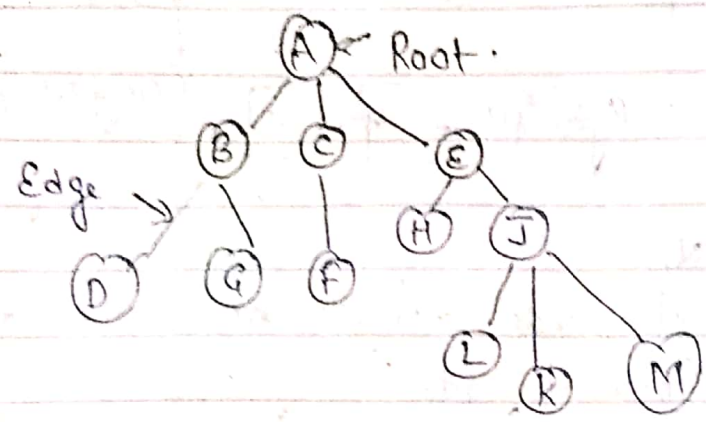
Unit - 4th.

Tree (hierarchy)



① Node :- Each data item in a tree.

② Degree of a tree :- Maxm deg of a node in a tree.

Eg :- Deg of above tree = 3 (A have 3 child)

A → (have who many child) = degree

Child = Degree

A = 3

E = 2 (H, J), C = 1 (F)

A = 1, 2, 3. (have any child)

③ Degree of a node = The total no. of children associated with a node is called as deg of that node.

Eg :- Deg of A = 3
Deg of E = 2

? (Sister & Brother)

④ Sibing = D and G are sibing of parent node B.

Eg:- H, J for E. (only one parent)

⑤ → Internal node :- (IN)

all node those have children node called as internal node.

(all have child called IN)

Eg:- F, D, G are not an internal node.

⑥ Non-terminal node :- any node "except Root" node "whose degree is not zero"

(have children B, C, E, A, J & not consider Root node)

⑦ Leaf (new leaf) node :- Those node, which have "no child" are called leaf node.

Eg:- D, G, F, H, L, K, M

⑧ forest :- Set of disjoint tree.

(more than tree but don't mix up)

Internal node :- except Root

Non-terminal node :- not --"--

Imp.

(9) Height of a node :-

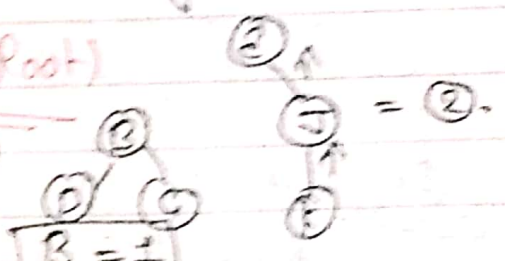
no. of edge on the longest path from the node to a leaf

A leaf node will have a hight of 0:

$H(E) = 0$

(child to Root)

(Bottom to top)



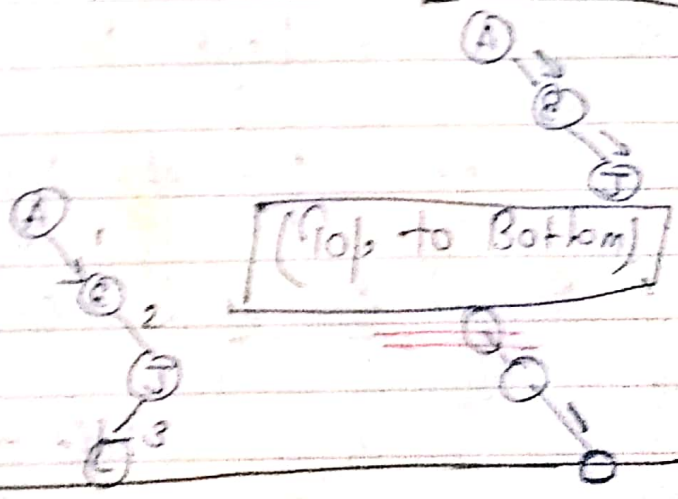
(10) Depth of node :-

no. of edges from the node to the true root node.

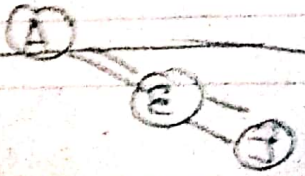
A Root node - depth of 0.

(1) 'E' :- D(J) = 2

(2) D(L) =



(11) Hight of tree = Hight of Root node = 3.



Q1) What is the hght of node H ?

Height of leaf node = 0

Q2) What is the degree of G.

Deg of leaf node = 0

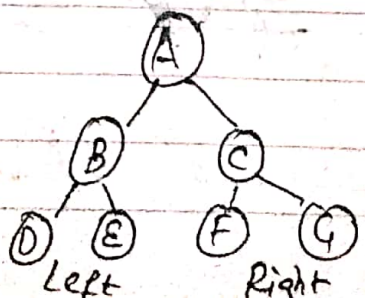
Trick
D = Daughter.

Q3) Height of a Tree
A:- 3 (hght of Root node)

Q4.)

Ordering in Tree.

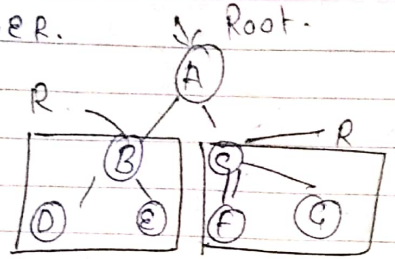
1. Preorder = Root, Left, Right.
2. Inorder = Left, Root, Right
3. Postorder = ~~Rot~~ Left, Right, Root
4. Breadth first or Level Order. = Top to Bottom & Left to Right



Root, 2 direction

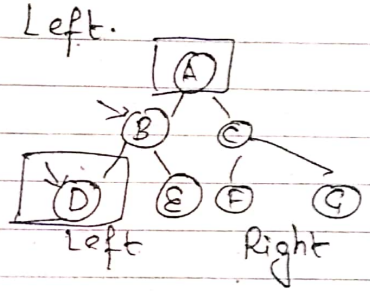
5

① PREORDER.



A, B, D, E, C, F, G - Preorder.

② Inorder.



Inorder. D, B, E, A, F, C, G.

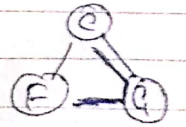
Post order.

L, R, Root

(D, B, E, F, C, G, A)

F, G, C, A

L, R,



Level order:-

A, B, C, D, E, F, G.

N(0) = 0
N(1) = 2

(7) formula.

AVL Tree	Binary Tree.
① Maxm no. of nodes in an AVL Tree of height H $2^{H+1} - 1$	Minm no. of nodes in a binary tree of hght H = H+1
② Minm no. of nodes in an AVL Tree of hght H is given by a recursive relation. $N(H) = N(H-1) + N(H-2) + 1$	Maxm no. of nodes in a binary tree of height H = $2^{H+1} - 1$
③ Minm height of an AVL Tree using N nodes floor ($\log_2 N$)	③ leaf nodes in a Binary Tree of hght H = Degree - 2 nodes + 1
④ Maxm hght of an AVL Tree using N nodes is calculated using recursive relation. $N(H) = N(H-1) + N(H-2) + 1$	④ Maxm no. of node at any level 'L' in a binary tree = 2^L

Q1.) maxm no. of node of an AVL Tree of hght 3

$2^{3+1} - 1 = 2^4 - 1 = \boxed{15}$.

$\boxed{2^{H+1} - 1}$ → formula.

Q2.) minm no. of node in a binary tree of hght 4.

$\boxed{N = H + 1}$

A:- $\boxed{4+1 = 5.}$

Q3.) Minm hght of an AVL Tree using 8 node

Soln:- $\downarrow floor(\log_2 N)$

$= 2^8 = 3$ cube.

$\boxed{3}$.

Q4.) In a bT, Deg 2 node is 3, then what are the no. of leaf node.

leaf node of BT = Degree − 2node + 1.

$3 + 1$.

$= \boxed{4}$. A:-

Q5.) find the minm no. of node required to construct an AVL Tree of $\boxed{\text{hght} = 3.}$

9

① Circled modified Restriction Clan

Step 1:-

Substituting $n=3$ in the recursive relation, we get :-

$$N(3) = N(3-1) + N(3-2) + 1.$$
$$N(3) = N(2) + N(1) + 1$$
$$N(3) = N(2) + 2 + 1$$
$$N(3) = N(2) + 3 \quad - (1)$$

To solve this Recursive relation, we need the value of $N(2)$

Step 2:-

To find $N(2)$, substitute $n=2$ in the Recu Relation.

Then, we get:-

$$N(2) = N(2-1) + N(2-2) + 1$$
$$N(1) + N(0) + 1.$$
$$= 2 + 1 + 1$$
$$N(2) = 4 \quad - (2)$$

$$\boxed{N(n) = N(n-1) + N(n-2) + 1.}$$

$\therefore$ Where $N(0) = 1$ and $N(1) = 2$.

Step 3:-

Using (2) in 1, we get:-

$N(3) = 4 + 3$. $(\because N(2) + 3) \ldots 1)$

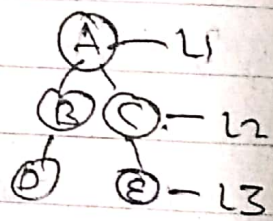
$N(3) = 7$.

The minm 7 nodes are required to construct an AVL tree of hght = 3.

Q find $h = 12$

Q4. What is the maxm no. of nodes at any level 3 in a BT?

Ans:- $2^2 = 2^3 = 8$ .. A.



Q5. What is the maxm hght of any AVL tree with 10 nodes.

$N(H) = N(H-1) + N(H-2) + 1$.

$H = 2$ in the Recursive relation,

$N(2) = N(2-1) + N(2-2) + 1$

$N(1) + N(0) + 1$.

$N(2) = 2 + 1 + 1 = 4$ — (1).

So, minm permissible no. of nodes in an AVL tree of hght 2 = 4.

Step 2:- H=3 in recursive relath, we get,

$N(3) = N(3-1) + N(3-2) + 1$

$N(2) + N(1) + 1$

$4 + 2 + 1 = 7$

So, minm permissible no. of nodes in an AVL tree of hght 3 = 7

Step 3:- Substituting H=4 in the RR, we get,

$N(4) = N(4-1) + N(4-2) + 1$

$N(3) + N(2) + 1$

$7 + 4 + 1$

$N(4) = 12.$

So, minm permissible no. of node in an AVL tree of hght 4 = 12

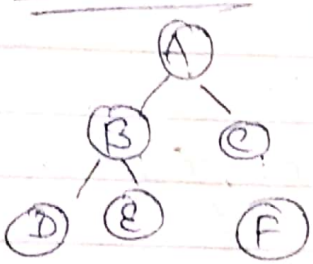
Hence, the maxm hght of an AVL tree that we can obtain using 10 node = 3.

3 & 4 N(3) & N(4)

∴ only consider (small)

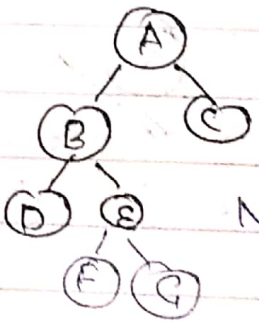
Types of BT

① Rooted BT



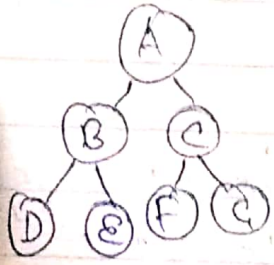
Root node + almost 2 child.

② full | Strictly BT



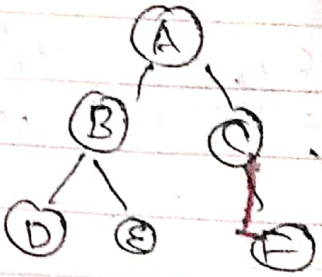
Node = either 0 or 2 child

③ Complete | perfect BT

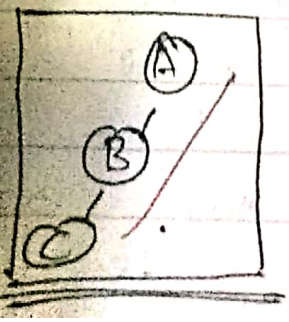


Every internal node & exactly 2 child & all leaf node at same level

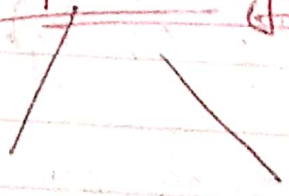
④ almost CBT.



⑤ Skewed BT.



Left & Right.



Binary tree :-

It defined as a finite set of element, called node. Such that

1.) T is empty (called null / empty tree)

2.) T contain a distinguished node R, called the Root of T, & the Remaining node of T from an ordered pair of disjoint binary tree T_1 and T_2.

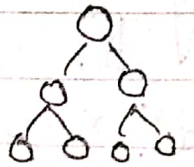
The defn

It is Recursive since T is defined in term of the binary subtree T1 and T2.

this means, in particular, that every node N of T contain a left & a Right subtree

Types

CBT :- A complete binary tree is defined as a binary tree whose non-leaf node have non-empty left & Right sub-tree and all leaves are at the same level.



(2) almost complete Binary tree :-

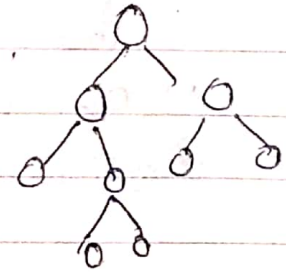
A CBT is defined as

binary tree whose non-leaf node have non-empty left & Right subtree & all leaves are either at the last level or 2nd last level.

(3) Strict Binary Tree (no 1 child, leaf node, 2 child)

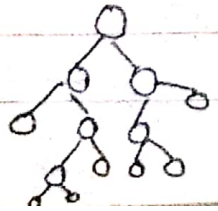
If every non leaf node in the Binary tree has non-empty left & Right subtree.

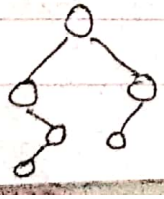
It means each node will have either 0 or 2 children



(4) Extended Binary Tree:-

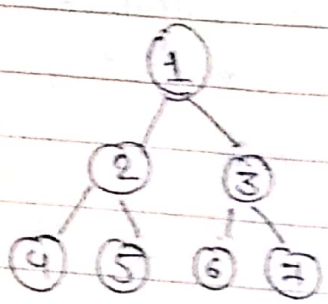
It is a tree that has been xformed into full binary tree. This xformatn is achieved by inserting special "external" node such that every "internal node" has exactly 2 children





(all hve internal node have exactly 2 child)

Representation of BT.



data store.

There are 2 possible representation of binary tree.

1.) array Rep.
2) Linked List Representation. (with the help of pointers)

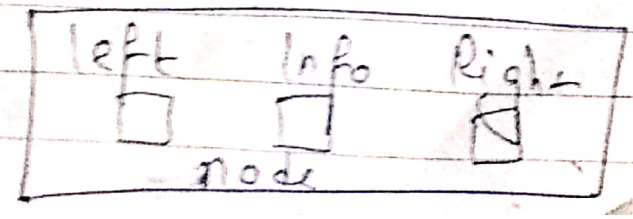
* Condition for any representation:-

* One shld hv direct access to the root R of T

* Any node N of T, one shld hv direct access to the children of N.

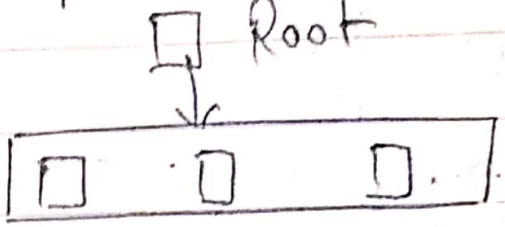
Linked Representation:-

i) Info is any information
ii) Left & Right are pointers to child node



6

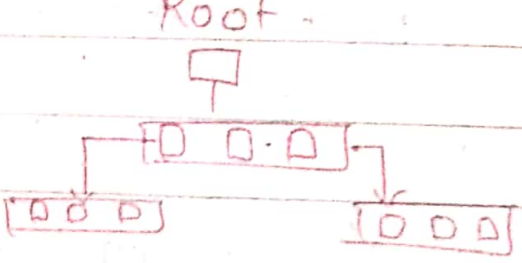
* Root is a node pointer variable to point root node of the tree.



x either root contain null, tree is empty.

Structure to Represent a BT:

```
Struct BinaryTreeNode
{
  int info;
Struct BinaryTreeNode * left;
  ——  ,,  ——————  * Right;
};
```



17

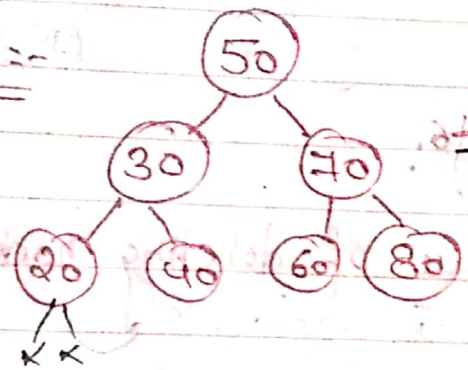
~~Xyzorg~~ (BT) Binary Search Tree.

1. Deletion:-

Imp:-

① Node = No child
② Node = 1 child
③ Node = 2 child

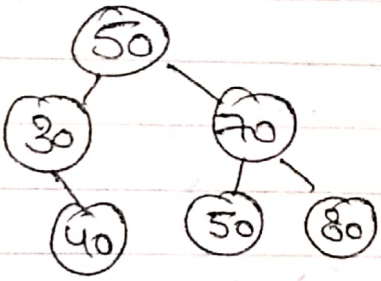
Case 1:-



del 20.
Just del.

direct del
(There are no child. directly you delete)

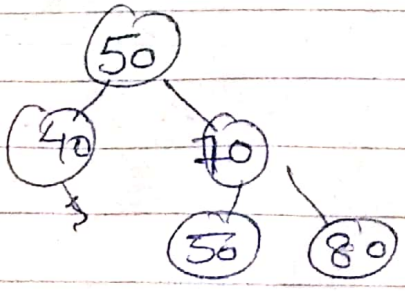
Case 2:-



Delete 30 (having 1 child)

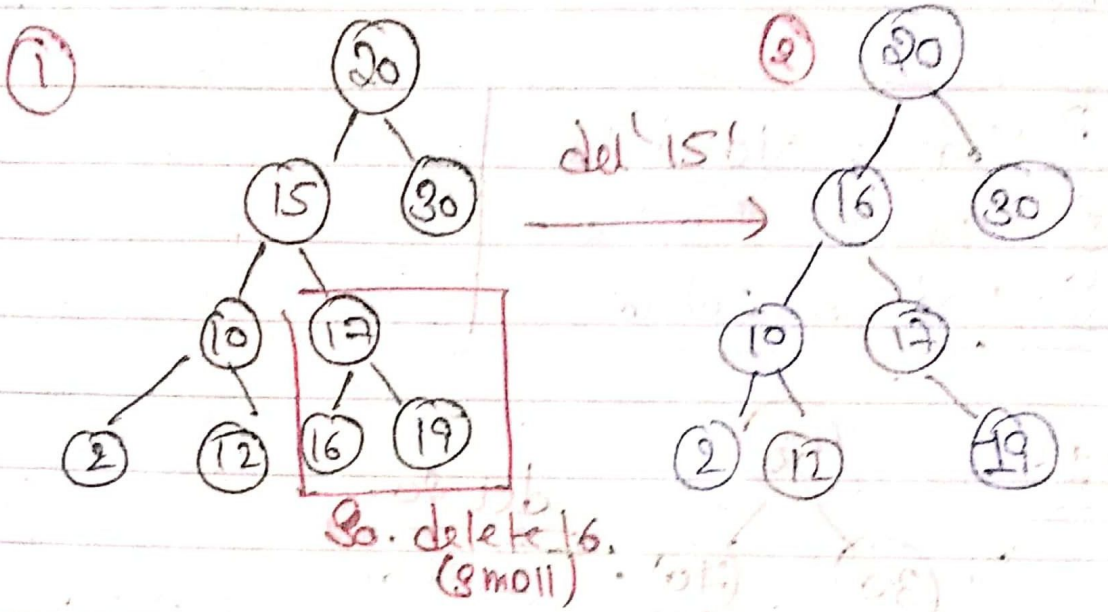
Trick:- (direct property. only for 1 child)

(So, "del 30" and put 40 in place of 30)



Just move the child as parent.

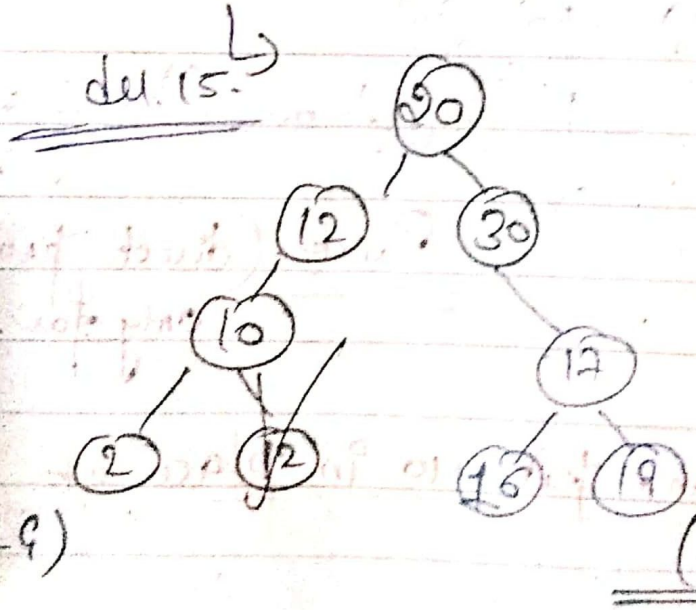
Case 3:- 2 child.



So. delete 16. (small)

(1) Visit Right subtree of deleting node, choo
"least element & replace"

(2) Visit left choose greater element (ag
del 15!



(19) node visit

Pre-order Traversal of BT using Recursion.

(1) Pre-order

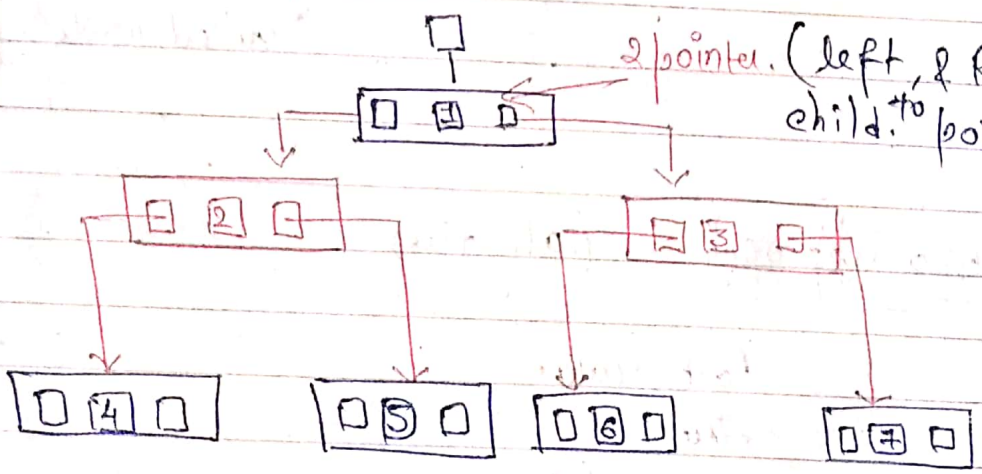
- process the root R
- traverse left subtree of R in pre-order.
- traverse the Right subtree of R in -11-

(2) Inorder:-

- traverse left subtree of R in Inorder.
- Process the root R
- traverse Right sub-tree of R in Inorder.

(3) Postorder:-

- traverse left subtree of R in postorder.
- traverse Right -11-
- Process the Root R.

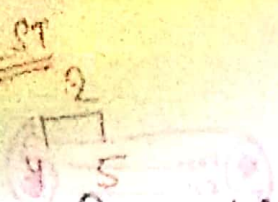


Info variable - (Data) keep inside the data.)

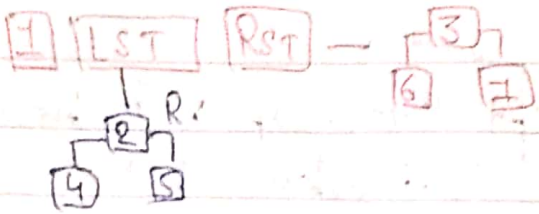
Root (pointer) → Root node

Root = Null (Then the tree is empty)

LST



Root left Right
Sub subtree.



1 2 4 5 3 6 7

Implementation (Structure to rep a Binary Tree)

```
Struct BinaryTreeNode
{
  Int info → keep data          // variable
  Struct BinaryTreeNode *left;  // 3 member
  ——  //  ——  *right;          pointer
};
```

build memory block = called node

Recursive approach (all tree)

Root value
then
Inode. (Root node ke Address)

```
void preOrder (Struct BinaryTreeNode * Root)
{
  if (pointer = Null)
```

Root

addr.

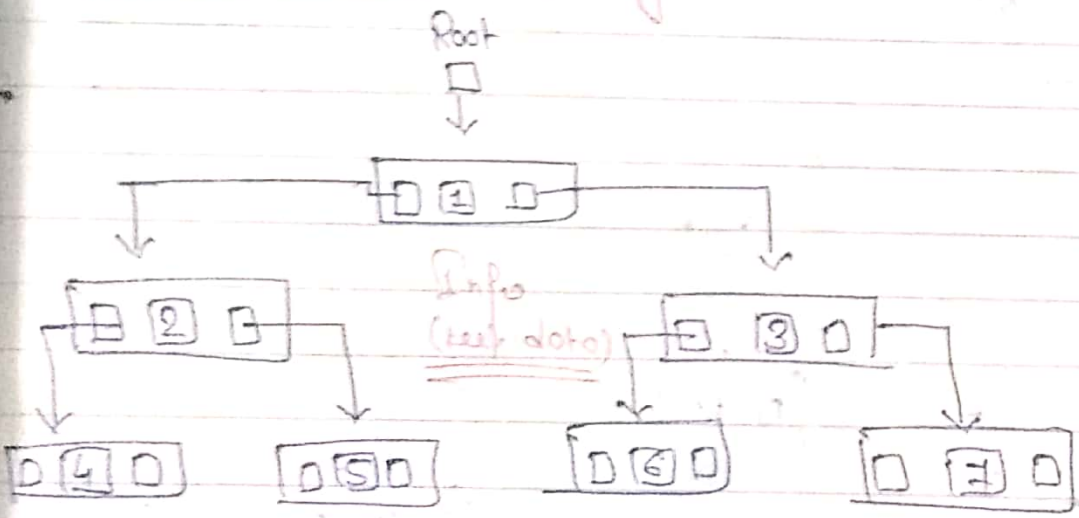
1. root = Null (no empty) (2)
   false

```
{
if (root)
{
```

1) printf ("%d", root → info); // info access
   preOrder (root → left); Root node ke data value was print.
   -11- (root → right);

```
}
}
```

2) Inorder traversal (LR Right)



4 5 2 1 6 3 7

```
void inOrder (struct BinaryTreeNode * root)
{
if
{
    inorder (root → left);   Sub-tree.
    printf ("%d", root → info);
    preOrder (root → right);
}
}
```

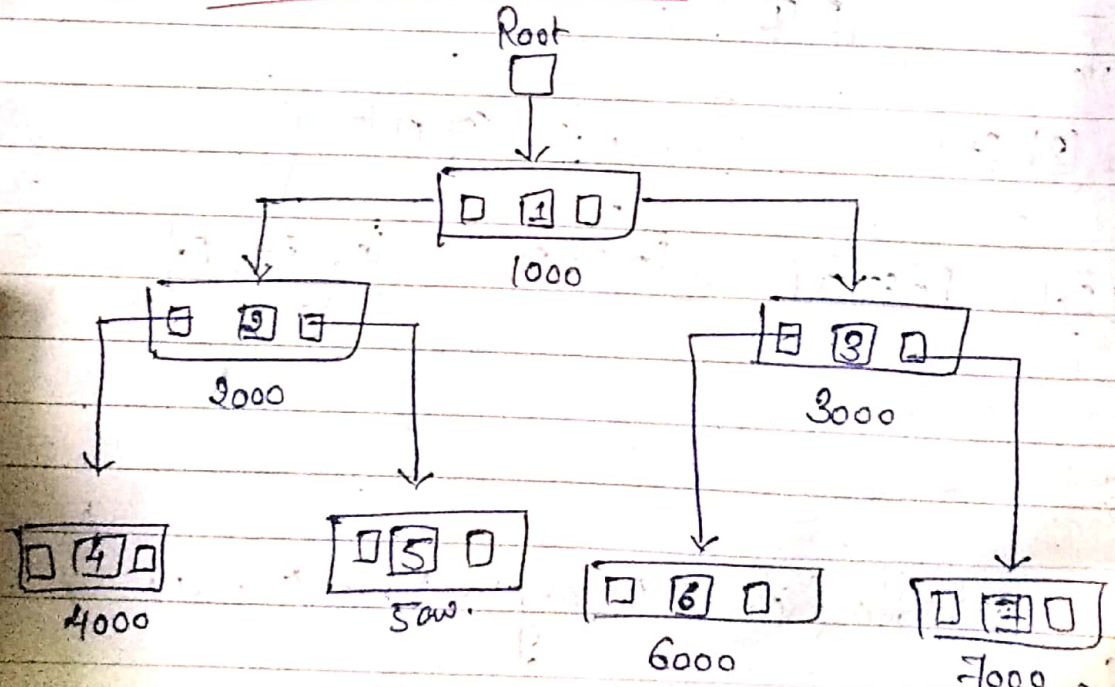
2

Postorder

4 5 2 6 7 3 1

```
void postOrder (Struct BinaryTreeNode * root)
{
  if (root)
  {
     postOrder(root → left);
     printf("%d", root → info);
     postOrder (root → right);
  }
}
```

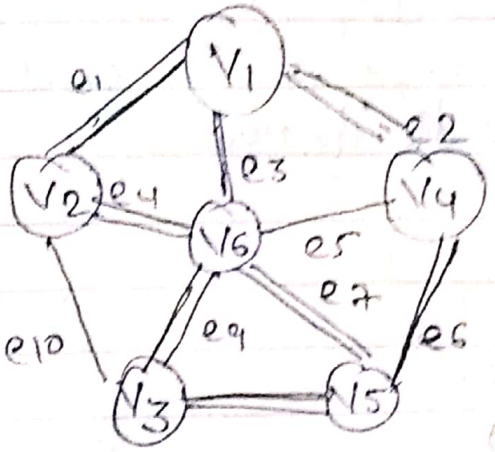
4 Level Order Traversal



Unit-4 (1)

Graph.

- Contain finite set of vertices & edges.
- denoted as $G = (V, E)$



$V_G = \{V_1, V_2, V_3, V_4, V_5, V_6\}$

$E_G = \{e_1 \; - \; - \; - \; - \; - \; e_{10}\}$.

Terms:

(1) adjacent matrix = Vi & Vj are adjacent if there is an edge b/w Vi & Vj.

$(V_1, V_4) = e_2$, (V_4, V_5)

(V_3, V_4) = not adjacent

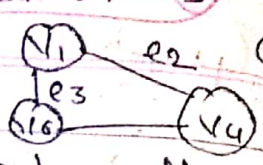
(2) Path:- Combinatn of edges.

↳ may be more than one path b/w 2 vertices.

$(V_1 \text{ —— } V_5) = \underbrace{[e_2, e_6]}_{(e_3 \to e_7)}$ n path can follow.

(2)

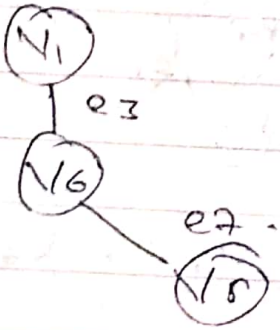
$(v_1, v_6, v_4, v_1) \rightarrow e_3, e_5, e_3$



Cycle.

Cycle :- path either 1st & last vertices are same.

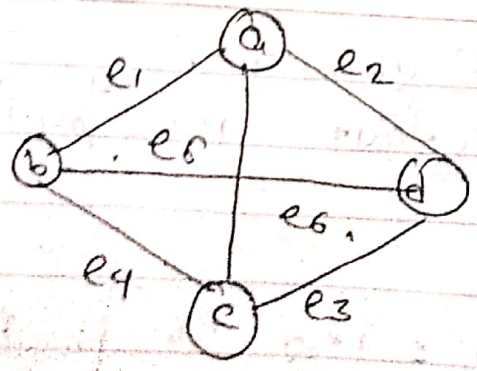
iv.) **Connected graph :-** If there is exist a path b/w any 2 of its nodes



(5.) **Complete graph :-** If every node is connected to it adjacent node.

no. of edges in a complete graph which 'n' node is

$$\frac{n(n-1)}{2}$$

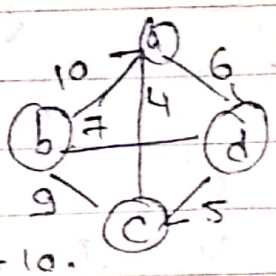


4 node

$n = 4$

$\frac{4(4-1)}{2} = 6$

(vi) Weighted graph :- If every edge in the graph is assigned some weight value.



: → find shortest path.
↳ MST

a-b = wght 10.
↳ cost is assigned for reaching node a to b.
↳ edge to wght assign hota h.

(7) - Directed graph :- If each edge is assigned a direction

BFs (Breadth-first Search) ?

algorithm

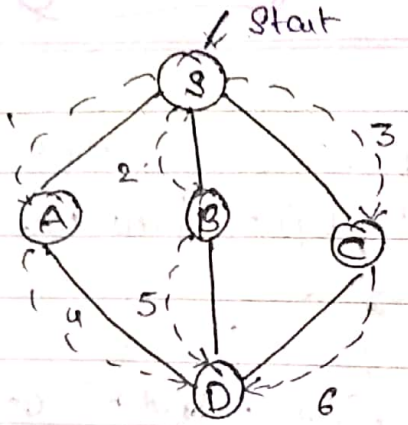
Queue.

(i) initialize.

(1.) BFs alg traverse a graph in a breadth ward motion. Traverse a graph

(2.) BFs alg uses Queue ds.

(4)



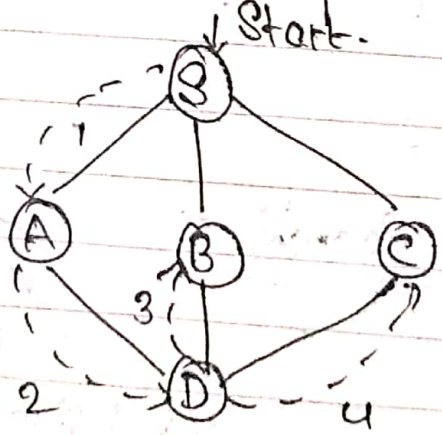
Initialize Queue.

A		
B	A	
C	B	A → dequeue

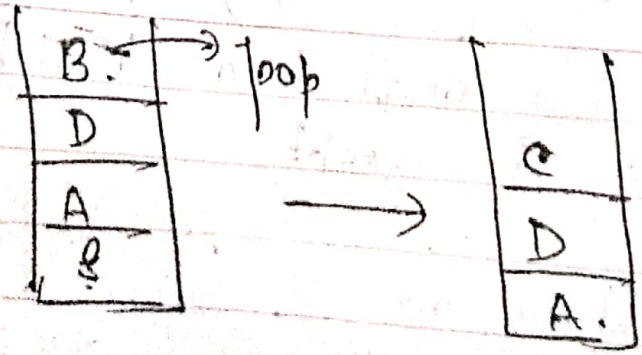
C	B	
D	C	B → dequeue.

Dfs. (Depth first Search) :-(downward)

1.) Traverse a graph in depth.
2.) Stack is used for implementing Dfs.



i) Initialize stack.



← key & data.
↓
Stored this key in the memory.

(1.) Hashing.

↳ Searching technique based on Hash Table

data.
← key → H(k) → address. (smaller in size)

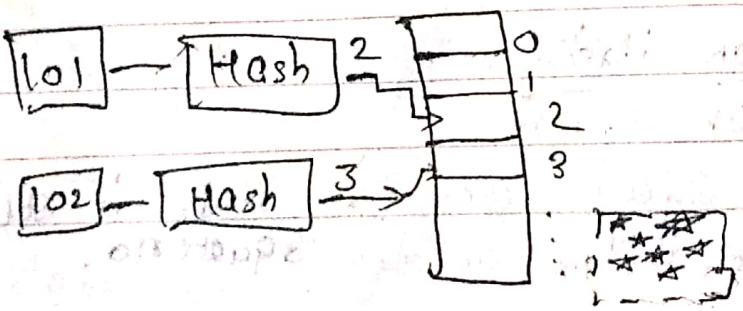
K1 → H(k)
K2 →
→ At & Collision

It is a appln of a funct^n to the key value that Result in mapping the Range of possible key value into smaller. -11- of relative addr.

Types of Hashing fn :-

(1.) Direct Hashing.

↳ No algorithmic manipulat^n.
↳ No collisn (minm no. of collisn)
↳ Limited
↳ Not suitable for large key value.



List Size = 10.

(6) (5) Key → 101 102 103 201 204 (key store in list)

Middle - Divison Hashing :-

↳ also klas Divison - Remainder.
↳ work with any list size.
↳ If list size is a prime no. than fewer collision

$$h(k) = k \bmod n$$

k → key
n → list size.

~~Mid - Square Hshing~~. $= h(10) = 101 \bmod 10$.
$= 1$

$h(102) = 102 \bmod 10$
$= 2$

$103 \bmod$

101 → 1	there are chances of collisn.
102 → 2	
103 → 3	
204 → 4.	
201 - 1	

$h(201) = 201 \bmod 10 = 1$.
↓
already pre-occupied.

3) Mid - Square Hashing. -

Middle of square

Key is Square and the addr is Selecte from the middle of the Square no.

Hash Size = 23.

then there are no collision in that case. (7)

The minm no. of field with each node of a doubly LL is.

3

Prev	Next node	Data.

The previous node pointer, the data field & the next node ---

---

→ k → k² (square ka jo value aayi h) middle.

Pblm :- Non-uniform distribution of the key.

k = 9452.

k² = (9452)².

= 89340304 = addr(4).

h(k) = 3403

Folding Hashing

fold-Shift

key value is divided into 2 part whose size matches the size of the required addrs.

fold-Boundary.

(2)

Addr.

k = 123456789 {3}

↓

123 456 789 (3 equal part divide)

↓

(add all these question.)

123
456
789

1368

↓ addr is {3} so you will discard the value 1.

→ h(k) = 368.

II fold - Boundary Hashing :-

Left & Right no are folded on a fixed boundary b/w them & the center no.

k = 123 456 789.

↓ ↓ ↓

Reverse middle Right (Reverse)

321 456 987.

↓

Sum

321
456
987

1764

h(k) = 764.

(discard the 1)