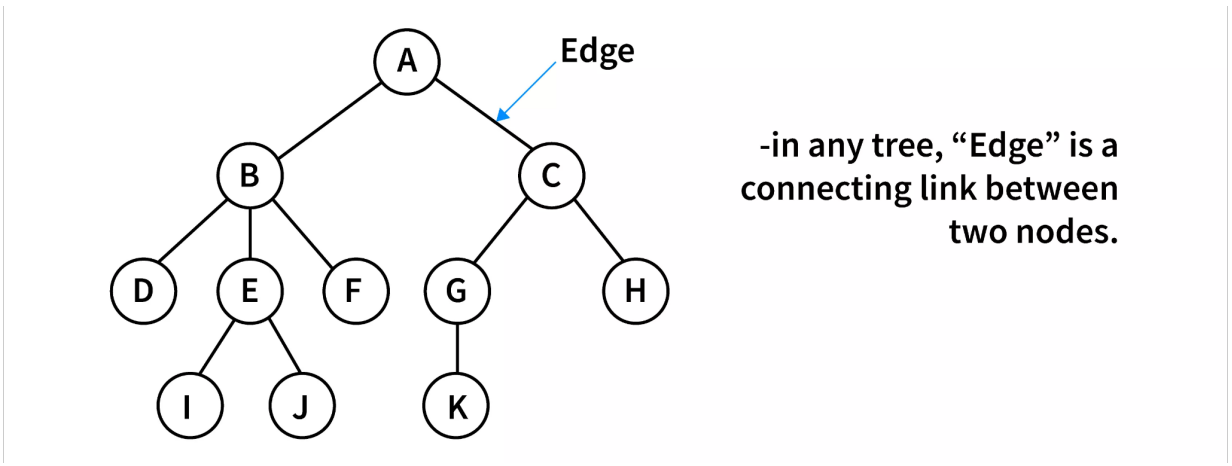
Root :

The first node of the binary tree is known as the root node of the binary tree. There cannot be any tree without a root node as it is the origin of the tree and the root node does not have any parent element. We cannot have more than one root node in a tree.



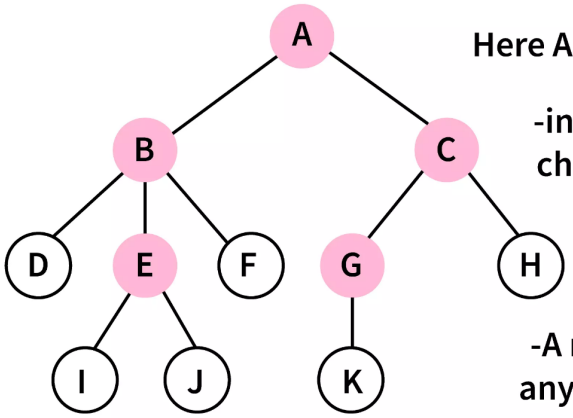
Edge :

The connecting link between the two nodes is known as the edges of the tree it is also referred to as the branch of a tree. If there are n nodes in a binary tree then there will be a total of n-1 edges.



Parent :

A node that has a connecting link to another node in the level below is known as a parent node. In a more formal term, a node that is a predecessor of another node is called a parent node.



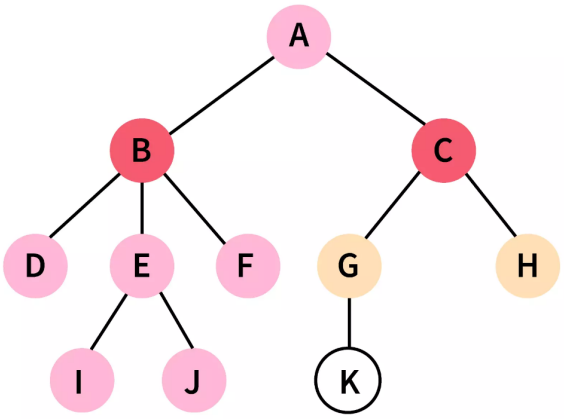
Here A, B, C, E & G are parent nodes

-in any tree the node which has child/children is called 'parent'

-A node which is predecessor of any other node is called 'parent'

Child :

Any node which is connected to a node that is one level above it, is known as the child node. In formal terms, it can be said that every note that is a descendant of any node is known as a child node. Every not present in the tree can be considered as a child node except the root node.

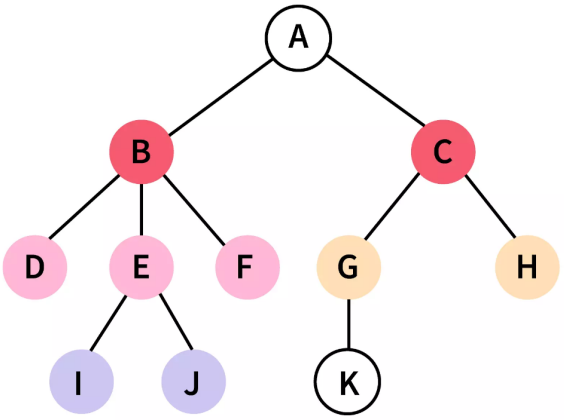


Here B & C are children of A
Here G & H are children of C
Here K is child of G

-descendant of any node is called as CHILD NODE

Siblings :

The children node that shares the same parent node are referred to as siblings.



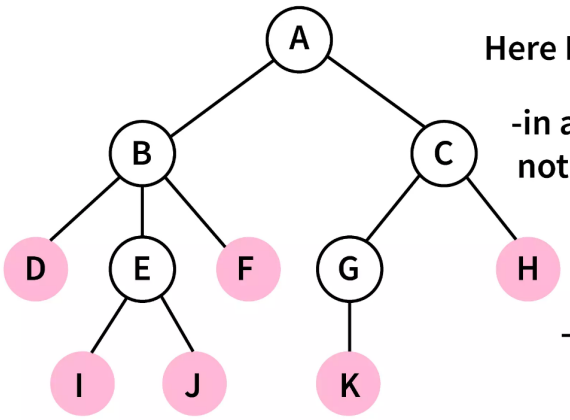
Here B & C are Siblings
Here D, E & F are Siblings
Here G & H are Siblings
Here I & J are Siblings

-in any tree the nodes which has same Parent are called 'Siblings"

-The children of a Parent are Called 'Siblings'

Leaf :

A node with no child is known as a leaf node. It is the terminal node of the tree.



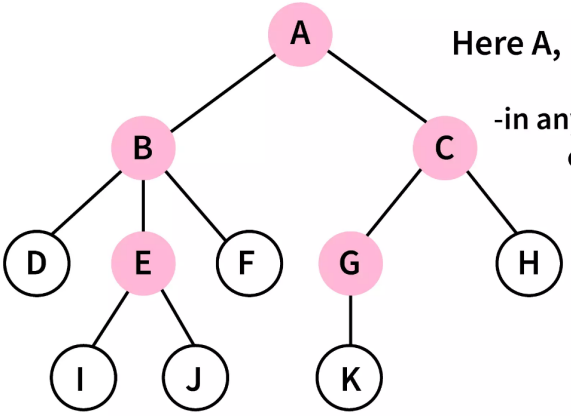
Here D, I, J, F, K & H are Leaf nodes

-in any tree the node which does not have children is called 'Leaf'

-A node without successors is called a 'lead' note

Internal Node :

The nodes that have at least one child node is known as an internal node. Every node (even the root node) is an internal node except the leaf nodes.



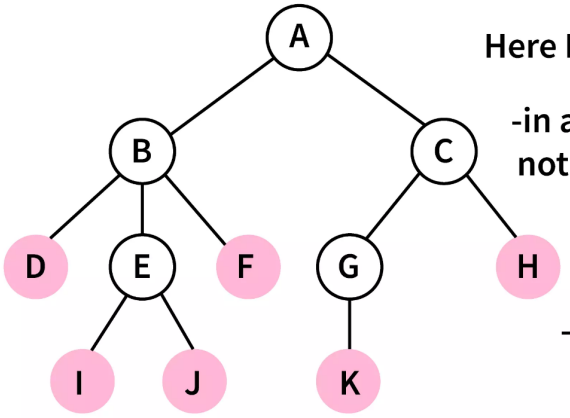
Here A, B, C, E & G are internal nodes

-in any tree the node which has atleast one child is called 'internal' node

-Every non-leaf node is called as 'internal' node

External Node :

A node that doesn't have any child node is considered an external node.



Here D, I, J, F, K & H are Leaf nodes

-in any tree the node which does not have children is called 'Leaf'

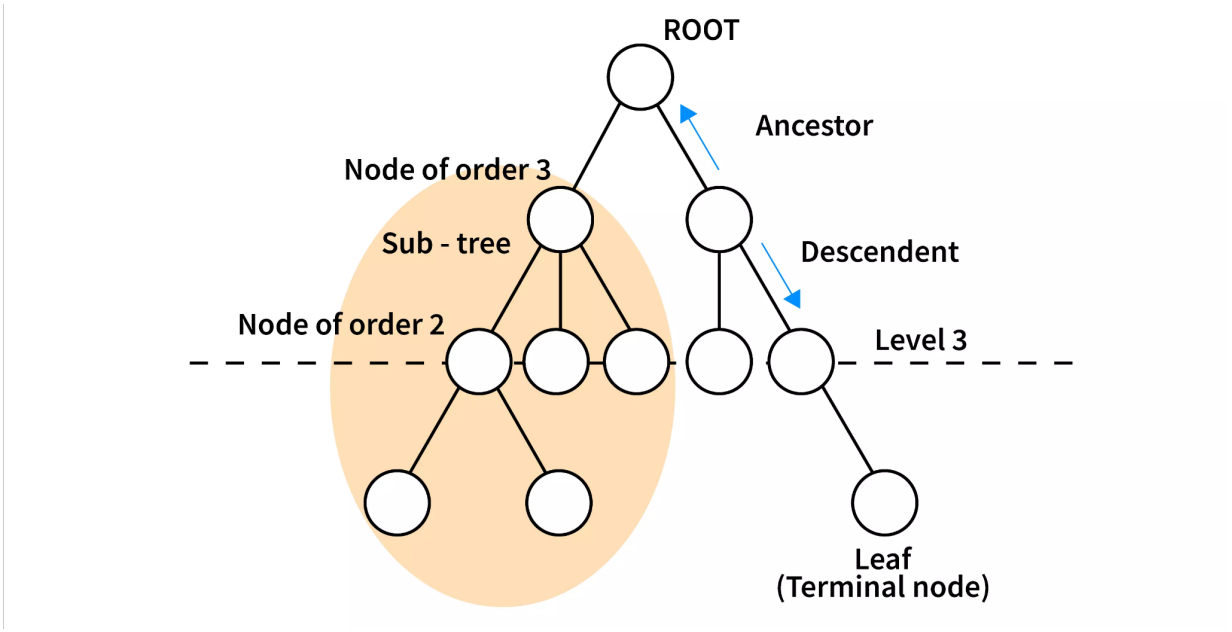
-A node without successors is called a 'lead' note

Ancestors :

Ancestors of a node, are the nodes in the path from the root to the current node, including the root node, and excluding the current node are ancestors of the current node.

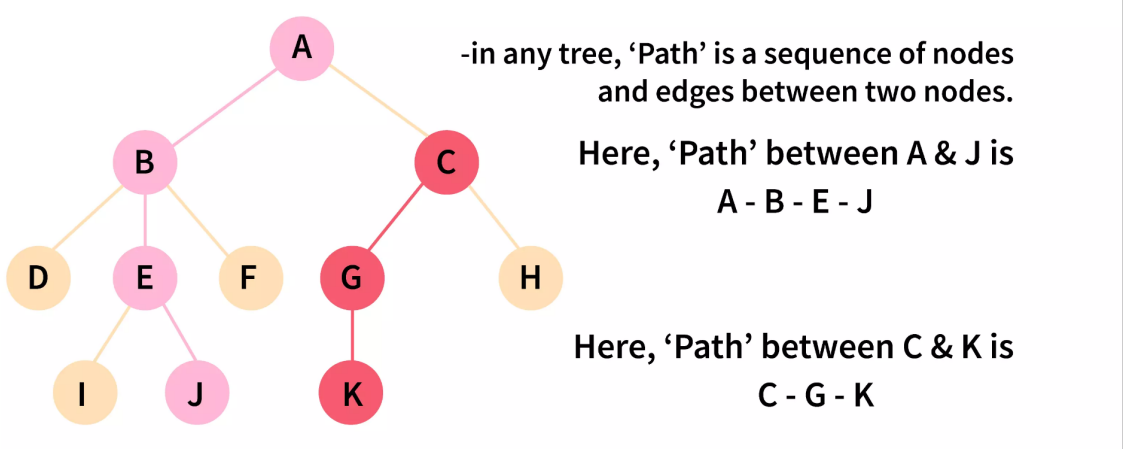
Descendants :

Descendants of a node, are the nodes that are below its level and are also connected to it, directly or via nodes. All the children, their children, and so on are the descendants of the current node.



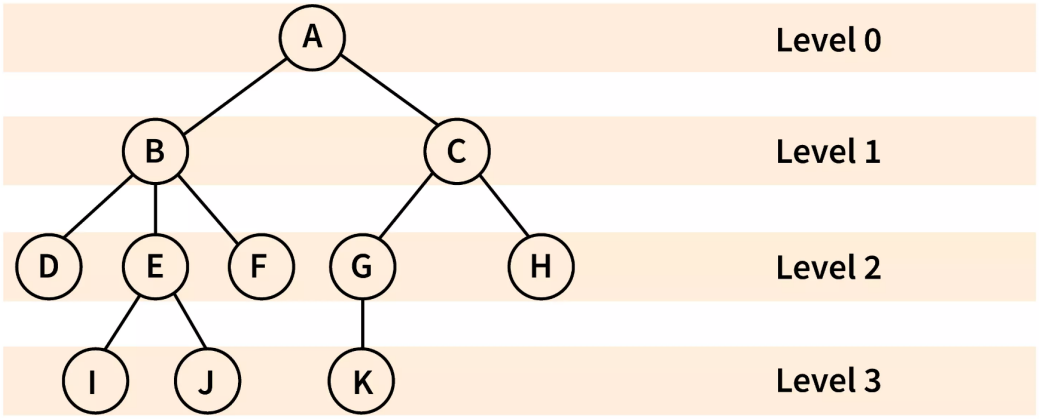
Path :

A path between any two nodes of a tree is the sequence of connected nodes along with the edges between the two nodes. And the path length is the number of nodes present between two nodes.



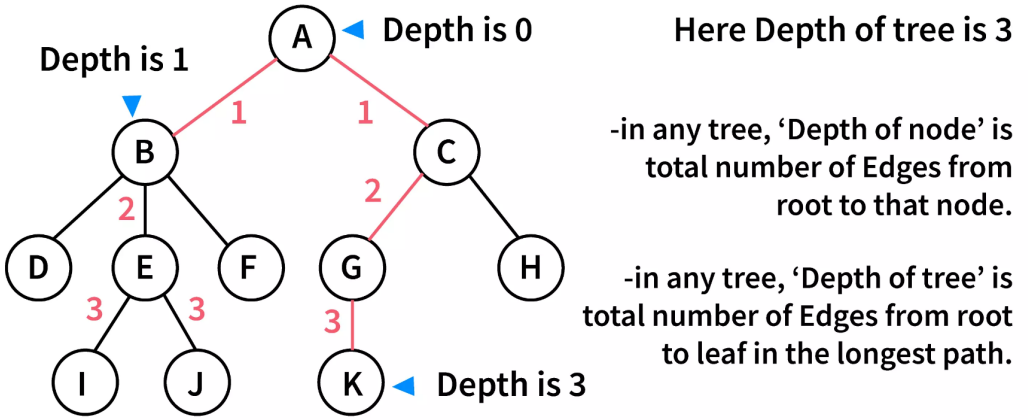
Level :

The level in the tree represents the number of steps from the top/root. The root node of the tree is said to be at level 0, and as we go downwards the level increases



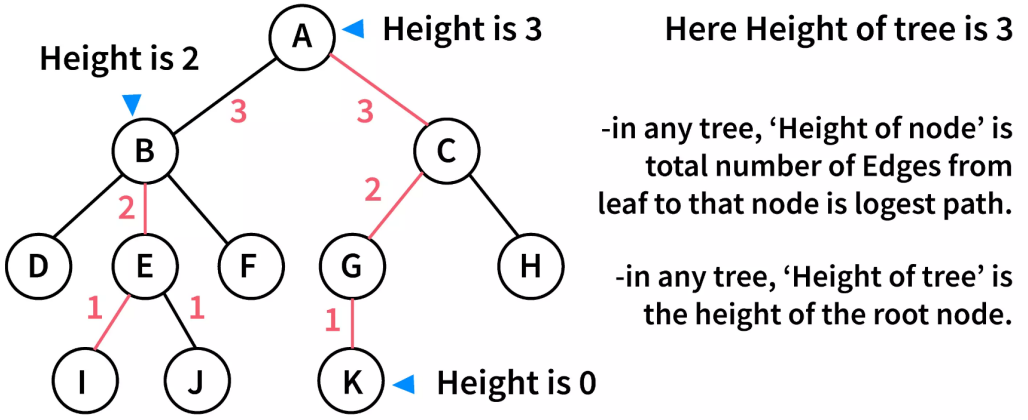
Depth :

The depth of any node in the tree is the length of the path from the root node to that particular node, i.e. The number of nodes between that particular node and the root node.



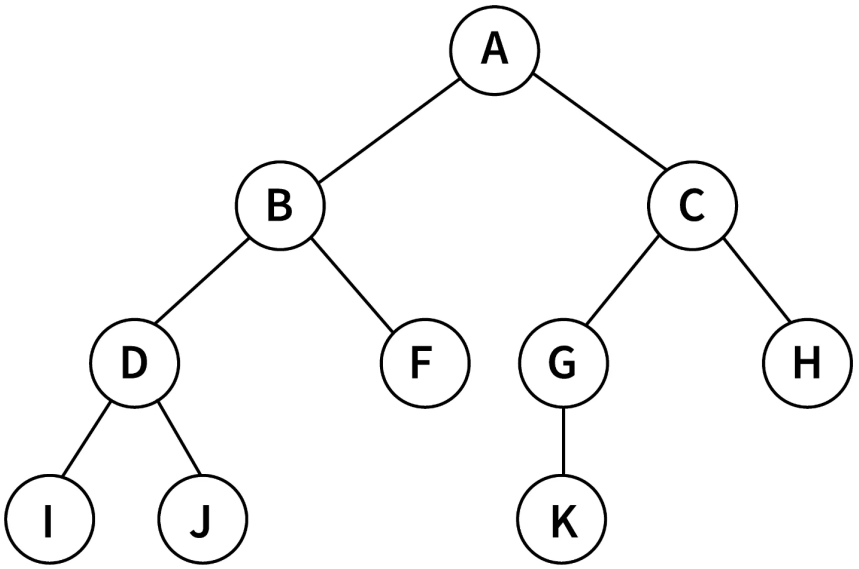
Height :

The height of any node is given by the maximum number of edges from the leaf node to that particular node.



Representation of Binary Tree in C

A binary tree can be represented using a linked list and also using an array.



Array Representation :

The binary tree can be represented using an array of size 2n+1 if the depth of the binary tree is n. If the parent element is at the index p, Then the left child will be stored in the index (2p)+1, and the right child will be stored in the index (2p)+2.

The array representation of the above binary tree is :

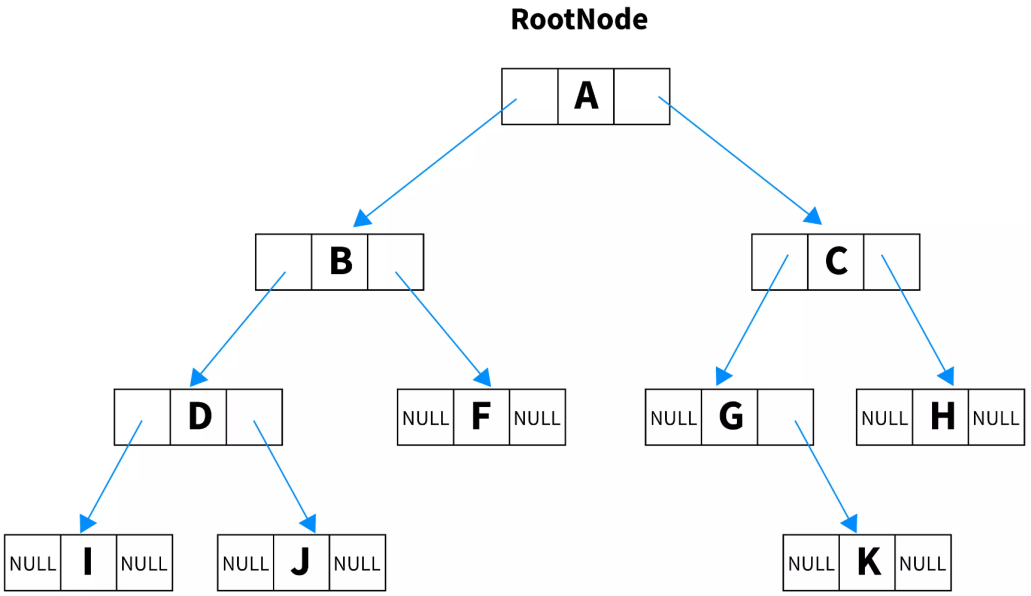
As in the above binary tree, A was the root node, so it will be stored in the 0th index. The left child of A will be stored in the 2(0)+1 that is equal to the 1st location. So, B is stored in index 1. And, similarly, the right child of A will be stored in the 2(0)+2 index. For every node, the left and right child will be stored accordingly.

A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

Linked List Representation :

For the linked list representation we will be using the doubly linked list which has two pointers so that we can point to the left and right children of a node of a binary tree. NULL is given to the pointer as the address when no child is connected to it.

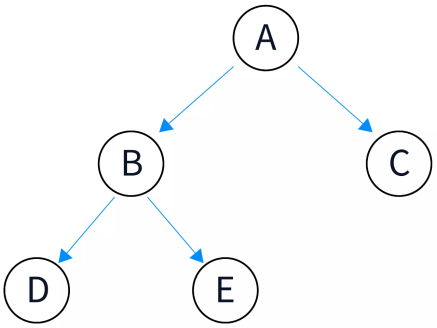
The Linked list representation of the above binary tree is :



Types of Binary Trees in C

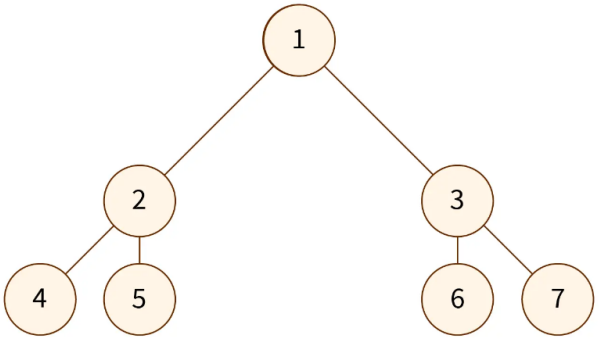
Full Binary Tree

In a full binary tree, every node has either 0 or two children. It is also referred to as a strict binary tree.



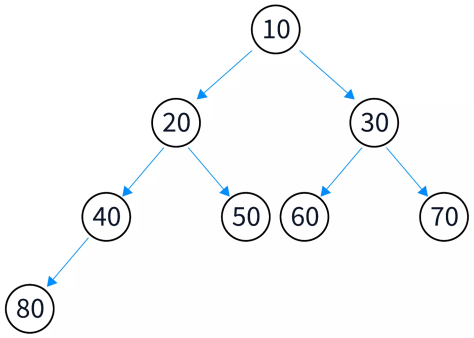
Perfect Binary Tree

As the name suggests, a perfect binary tree is a type of binary tree in which all the nodes(internal nodes) have two children and all the leaf nodes are at the same level.



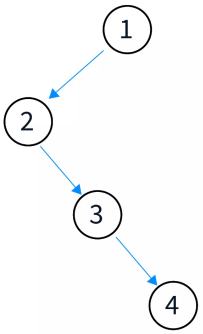
Complete Binary Tree

In a complete binary tree, all the levels except the last level(that contains the leaf nodes) should be full. In this type of binary tree, the node should be inserted to the left first, i.e. It should be left-leaning (it is not mandatory to have a right sibling for the node).

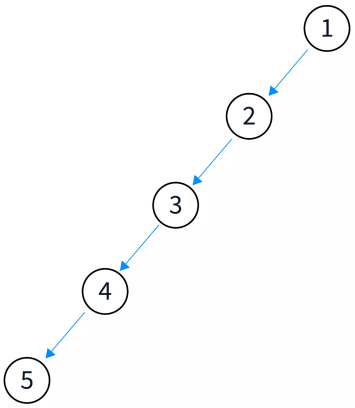


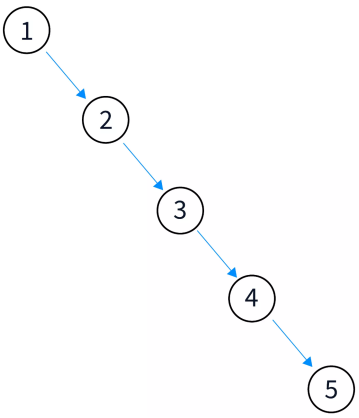
Degenerate or Pathological Tree

In this type of tree, every node has only one child that can be either on the left or on the right.



If nodes only have children towards the left, then it is called a left-skewed binary tree. If nodes only have children towards the right, then it is called a right-skewed binary tree .





Balanced Binary Tree

It is a type of binary tree in which for every node the height of the left subtree and the height of the right subtree defer by zero or at most one.

Implementation of Binary Tree in C

The binary tree is implemented using the structure. Each node will contain three elements, the data variable, and 2 pointer variables. We will use user-defined datatype (structure) to create the nodes. The node will be pointing to NULL if it doesn't point to any children.

Declaration of a Binary Tree in C

To declare the nodes of the binary tree, we will create a structure containing one data element and two pointers of the name left and right.

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

Create a New Node of Binary Tree in C

To create a new node of the binary tree we will make create a new function that will take the value and return the pointer to the new node that is created with that value.

```
struct node* create(value) {
  struct node* newNode = malloc(sizeof(struct node));
  newNode->data = value;
  newNode->left = NULL;
  newNode->right = NULL;

  return newNode;
}
```

Inserting the Left and/or the Right Child of a Node

This function will take two inputs, first the value that needs to be inserted and second the pointer of the node on which the left or the right child will be attached. To insert into the left side of the root node, we will simply call the create function with the value of the node as a parameter. So that, a new node will be created, and as that function returns a pointer to the newly created node we will assign that pointer to the left pointer of the root node that is being taken as input.

Similarly, insertion on the right side will call the create function with the given value and the returned pointer will be assigned to the write pointer of the root node.

```
// Insert on the left of the node
struct node* insertLeft(struct node* root, int value) {
  root->left = create(value);
  return root->left;
}

// Insert on the right of the node
struct node* insertRight(struct node* root, int value) {
  root->right = create(value);
  return root->right;
}
```

Traversal of Binary Tree

Binary tree traversing is a process of accessing every node of the tree and exactly once. A tree is defined in a recursive manner. Binary tree traversal is also defined recursively. There are three techniques of traversal:

- Preorder Traversal
- Postorder Traversal
- Inorder Traversal

Preorder Traversal

Algorithm for preorder traversal

Step 1 : Start from the Root.
Step 2 : Then, go to the Left Subtree.
Step 3 : Then, go to the Right Subtree.

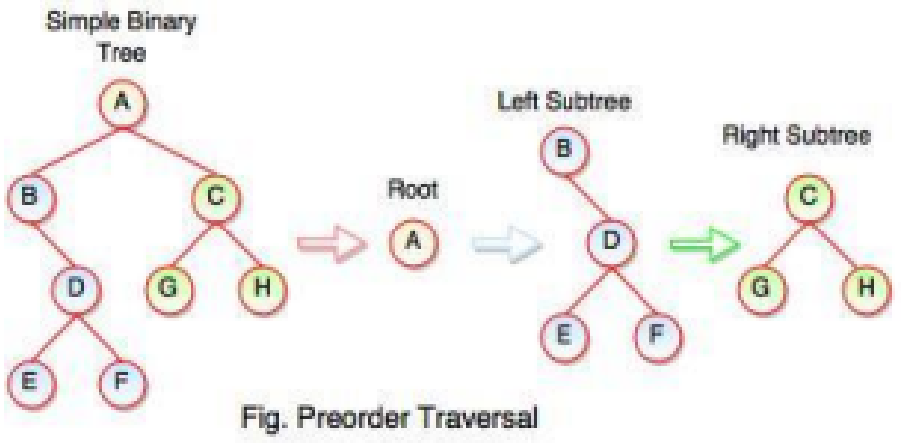


Fig. Preorder Traversal

Following steps can be defined the flow of preorder traversal:

Step 1 : A + B (B + Preorder on D (D + Preorder on E and F)) + C (C + Preorder on G and H)
Step 2 : A + B + D (E + F) + C (G + H)
Step 3 : A + B + D + E + F + C + G + H

Preorder Traversal : A B C D E F G H

```
void preorderTraversal(struct node* root) {
  if (root == NULL) return;
  printf("%d ->", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}
```

Postorder Traversal

Algorithm
Step 1 : Start from the Left Subtree (Last Leaf).
Step 2 : Then, go to the Right Subtree.
Step 3 : Then, go to the Root.

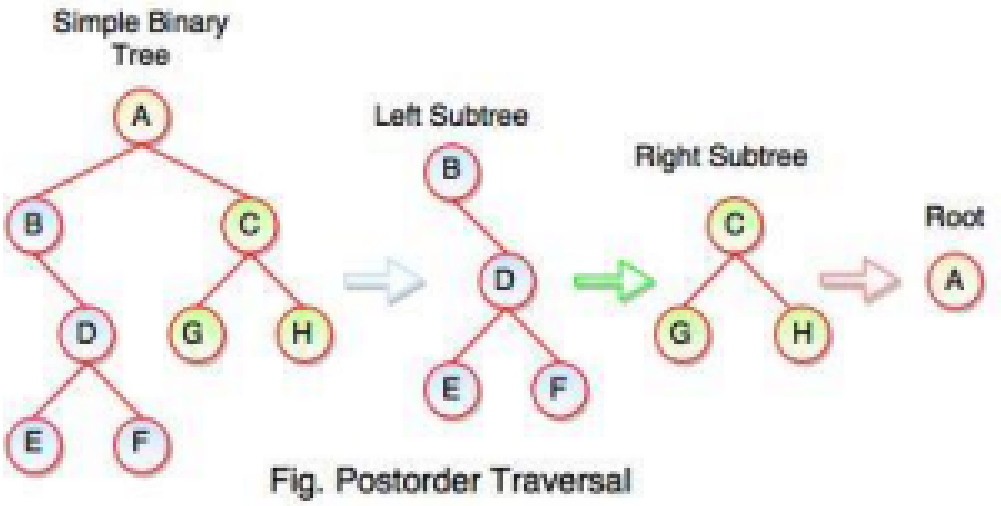


Fig. Postorder Traversal

Following steps can be defined the flow of postorder traversal:

Step 1 : As we know, preorder traversal starts from left subtree (last leaf) ((Postorder on E + Postorder on F) + D + B)) + ((Postorder on G + Postorder on H) + C) + (Root A)
Step 2 : (E + F) + D + B + (G + H) + C + A
Step 3 : E + F + D + B + G + H + C + A

Postorder Traversal : E F D B G H C A

```
void preorderTraversal(struct node* root) {
  if (root == NULL) return;
  printf("%d ->", root->item);
  preorderTraversal(root->left);
  preorderTraversal(root->right);
}
```

Inorder Traversal

Algorithm
Step 1 : Start from the Left Subtree.
Step 2 : Then, visit the Root.
Step 3 : Then, go to the Right Subtree.

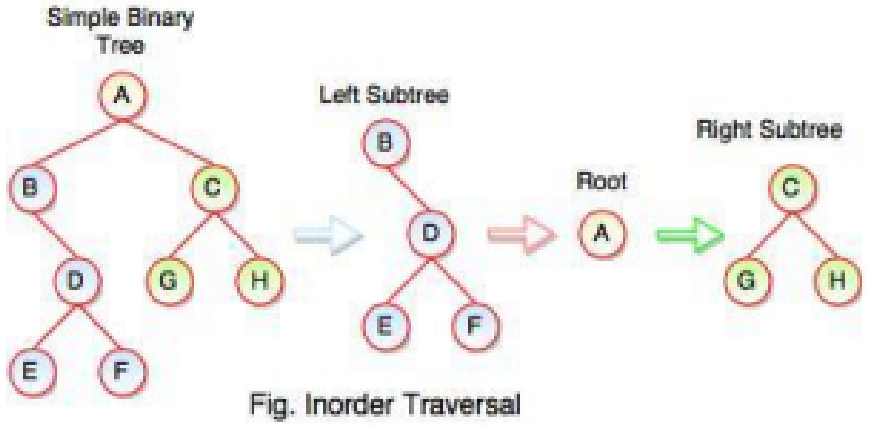


Fig. Inorder Traversal

Following steps can be defined the flow of inorder traversal:

Step 1 : B + (Inorder on E) + D + (Inorder on F) + (Root A) + (Inorder on G) + C (Inorder on H)
Step 2 : B + (E) + D + (F) + A + G + C + H
Step 3 : B + E + D + F + A + G + C + H

Inorder Traversal : B E D F A G C H

```
void inorderTraversal(struct node* root) {
  if (root == NULL) return;
  inorderTraversal(root->left);
  printf("%d ->", root->item);
  inorderTraversal(root->right);
}
```

Searching

Searching is an operation or a technique that helps finds the place of a given element or value in the list. Any search is said to be successful or unsuccessful depending upon whether the element that is being searched is found or not. Some of the standard searching technique that is being followed in the data structure is listed below:

- Linear Search or Sequential Search
- Binary Search

Binary Search

```
# Input: Sorted Array A, integer key
# Output: first index of key in A,
# or -1 if not found
Algorithm: Binary_Search (A, left, right)
left = 0, right = n-1
while left < right
 middle = index halfway between left, right
 if A[middle] matches key
 return middle
 else if key less than A[middle]
 right = middle -1
 else
 left = middle + 1
return -1
```

Exercise

1. Write a program to do the following operations.
    - Create a Binary Tree by collecting information from users.
    - Create a Binary Search Tree by collecting information from users.
    - Traverse the created trees using
        - preorder
        - postorder
        - inorder
        - levelorder
    - Search Element in Binary Search Tree
    - Find Internal Nodes, External Nodes, Total Nodes and Height of Tree
2. Write a program to do the following operations.
    - Create an array from user input.
    - Search Element in an array using linear search - prints iteration done to find the element
    - Search Element in an array using binary search - prints iteration done to find the element