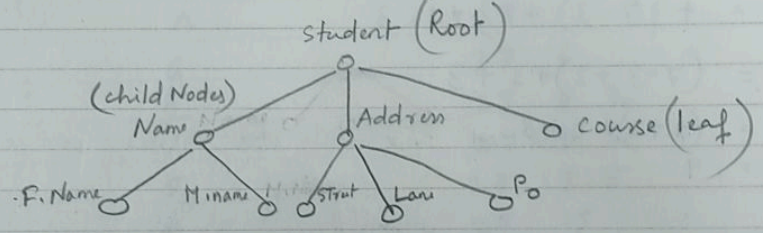
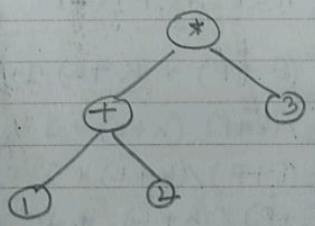
Tree

DATE

Tree is a non linear data structure.
Data in a tree is stored in a hierarchy form.
Examples of tree application :-

- Represent algebraic formulas.
- store data in hierarchy form.
  eg. student Record / Information chart.
- Artificial intelligence - information is accessed based on certain decision which is stored in a tree.





algebraic formulas : (1+2) * 3.

Natural

A tree is a collection of nodes and edges that connect the nodes.

- The collection can be empty.
- If not empty, a tree consists of a root, and zero or more nonempty subtrees.
- Any two vertices in a tree must have only one path between them or else it's not a tree.
- Trees are hierarchical
  - Has parent-child relationship between two nodes.
  - Has ancestor-descendent relationships among nodes

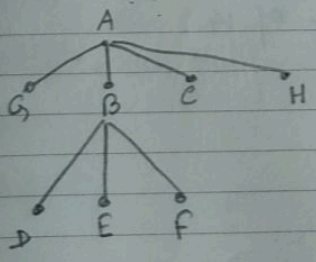
General tree :

→ A General tree is a set of one or more nodes that is partitioned into : i) the root
ii) sets that are general trees, called subtrees.

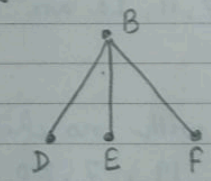
→ Each node in general tree can have an unlimited children

Subtree of a tree : Any node and its descendents

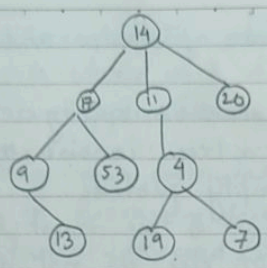
Tree Terminology



A general tree



A subtree of the tree in general tree



Root : The only node in the tree with no parent.
A tree has only one root.
Root : 14

child and Parent :
→ Every node except the root has one parent.
→ Parent of node n
- The node directly above node n in the tree.
- 14 is parent to 17, 11, 20.

→ A node can have an arbitrary number of children
→ child of node n.
- A node directly below node n in the tree.
- 17, 11, 20 are children of 14

Leaves :
- Nodes with no children
- 13, 53, 19, 7, 20.

Sibling :

- nodes with the same parent
- 17, 11, 20 are sib-lings
- 19, and 7 are siblings.

Subtree of node n :

- A tree that consists of a child (if any) of node n and the child's descendants

Binary Tree

DATE

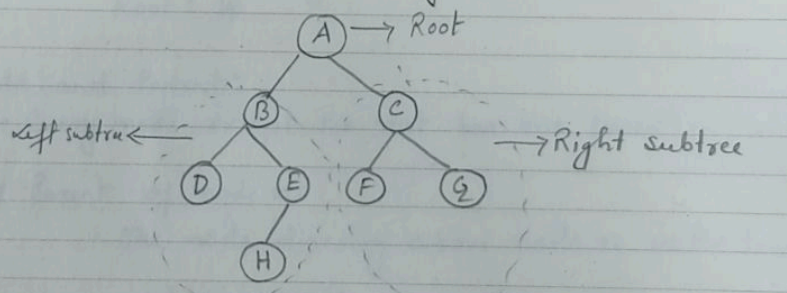
A binary tree is a finite set of nodes.

① It is either empty or

② It consists a node called root with two disjoint binary trees called left subtree and right subtree

Representation of Binary tree:

We use graph to represent tree in which every node will be represented by circle and a line from one node to other is called edge.



The left and right subtree also represent binary tree.
Each element of tree is called a node of the tree.
Here, 'A' is beginning node of the binary tree.
Hence, A is root.
It has left successor 'B' and right successor 'C'.
These are also called left and right child of father.

The level of every node in binary tree can be defined as -

(i) Root of tree is defined as level zero.
(ii) Level no. of other nodes is one more than level no. of its father node. Here A has level 0. and B and C have level 1.

The depth of tree is one more than the largest level no. of tree. Depth is also sometimes known as height of the tree.

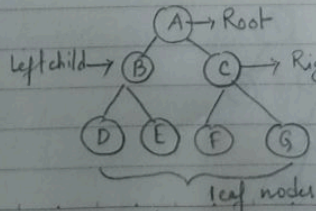
In the given binary tree, the highest level no. is 3, which is level no. of H. Here, the depth of the tree is 3+1 = 4 (more than largest level no).

Types of Binary tree:

Binary trees are of 3 types:

1. Complete Binary tree.
2. Almost complete binary tree.
3. Strictly Binary tree

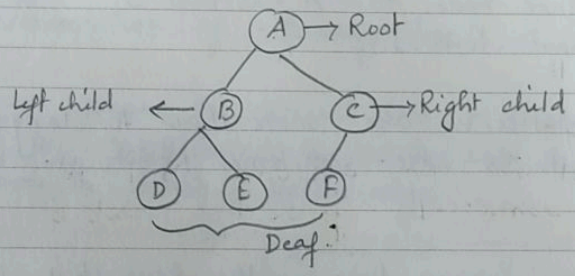
(1) Complete Binary tree: It is a tree in which each leaf is at the same distance from the root. It can be defined as a binary tree whose non-leaf nodes have non-empty left and right subtree. All leaves are at the same level.



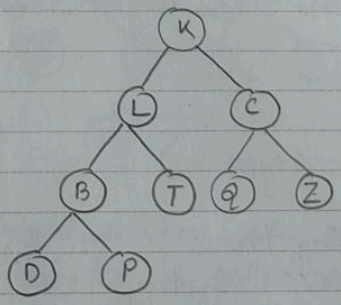
In a complete binary tree, the degree of every node is either 2 or

DATE

② Almost Complete Binary tree : It is a binary tree in which the following conditions hold - All the leaves are at the bottom level, all the leaves are in the left most possible positions, and all levels are completely filled with nodes.

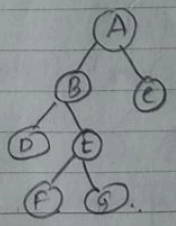


(a) Almost complete Binary tree.



(b) Almost complete Binary tree.

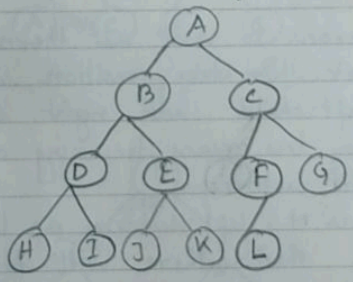
③ Strictly Binary Tree : If every internal node (non-terminal nodes) has its non-empty left and right children, then it is called strictly Binary tree.



Here every internal node A, B and E has 2 non-empty left and right childs. Hence it is strictly binary tree. This means each node in a binary tree will either have 0 or 2 children.

Note: Advantage of binary tree:

The main advantage with the structure is that we can easily find the left and right childs of any node and parent of any child. Left and right child of node will be at position $2n$, $2n+1$. Similarly, parent of any node N will be floor(N/2)



Parent of I is floor $\frac{9}{2} = 4$ i.e. D

Similarly left and right child of D will be -

$2n = 2 \times 4 = 8$ i.e H

$2n+1 = 2 \times 4 + 1 = 9$ i.e I.

DATE

Suppose it has N nodes. then the depth will be,
floor $(\log_2 N + 1)$

Suppose the tree has 1000 nodes. then depth will be.

floor$(\log_2 1000 + 1)$

=

Note: A strictly binary tree with N leaves always contain exactly $(2N-1)$ nodes.

Binary tree traversal Method:

To traverse a tree means to visit all the nodes in some specified order. In tree creation, we take 3 parameters-node, left child and right child.
So, traversing of binary tree means traversing of node, left subtree and right subtree
If root is denoted as N, left subtree as L and right subtree as R, then there will be six combinations of traversals.
NRL, LNR, NLR, LRN, RNL, RLN
In these, only three are standard.

NLR (Node - left - Right)
LNR (left - Node - Right)
LRN (left - Right - Node) traversals.

Here, NLR is called preorder, LNR is called in-order. and LRN is postorder.
Here, left subtree is always traversed before right subtree.

Natural

Traversals are —

① Inorder : (LNR) It is also known as symmetric order.
The traversal for inorder are —

i) Traverse the left subtree
ii) Visit the root.
iii) Traverse the right subtree.

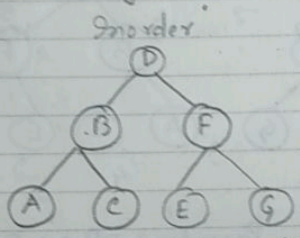


fig: A binary tree with 7 nodes.

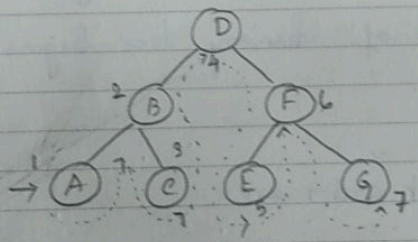


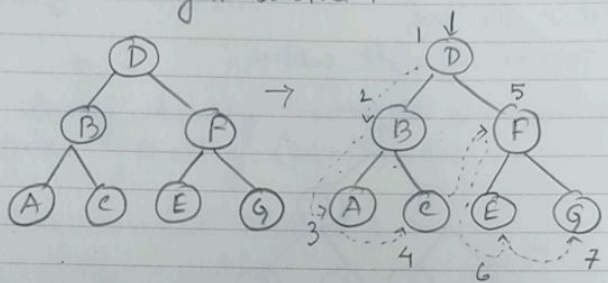
fig: Inorder traversal of binary tree

DATE

② Preorder : (NLR) (also known as depth first order)

The traversal for preorder are –

i) Visit the root.
ii) Traverse the left subtree.
iii) Traverse the right subtree.



Preorder : DBACFEG

fig: Preorder traversal for Binary tree.

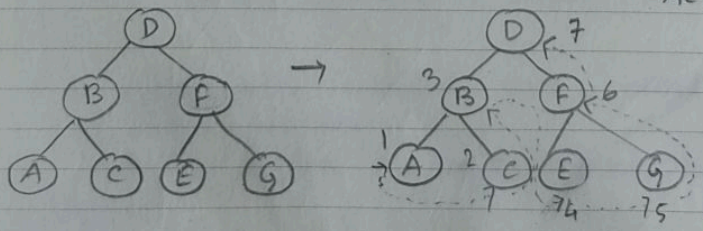
The preorder traversal of binary tree begins with root.

③ Postorder (LRN)

The traversal for post order are

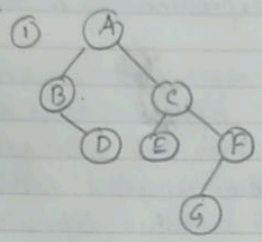
i) Traverse the left subtree.
ii) Traverse the right subtree.
iii) Visit the root.

Postorder → ACBEGFD



Natural

H.W

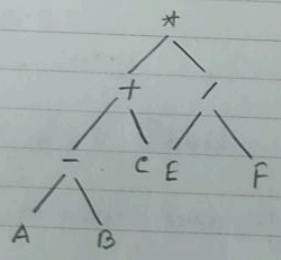


DATE

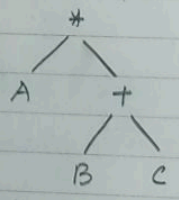
Representation of an algebraic expression in a Binary tree

A binary tree can be used to represent as an algebraic expression containing operands and binary operation such as +, -, /, *. The root node holds an operator and the other nodes represent either a variable name like a, b, c or another operator. The root of a strictly binary tree contains the operator and the leaf contains the operand.

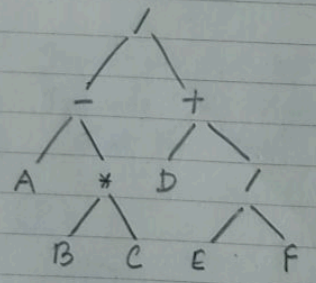
i) (A-B) + C * E/F



ii) A * (B+C)



iii) (A-B*C)/(D+E/F)



Natural

* Recursive Algorithm.

A binary tree is recursive in which each subtree is also a binary tree. Thus, traversing a binary tree involves visiting the root node and traversing its left and right subtrees. The only difference among the methods is the order in which these operations are performed.

Recursive algorithm of

(A) Inorder Traversal.

① Begin
Inorder (node) /* set the current node */
② If current node != NULL then
③ Call inorder (left child [node]).

/* Recursively do an inorder traversal of the left subtree of the current node */.

④ Print info [Node].

/* Print information at the current node */

⑤ Call inorder (Right child [node]).

/* Recursively do an inorder traversal of the right subtree of current node */

⑥ Stop.

DATE

(b) Preorder Traversal.

(1) Begin
Preorder (node) /* set the current node */

(2) If current node != NULL then
(a) Print info [node]

/* Print information at the current node */

(b) Call Preorder (left child [node])

/* Recursively do a preorder traversal of the left subtree of the current node */

(c) Call Preorder (right child [node])

/* Recursively do a preorder traversal of right subtree of the current node */

(3) Stop.

(c) Postorder Traversal.

(1) Begin
Postorder (node) /* set the current node */.

(2) If current node != NULL then

(a) Call Postorder (left child [node])

/* Recursively do a postorder traversal of left subtree of the current node */

(b). Call Postorder (right child [node]).

/* Recursively do a postorder traversal of right subtree of the current node */

(c). Print info [node]

/* Print information at the current node */

(3) stop.

Creation of Binary tree from Preorder and Inorder Traversals.

Preorder. : ABDHECFG
Inorder : DHBEAFCG.

Step 1 : In preorder traversal, root is at the first node. Hence A is the root of the binary tree.

(A) root

Step 2 : Now we can find the node of left subtree and right subtree with inorder sequence. Nodes which are in the left side of the root in inorder are nodes of left subtree and nodes which are in the right side of root in inorder are nodes of right subtree.

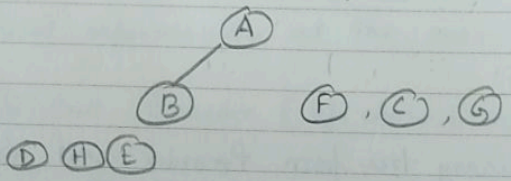
Nodes of left subtree → DHBE

Nodes of right subtree → FCG.

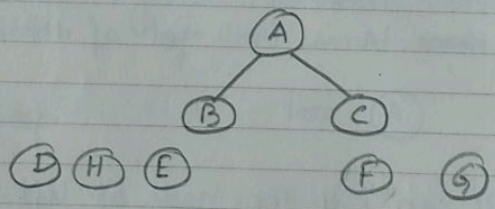
DATE

D, H, B, E F, C, G

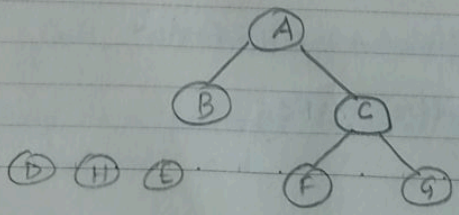
Step 3 → The left child of root 'A' will be first node in preorder traversal after root 'A'. Hence 'B' is the left child of A.



Step 4 → The right child of root A will be the first node after nodes of left subtree in preorder traversal. Hence 'C' is the right child of A

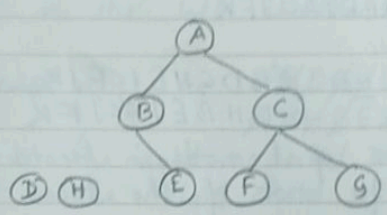


Step 5 → In inorder sequence 'F' is the left of 'C', and 'G' is on the right side of 'C'. Hence 'F' will be left subtree of 'C' and 'G' will be right subtree of 'C'.

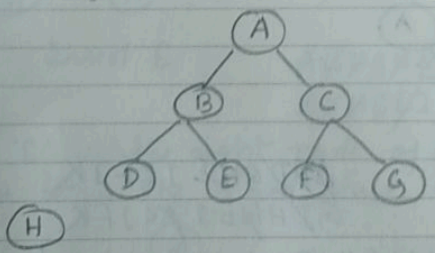


Natural

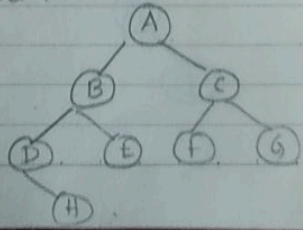
Step 6 → Now in inorder sequence 'D and H' are on the left side of 'B' and 'E' is on the right side of 'B'. So, D and H will form left subtree of B and E will be the right subtree of B.



Step 7 → In preorder sequence root is traversed before left and right subtrees. In preorder traversal of this tree 'D' is coming before H. Hence 'D' is the root of left subtree of 'B' and 'H' can be either in left or right subtree of D.



Step 8 → To findout whether 'H' is in left or right subtree of 'D', we look at inorder traversal. Since 'H' is the right subtree of 'D' which is a binary tree.



DATE

Shortcut method of creating tree from Preorder and Inorder traversal.

Preorder A B D G H E I C F J K
Inorder G D H B E I A C J F K

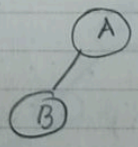
Step 1 : Insert A ~~A~~ B D G H E I C F J K Pre
G D H B E I ~~A~~ C J F K In

Since 'A' is the first node in Preorder traversal. Hence, it is the root of the tree.

(A) → root

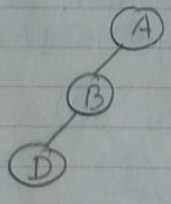
Step 2 : Insert B ~~A~~ ~~B~~ D G H E I C F J K Pre
G D H ~~B~~ E I ~~A~~ C J F K In

Since 'B' is the left of 'A' in inorder traversal. Hence it is the left child of 'A'



Step 3 : Insert D ~~A~~ ~~B~~ ~~D~~ G H E I C F J K Pre
G ~~D~~ H ~~B~~ E I ~~A~~ C J F K In

Since 'D' is the left of 'B' in inorder traversal. Hence 'D' is the left child of B.



Natural

Step 4 : Insert G

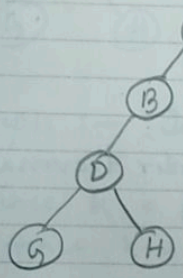
A B D G H E I C F J K Pre
G D H B E I A C J F K In

since G is the left node of D in inorder traversal.
Hence it is the left child of D.

Step 5 : Insert H.

A B D G H E I C F J K Pre
G D H B E I A C J F K In

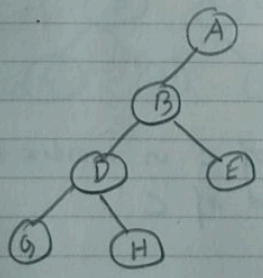
Since 'H' is the right node of D in inorder traversal.
Hence it is the right child of D.



Step 6 : Insert E

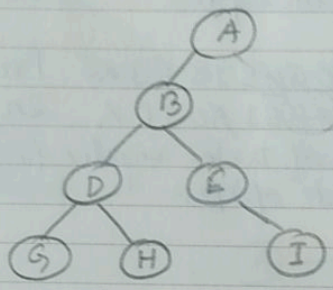
A B D G H E I C F J K Pre
G D H B E I A C J F K In

Since 'E' is the right node of B in inorder traversal.
Hence, it is the right child of B.



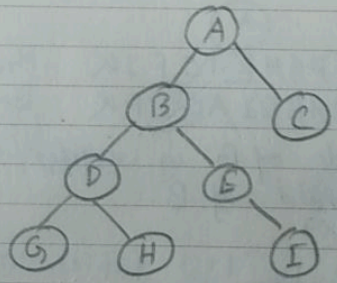
Step 7 :- Insert I Pre
In.

Since I is the right node of E in inorder traversal.
Hence it is the right child of 'E'



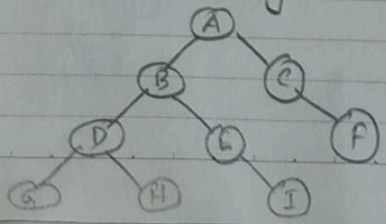
Step 8 :- Insert C

Since 'C' is the right of 'A' in inorder traversal.
Hence, C is the right child of A.



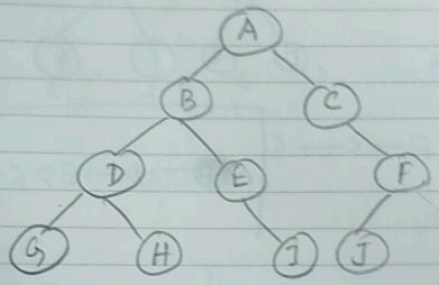
Step 9 :- Insert F.

Since 'F' is the right child of C in inorder traversa
Hence, 'F' is the right child of 'C'



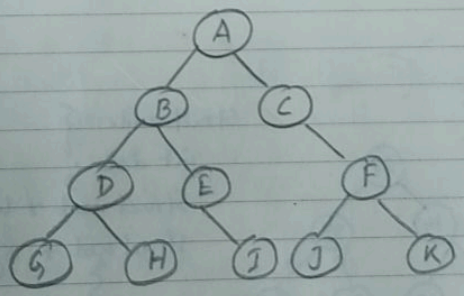
Step 10 :- Insert J

Since J is the left of F in inorder traversal.
Hence J is the left child of F.

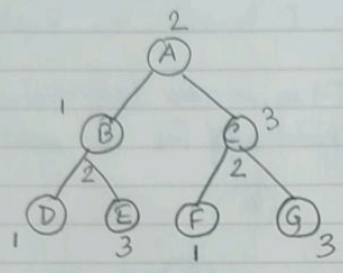


Step 11 :- Insert K

Since K is the right of F in inorder traversal.
Hence K is the right child of F

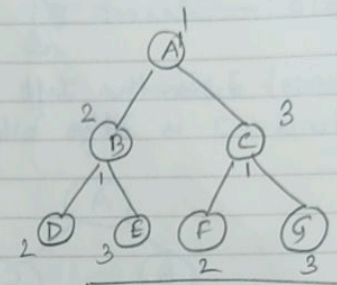


Inorder



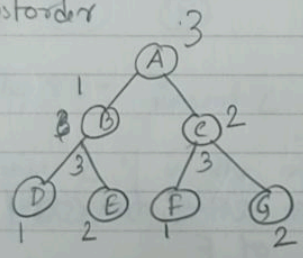
D → B → E → A → F → C → G

Preorder



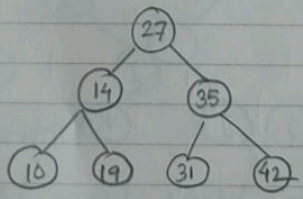
A → B → D → E → C → F → G

Postorder



D E B F G C A

Binary Search Tree :




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