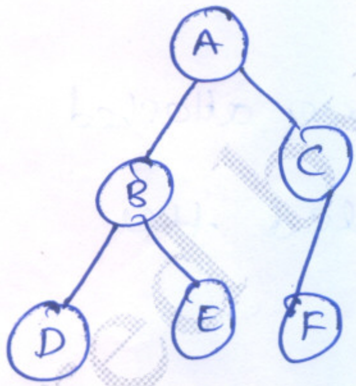
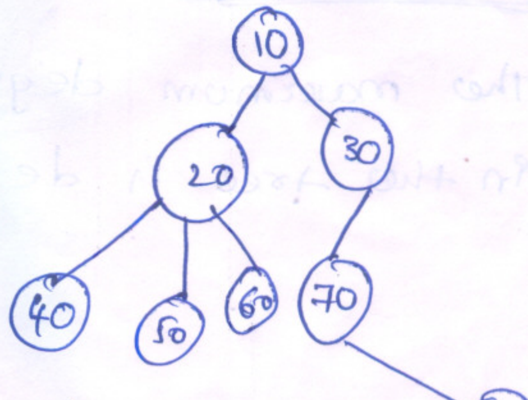
TREES UNIT-4

→ Basic Tree Concepts :-

Def for Tree :- A tree is a non linear data structure that can be represented in a hierarchical manner. It contains a finite set of elements called 'nodes' which are connected to each other using a finite set of directed lines called 'branches'



Root :- Root is a unique node in the tree to which further subtrees are attached

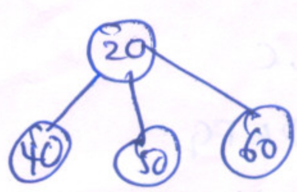


Form the figure given tree, node '10' is a root node

Parent node :-

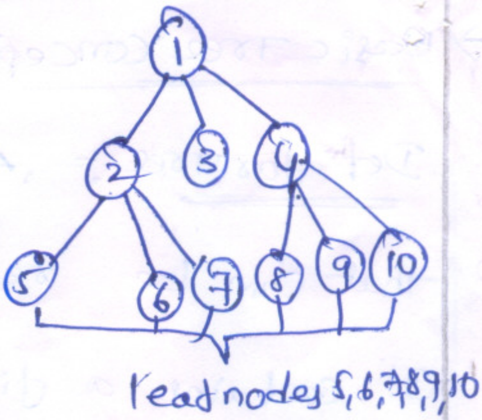
The node having sub-branches is called parentnode.

From the figure 20 is parent node of 40, 50, 60



leaf nodes :-

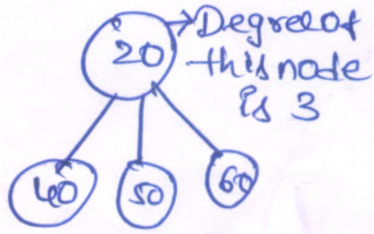
The nodes at the lowest level are known as the leaf nodes



Child node :- The root node is a special node with no parent node and leaf nodes have no child nodes.

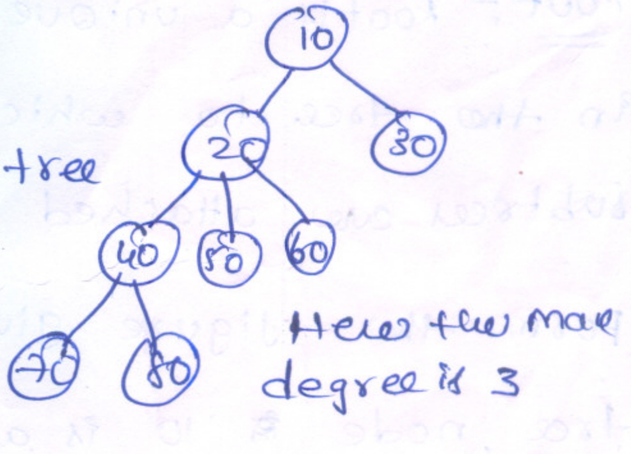
Degree of the node :-

The total no. of sub-trees attached to that node is called the degree of a node



Degree of the tree :-

the maximum degree in the tree is degree of tree



level of the tree :-

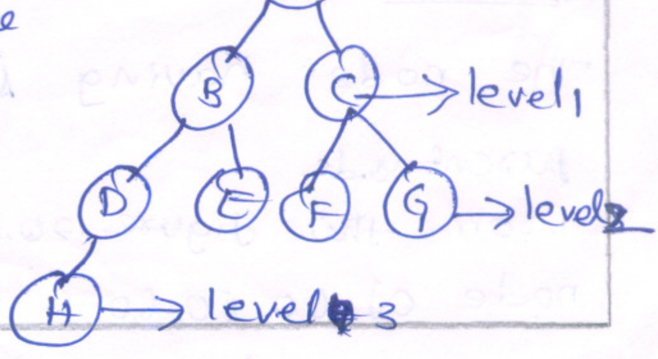
It is defined for binary tree

level0 :- A

level1 :- B, C

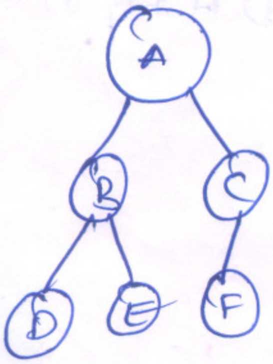
level2 :- D E F G

level3 :- H



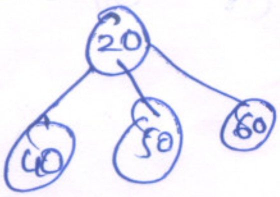
Height of the tree (or) depth of tree :-

- The maximum level is the height of the tree



The height of tree is 3
For sometimes height of the tree is also known as depth of the tree

Successor :- It is a node which occurs next to some node

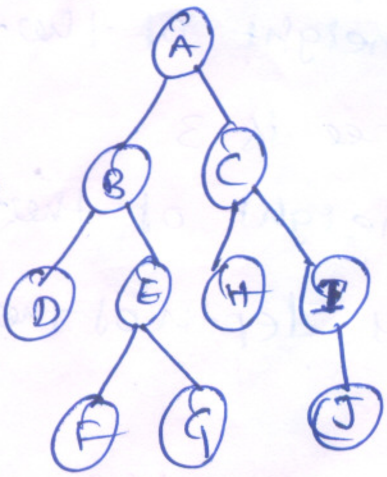


we read node 60 after reading node 20 then 60 is called successor of 20

predecessor :- Some particular node occurs previous to some other node then that node is called predecessor of the other node
In above figure we read 20 first and then we read node 40 then 20 is predecessor of 40.

Internal and External node :-
leaf nodes are not having further links then those leaf nodes are External nodes

and nonleaf nodes are called internal nodes



Internal nodes are B C E I

External nodes are D H F G J

Siblings (Brothers) :-

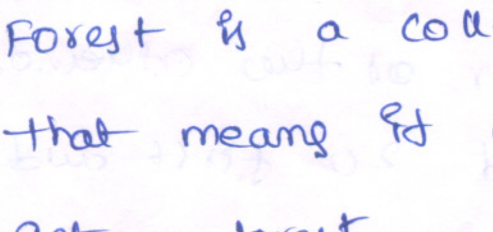
2 nodes are said to be siblings if they are ~~leaf~~ left and right child of same parent

From the above figure B C E I are siblings

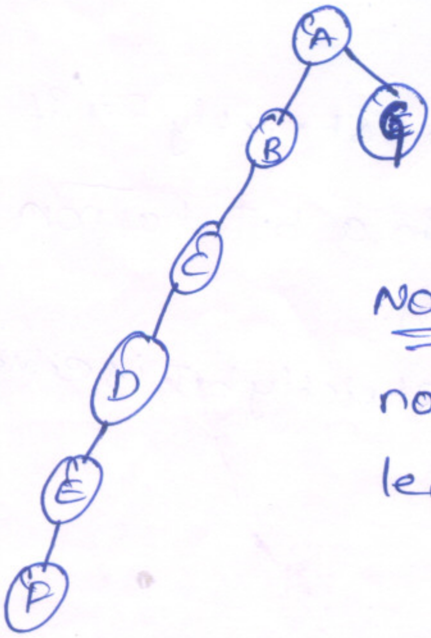
Forest :-

Forest is a collection of disjoint trees that means if we remove its root then we get a forest

From the above figure

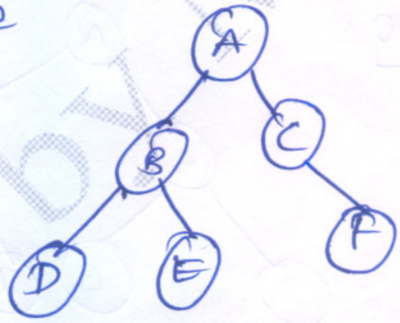


Ancestor:- It is a node which contains the subbranches of longer depth or height



Here A is an ~~a~~ ancestor of node D or E or F

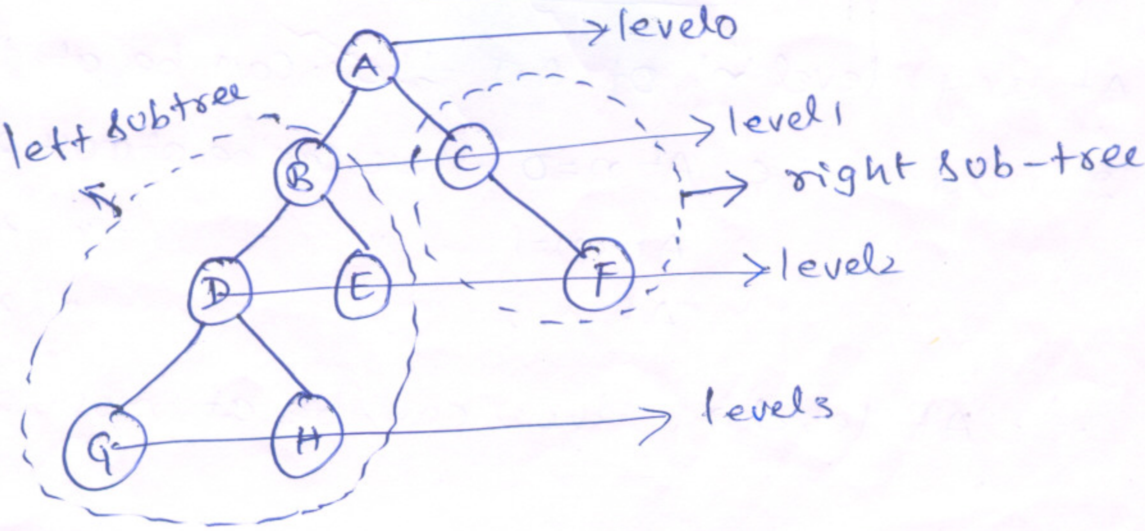
Non tereminals:- These are the nodes other than root node and leaf node



B,C are nonterminals

Binary Trees:-

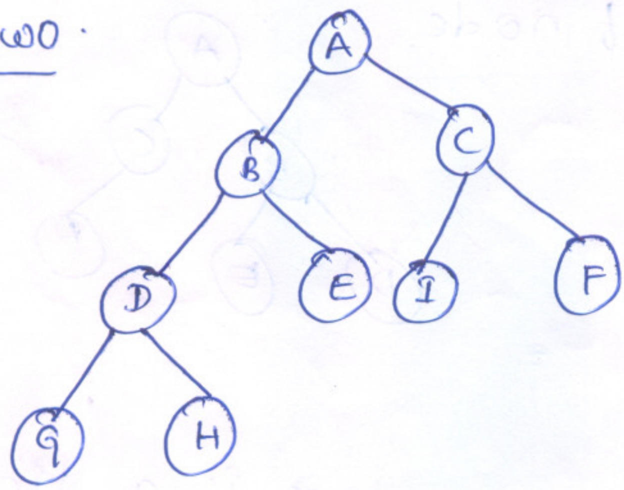
Def of B.T:- A binary tree is a finite set of nodes which is either empty or consists of a root and two disjoint binary trees called the left sub-tree and right sub-tree.



Forms of B.T :- There are various forms of b.T that are formed by imposing certain restrictions on them. Some variations include Strictly b.T, Complete b.T, extended b.T

Strictly B.T :- * A B.T is said to be strictly B.T if every internal node (non-leaf node) in a b.T has non empty left and right sub-trees.

* The degree of each node in a strictly b.T is either zero or two.



Complete B.T :-

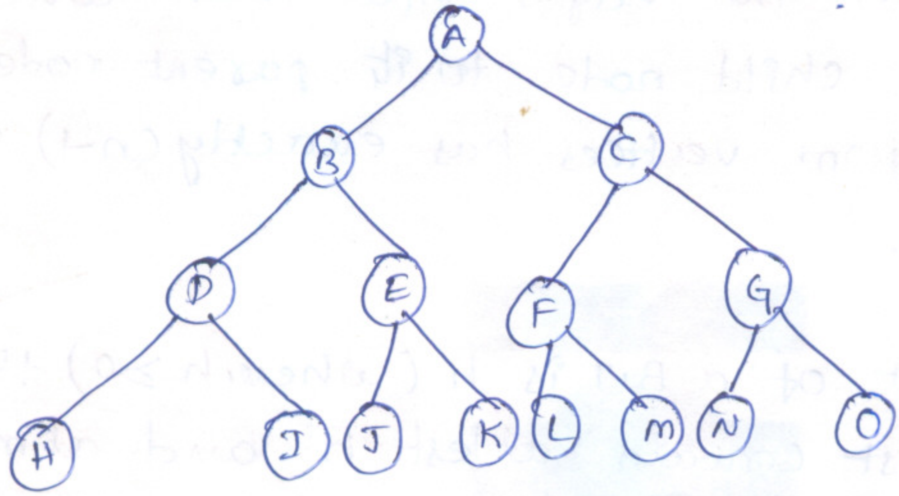
* A B.T is said to be a complete B.T if all the leaf nodes of the tree are at same level
* The tree has maximum no. of nodes at all the levels

At any level 'n' of B.T there can be at the most 2^n nodes. i.e At $n=0$ there can be atmost $2^0=1$ node

At $n=1$ " " $2^1=2$ nodes

At $n=2$ " " $2^2=4$ nodes

$\therefore$ At level n, there can be at most 2^n nodes.



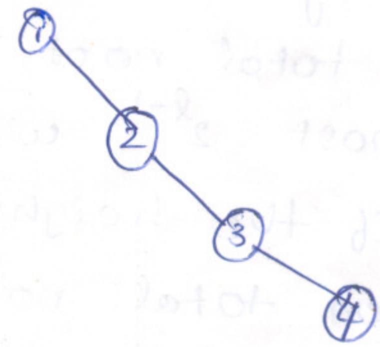
Complete B.T

Skewed B.T :- A B.T which contains only left child nodes or only the right child nodes is called a Skewed B.T

Example :-

left Skewed B.T

Right Skewed B.T



properties of B.T :-

property 1 :- A B.T with 'n' vertices must contain exactly (n-1) edges

proof :- In a B.T, every node except the root node contains a single parent node also the nodes

of the B.T have the edges b/w them which Connects every child node to its parent node.
a binary tree of 'n' vertices has exactly (n-1) edges

Property 2 :-

If the height of a B.T is h (where h ≥ 0) then the tree must contain atleast 'h' and atmost $2^h - 1$ nodes.

proof :-

* Every level of the B.T contains one or more elements the total no of elements in a tree is atleast 'h'
* Every node of the B.T contains atmost 2 childs the total no of nodes in a tree at level 'l' is atmost 2^{l-1} where $l > 0$
* If the height of the tree is zero (h=0) then total no of nodes in a tree is zero

$$2^l - 1 = 2^0 - 1 = 1 - 1 = 0$$

* If the height of a tree is greater than zero then total no of nodes is atmost $2^h - 1$
* It cannot exceed the value $\sum_{l=1}^{h} 2^{l-1} = 2^h - 1$

Property 3 :-

The height of a B.T with 'n' vertices is atmost 'n' and atleast $\lceil \log_2 (n+1) \rceil$

proof :-

* Every level of the B.T contains atleast one element the height of the B.T is atmost 'n'
* It connot exceed the value 'n'
* From property 2 it is clearly stated that B.T of height 'h' has atmost $2^h - 1$ nodes

$$\therefore n \leq 2^h - 1$$

$$\Rightarrow n+1 \leq 2^h$$

Apply 'log' on bothsides

$$\log(n+1) \leq \log 2^h$$

$$\log(n+1) \leq h \log 2$$

$$\frac{\log(n+1)}{\log 2} \leq h$$

$$\therefore \log_2(n+1) \leq h$$

$$h \geq \log_2(n+1)$$

$$h \geq \lceil \log_2(n+1) \rceil \quad (\because h \text{ is an integer})$$

→ Representation B.T

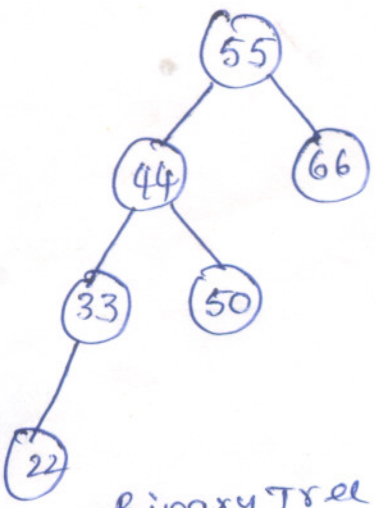
there are 2 types of representation for B.T using 1, Array (linear or sequential representation)
2, linked list

Array Representation of Binary Tree:-

This representation uses only a single linear array tree as follows:

i, The root of the tree is stored in tree[0]

ii, If a node occupies tree[i] then its left child is stored in tree[2*i+1] its right child is stored in tree[2*i+2] and the parent is stored in tree[(i-1)/2]



Binary Tree

55	44	66	33	50			22
0	1	2	3	4	5	6	7

Array representation of B.T

seQ1

linked list Representation of B.T:-

A node is divided in to ~~3~~ 4 fields

info, left, Right, root

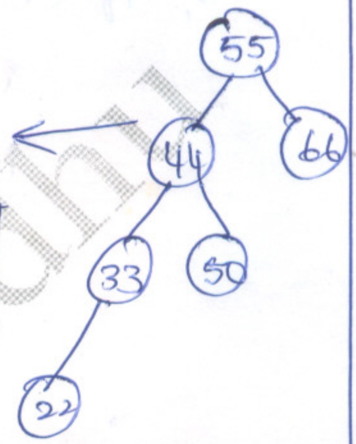
info:- which is used to stored the data item

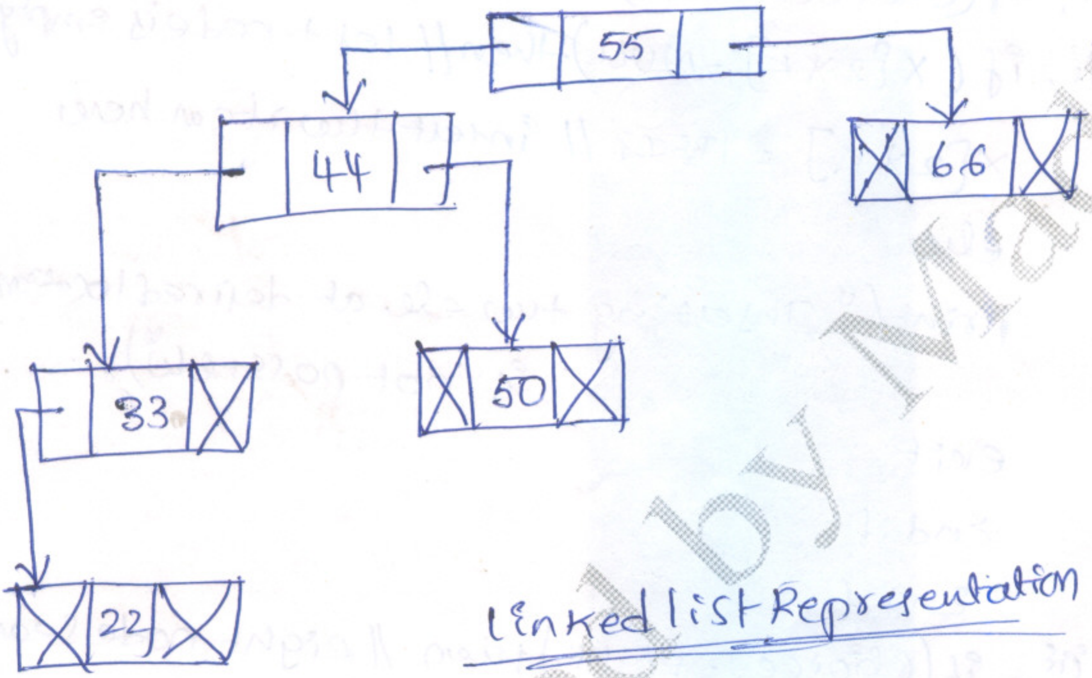
left:- left pointer field which is used to store the address of the left child.

Right:- right pointer field which is used to store the address of the right child

root:- which will contain the location of the root node. If any subtree is empty then the corresponding pointer will contain the NULL value

* If the tree it self is Empty then the root will contain the NULL value.





Linked list Representation ✓

→ Operations on Binary Tree:-

4 Basic operations supported by a binary tree are listed below 1, Insertion 2, Deletion 3, Traversal 4, merge.

1, Insertion:- *This operation inserts a new node at any position in the B.T.

* If the node is to be inserted internally i.e inside the B.T it is called internal node

Algorithm for insertion operation in a sequential storage

step 1. K = Search(1, key) //Search key node in the tree

2, if (K=0) then
print" Insertion failed"
endif
3, if ((X[2*K] =NULL || (X[2*K+1] =NULL))
i, if(Choice =left) then // leftnode search
ii, if (X[2*K] =NULL) then // left node is empty
X[2*K] =ITEM // insert the item here
else
print(" Inserting the ele. at desired location is not possible");
exit
Endif
Endif
iii, if(Choice =Right) then // Right node search
if (X[2*K+1] =NULL) then // right node is Empty
X [2*K+1] =ITEM // insert the item as the right node
else
print (" Desired operation can't be performed");
exit
Endif
4, Else
print" Insertion of item is not performed"
Endif
5, Exit

Search Function for Sequential Representation:-

Here Index = Index of the node from where search has to be started

Step1: j = INDEX; //set point of search i.e at root

Step2: if(X[i]!= key) then // If node is not equal to required key

i) If (2*j ≤ S) then // left sub tree is searched

Search(2*j, key)

ii) Else

iii) if (2*j+1 ≤ S) then //Right subtree is searched

a) Search (2*j+1, key);

b) Else

c) return(0) //unsuccessful search

d) endif

e) endif

Step3: Else

Return(i); //Return j, i.e address of the key

Endif

Representation Search Function for linked Representation

Step1: ptr = ptr1 // ptr is the node where search has to be started

Step2: if (ptr → item != key) //current node is not required

if (ptr → leftchild != NULL)

Search (ptr → leftchild) //search left subtree

Else

return(0)

Endif

if (ptr → Rightchild != NULL)

Search (ptr → Rightchild) //Search Right sub-tree

Else

Return(0)

Endif

Step3: Else

Return(ptr) //pointer Contains the address of key

Step4: Endif

Insertion Function Algorithm:-

Step1: ptr = Search (root, key) //Call the search fun

Step2: if (ptr = NULL) then //ptr is NULL

print "unsuccessful search"

Exit

Endif

Step3: if (ptr → leftchild = NULL) // (ptr → Rightchild = NULL)

Ask user whether left element or right element is being entered

if (choice = leftchild) then

Step4: if (ptr → leftchild = NULL) then // leftchild NULL

new → Getnode (Node) //Create a new node

new → item; //new node is assigned to item

new → leftchild = NULL;

new → Rightchild = NULL;

ptr → leftchild = new //current node is linked with new node towards left

Set-II Q1

Binary Tree Traversals :-

* The process of going through a tree is such a way that each node is visited once and only once is called tree traversal
* The traversal in a B.T involves 3 kinds
  1, visiting the root
  2, Traverse the left subtree
  3, Traverse the right subtree.
* The only difference among the methods is the order in which these 3 operations are performed
* There are 3 standard ways of traversing
  1, In-order
  2, pre-order
  3, post-order.

1, In-order :- (L, Root, R)

Basic steps for Inorder

1, Traverse the left subtree in inorder
2, visit the root
3, Traverse the right subtree in inorder

Recursive algorithm for Inorder :-

~~preorder (node)~~
~~1. If node ≠ NULL~~
~~2. process (node)~~
~~3. preorder (left[node])~~
~~4. preorder (right[node])~~
~~[End of step 1 If structure]~~
~~5. Exit~~

Else

print " Insertion is not performed";

Exit

Endif

Else

Step5: if (ptr → Rightchild=NULL) //Rightchild is NULL

new → GetNode (Node) // create a new node

new → item = ITEM // New data is assigned with the required value

new → leftchild = NULL

new → Rightchild = NULL

ptr → Rightchid = new // current node is linked with its Right child to new node

else

print "Insertion as rightchild is not done";

Endif

Else

print "Key Node has a child"

Endif

Endif

exit

Deletion:-

Step1: set flag as FALSE // start at root node

Step2: K = Search (1, key // searching from index ie root

Step3: if (K==0) Goto Step7

Step4: if ((X[2*K]==NULL) && (X[2*K+1]==NULL)

// Testing left node

Set flag TRUE // delete the item

X[k]=NULL; // Make it as NULL

Step5: Else

print "item specified is not a leaf node"

Step6: endif

Step7: if (flag==FALSE) // if flag=FALSE no such node exists

print "Deletion Failed"

Step8: endif

Step9: STOP

Recursive Algorithm for In-order:

In order (node)

1. If node ≠ NULL
2. Inorder (left [node])
3. process (node)
4. Inorder (right [node])

[End of step 1 if structure]

5. Exit

Non-Recursive algorithm for In-order:

Inorder Trav (root)

1. Set top = 0, stack [0] = NULL, node = root
   [Initially push NULL on to stack and initialize node]
2. Repeat while node ≠ NULL
   [pushes left most path on to stack]
   Set top = top + 1, stack [top] = node
   Set node = left [node]
   [End of step 2 loop]
3. Set node = stack [top], top = top - 1
   [pops node from stack]
4. Repeat steps 5 to 7 while node ≠ NULL
5. process (info [node])
   [process the node]
6. If right [node] ≠ NULL
   (right child exits)

Set node = right[node], Go to step 2

[End of step 6 if structure]

7. Set node = stack[top], top = top-1

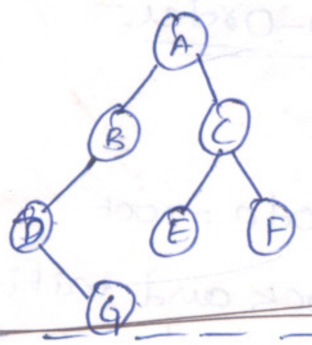
[pops node from stack]

[End of step 4 loop]

8. Exit

<u>Example</u>:- consider the binary tree by using following postorder

tree



<u>Ans</u>:- GDBEFCA

Steps	Node processed	Stack
1) Initially push NULL onto stack. Set node = A		0 ← top
2) By taking node = A proceed to left most path on the way if right child is obtained push -ve of that on to stack		G ← top, D, B, C, A, 0
3) POP and process +ve node if -ve node is popped make it +ve and repeat step G is popped so Node = -G now reset node = G		D ← top, B, C, A, 0

Node processed	Steps	Stack
	4) By taking node = G proceed to left most path, push G on to stack	G (top), D, B, -C, A, 0
G D B	5) pop and process G, D and B, -C is popped so node = -C now reset node = C	A (top), 0
	6) By taking node = C proceed to left most path if right child is obtained push -ve of that on to stack push C -F and E	E (top), -F, C, A, 0
E	7) pop and process E, -F is popped so, node = -F now reset node = F	C (top), A, 0
	8) By taking node = F, proceed to left most path push F on to stack	
F C A	9) pop and process F, C and A	0 (top)
	10) The next element NULL is popped since node = NULL algorithm is completed	Stack empty

O/P :- GDBEFCA

preorder traversal :- (R, L, R) (Root, Left, right)

Basic step for preorder

1. visit root
2. Traverse the left subtree in preorder
3. traverse the right subtree in preorder

~~Algorithm for~~

Recursive algorithm for preorder:-

preorder (node)
1. If node ≠ NULL
2. process (node)
3. preorder (left [node])
4. preorder (right [node])

[End of step 1 If structure]

5. Exit

Non-Recursive algorithm for preorder:-

preorder trav(root)

1. Set top = 0, stack [0] = NULL, node = root
   [initially push NULL on to stack and initialize node]
2. Repeat steps 3 to 5 while node ≠ NULL
3. process (info [node])
   [process the node]

4. If right[node] ≠ NULL

Set top = top+1, Stack[top] = right[node]

[push the right child into the stack]

[End of step 4 If structure]

5. If left[node] ≠ NULL

Set node = left[node]

else

Set node = Stack[top], top = top-1

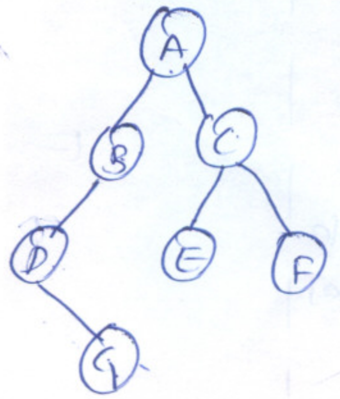
[pops node from stack]

[End of step 5 If structure]

[End of step 2 loop]

6. Exit

Example :- consider the B.T as shown in below and simulate the algorithm by using preorder



Ans: A B D G C E F

1) Initially push NULL onto stack Set node=A		0 top
2) i) process A and push its right child on to stack	A	C top 0

ii) process B (no right child)	B	C (top), 0
iii) process D and push its right child on to stack no other node is processed since D has no left child	D	G (top), C, 0
3) pop the top element from stack and set node = G		C (top), 0
4) since node ≠ NULL Enter ~~the~~ into the loop process G	G	C (top), 0
5) pop the top element from stack and set node = C		0 (top)
6) Since node ≠ NULL Enter into the loop i) process C and push its right child on to stack ii) process E no other node is processed since E has no left child	E	F (top), 0
7) pop the top element from stack and set node = F		0 (top)
8) since node ≠ NULL enter into the loop process F	F	0 (top)
9) pop the top element from stack & set node = NULL		Stack empty

10) Since node = NULL the algorithm is completed.

postorder Traversal :- (L, R, Root)

Basic steps for postorder.

1. Traverse the left subtree in postorder

2. traverse the right subtree in postorder

3. visit the root

Recursive algorithm for postorder :-

postorder (node)

1. If node ≠ NULL

2. postorder (left [node])

3. postorder (right [node])

4. process (node)

[End of step 1 If structure]

5. Exit

Non-recursive Algorithm for postorder :-

postorder Trav (root)

1. Set top = 0, stack [0] = NULL, node = root

[Initially push NULL on to stack & initilize node]

2. Repeat step 3 to 5 while node ≠ NULL

[pushes left most path on to stack]

3. Set top = top + 1, stack [top] = node

[pushes node on to stack]

4. If right [node] ≠ NULL

[right child exits]

Set top = top+1, Stack [top] = -(right [node])

[pushes the -ve pointer]

[End of If structure]

5. Set node = left [node]

[End of step 2 loop]

6. Set node = Stack [top], top = top-1

[pops node from stack]

7. Repeat while node>0

[while non -ve pointer]

process (info [node])

Set node = Stack [top], top = top-1

[End of loop]

8. If node<0

[test for -ve pointer]

Set node = -node

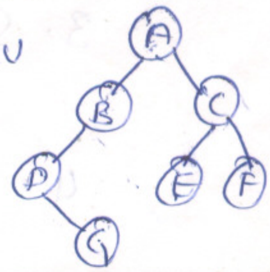
[change to the original value of node]

Go to step 2

[End of If structure]

9. Exit

example :- Consider the B.T by using below tree simulate the algorithm Inorder trav



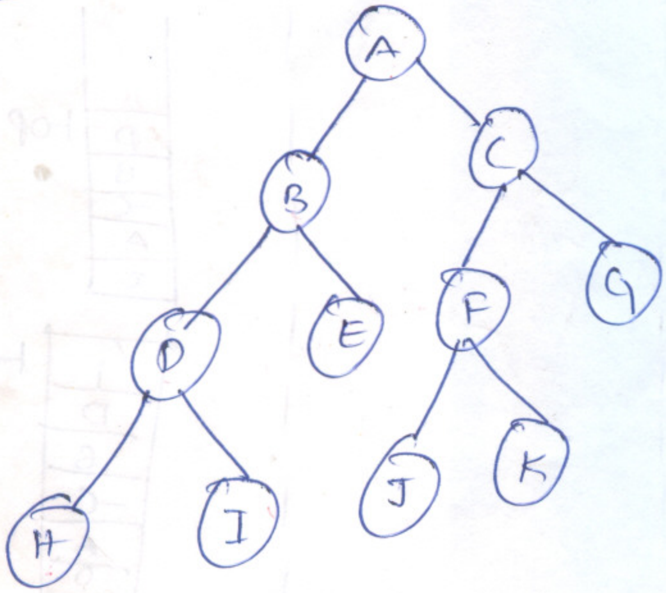
Ans :- D G B A E C F

Steps	Node processed	Stack
1) Initially push NULL on to stack set node = A		0 ← top
2) By taking node = A proceed to left most path on the way if right child is obtained push -ve node		-G ← top D B -C A 0
3) pop & process +ve node if -ve node is popped make it +ve & repeat the step node = -G now reset node = G		d D ← top B -C A 0
4) By taking node = G proceed to left most path push G onto stack		G ← top D B -C A 0
5) pop & process G, D & B, -C is popped node = -C now reset node = C		~~G~~ ← top ~~D~~ ~~B~~ ~~C~~ ~~A~~ ~~0~~ E ← top -F C A 0
6) By taking node = C proceed to left most path if right child is obtained push C -F and E		top → E -F C A 0
7) pop & process E, -F is popped so, node = -F now reset node = f		C ← top A 0
8) By taking node = F proceed to left most path push F on to stack		F ← top. C A 0

9) pop & process F, C & A	F C A	0 top
10) The next element NULL is popped since node = NULL algorithm is completed		Stack Empty

Set II Q)

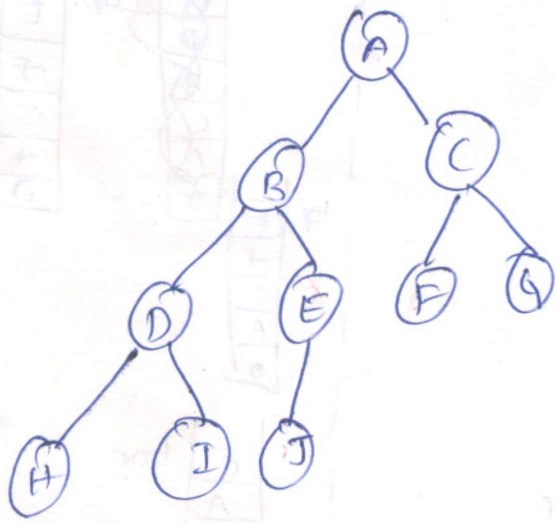
Example



preorder = ABDHIECFJKG

Inorder = HDIBEAJFKCG

postorder = HIDEBJKFGCA



preorder = ABDHIEJCFG

Inorder = HDIBJEAFCG

postorder = HIDJEBFGCA