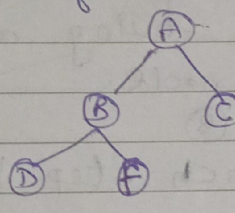
5 = Trees

* A tree is a non-linear D.S that organizes data in a hierarchical Str & this is a recursive def.
* It is a set of 1 / more nodes, with 1 node identified as the tree's root and all remaining nodes partitionable into non-empty sets, each of which is a subtree of the root.



(A) → root node.

→ Tree d.s.

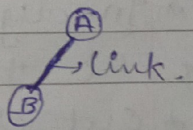
(B) (D) (E) → left subtree

(C) → right subtree.

→ Tree Terminologies =

* Node = It is a unit of data that contains a key / value as well as pointers to its child nodes.

* Edge = connecting link b/w 2 nodes. is → a link.



* Root = Topmost node in a tree. This is where the tree is originated frm.
  eg → (A)
* parent node / ancestor = It is a node that has a branch frm it to another node.
  eg → (B)

* Child node / descendant / successor =
It is a node that is a descendant of anthr node.
eg → B & C childnode of A

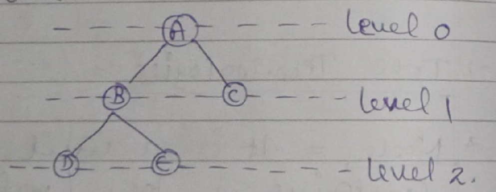
* Sibling nodes =
Nodes that have the same parent
eg → D & E.

* Leaf node = node which does not have any child. eg → D E C
(terminal node)

* Internal node = nodes hving at least a child node.
eg → A B

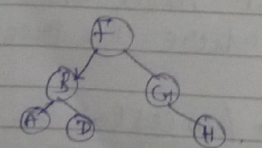
* Level of a tree = Each step frm top to bottom in a tree

eg →



* path = a sequence of consecutive edges.
* Branch = a path ending in a leaf node.

* In-degree of a node = no. of edges arriving (entering) at that node.
* Out degree of a node = no. of children it has

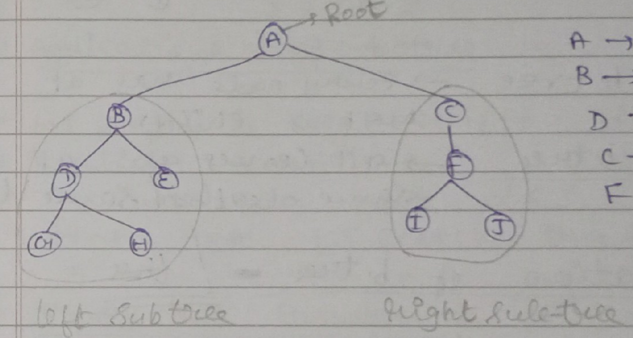


* R. node = F
* leaf node = A, D, H
* Branch. node = B, G
* B is prnt of child node A & D.
* A & D → siblings

Indegree → 1 [F → B]
Outdegree → 2. [A & D].

∴ Total degree = 3

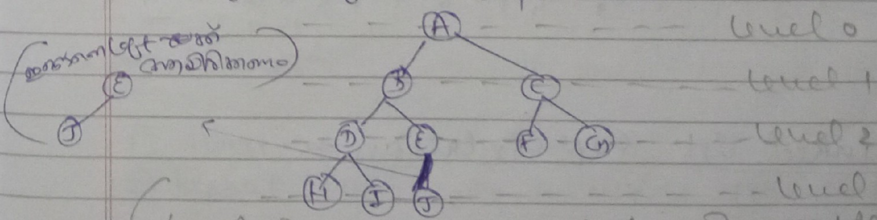
⇒ Binary tree =
* It is a sple tree where each node can hve no more than 2 children. (max 2 node)
* A non-empty b.tree consist of —
a) A node → root node
b) A left sub-tree
c) A right sub-tree.



1/2 children
A → B & C
B → D & E
D → G & H
C → F
F → I & J.

→ Types of b.tree =

a) complete b.tree =
* tree T is said to be complete if all its levels, except possibly the last, must be completely filled & all the leaf nodes must appear as far left as possible.

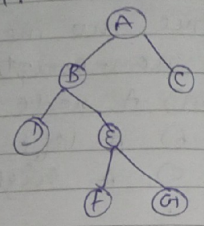


(level 3 → not completed.)

b) full b-tree = It is a spl type of tree in which the leaf nodes will have 0 children & other non-leaf nodes will have exactly 2 children.

leaf node → D, F, G, C

0 children



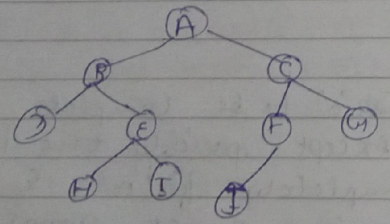
non-leaf node → A → B & C
B → D & E
E → F & G

c) rooted b-tree. → Every node has at most 2 children.

d) perfect b tree → all leaves are at same depth. / same level

→ Representation of b-tree = / tree =

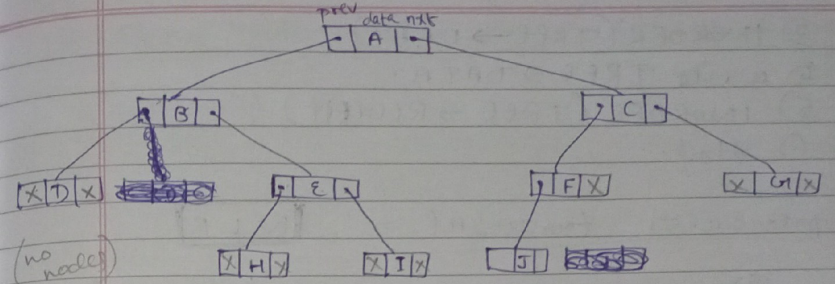
a) Array representation =



A → B C
B → D E
C → F G
D → no nodes so 2 blanks — —
E → H I
F → J —
G → — —

A	B	C	D	E	F	G	-	-	H	I	J	-	-	-
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

b) Linked list representation =



X → no nodes / children

→ Applications of b-tree =

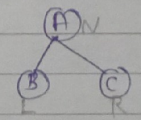
* for rapid & easy access to data.
* Routing algorithms.
* To implement heap ds,
* Syntx tree, b search tree, hash tree —

→ Binary tree traversal = (using recursion)

can be traversed in 3 different ways:-

① Inorder traversal = L N R

* Traverse left subtree of R
* process the root R
* Traverse right subtree of R.



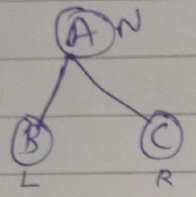
Inorder → L N R => B A C

* A1 INORDER.

1) Start.
2) Repeat stps 2 to 4 while TREE != NULL

3) INORDER (TREE → LEFT). (L)
4) write TREE → DATA. (N)
5) INORDER (TREE → RIGHT) (R)
6) Stop.

② pre-order traversal = N L R

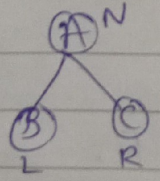


N L R
→ A B C ⇒ A B C

* (A1) PREORDER.

1) Start.
2) Repeat stps 2 to 4 while TREE != NULL
3) write TREE → DATA. (N)
4) PREORDER (TREE → LEFT) (L)
5) PREORDER (TREE → RIGHT) (R)
6) Stop.

③ post-order traversal = L R N



→ L R N ⇒ B C A

* (A1) POSTORDER.

1) Start.
2) Repeat stps ....
3) postORDER (TREE → LEFT) (L)
4) postORDER (TREE → RIGHT) (R)
5) write TREE → DATA. (N)
6) Stop.

→ Binary tree traversal = (without recursion)

a) pre-order (t) =

1) start with root node & push onto stack
2) Repeat while stack is not empty -
   - pop the top element (PTR) frm stack. & process the node.
   - PUSH right child of PTR onto a stack.
   - PUSH left child of PTR onto a stack.

b) In-order (t) =

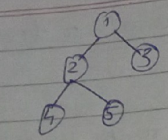
1) start frm root, call it PTR.
2) PUSH PTR onto stack if PTR is not null
3) move to left of PTR & repeat stp 2.
4) if PTR is null & stack is not empty then pop element frm stack & set as PTR.
5) process PTR & move to right of PTR, go to stp 2.

c) post-order (t) =

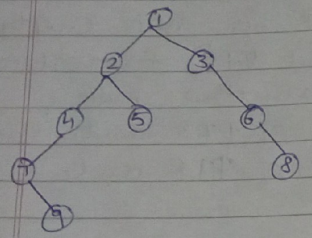
1) push root node to stack one
2) repeat stps while 1st stack is not empty.
   a) pop a node frm 1st stack & push it to and stack.
   b) push left & right children of popped node to 1st stack.
3) print contents of and stack.

* inorder =

(LNR)



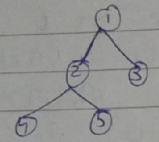
→ 4→2→5→1→3



→ 7→9→4→2→5→1→3→6→8.

* preorder =

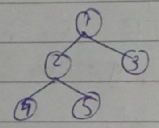
(NLR)



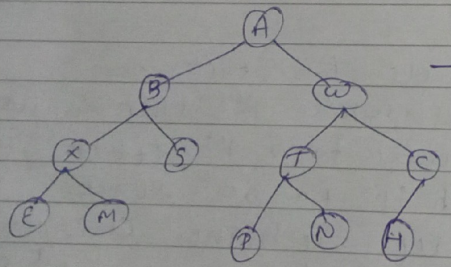
→ 1→2→4→5→3

* postorder =

(LRN)



→ 4→5→2→3→1.

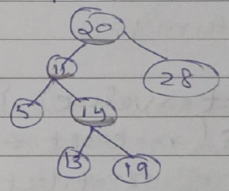


→ E M X S B P N T H C W A.

⇒ Binary Search tree (BST) =

* It is a spl b.tree used for efficient storage of data.
* The nodes in BST are arranged inorder.
* It allows for the simpleing of dlt of items.
* It is a b.tree where each node can have a max of 2 children.
* It has follow (pro) →
  a) All the elemnts in left subtree is less than the root node.
  b) All the elemnts in right subtree is > than root node.

eg →



→ BST

→ operations =

(a) search =

1) if root = NULL
   return null.
2) else
   if root → data = x
   return root.
3) else
   if x < root → DATA.
   return search (ROOT → LEFT, x)
   else.
   return search (ROOT → RIGHT, x)

① insertion =

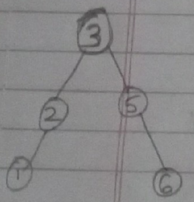
1) if TREE = NULL
2) Allocate memory for ~~FREE~~ TREE.
3) set TREE → DATA = VAL
4) set TREE → LEFT = TREE → RIGHT = NULL
5) else
   if VAL < TREE → DATA.
   insert (TREE LEFT, VAL)
   else
      insert (TREE RIGHT, VAL)

(or)

```
insert(value, root) {
    if (root == NULL) {
        root = new node(value);   3
        return;
    }
    if (root.data > value) {
        if (root.left == NULL) {
            root.left = newnode (value);
            return;
        }
        insert(value, root.left);
    }
    if (root.data < value) {
        if (root.right == NULL) {
            root.right = newnode (value);
            return;
        }
        insert (value, root.right);
}.
```

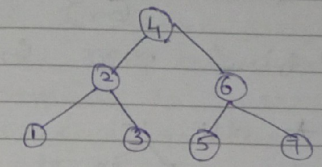
① 3
② 2
③ 1
④ 5
⑤ 6

(prgm)



5 cases

② Create BST with 4, 2, 3, 6, 5, 7, 1
[illegible] root node [illegible]

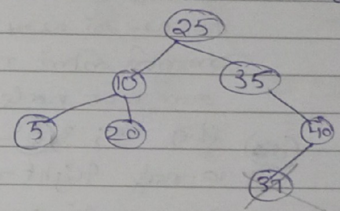


③ Deletion =

3 cases

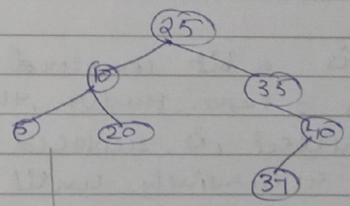
case1 → dlt leaf node

It is the simplst case, here, replace the leafnode with NULL & simple free the allocated spce



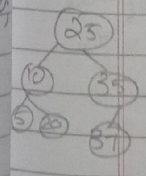
If we want to dlt 37, jst dlt it.

case 2 → dlt a node with 1 child.
* go to parent node of targeted node.
* replace the targeted with node with its child node.

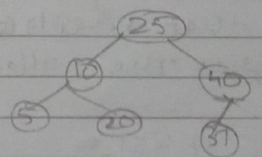


If we want to dlt 35
① parent node of 35 → 25
then dlt 35
② so 25 links to 40.

dlt 40 →



eg - ②

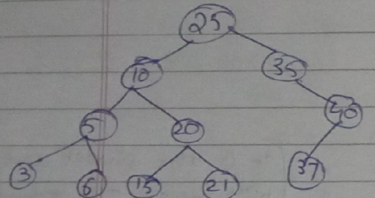


eg - ①

In this case-② replace the node with its child node, which row containing the value which is to be dlted, simply replace it with the NULL & free allocated space.

case3 = dlt a node with 2 child.

2 ways { a) find min in right (or) b) find max in left.

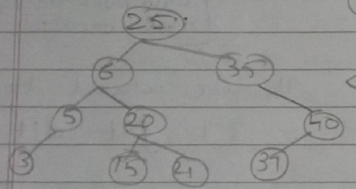


* dlt 10.

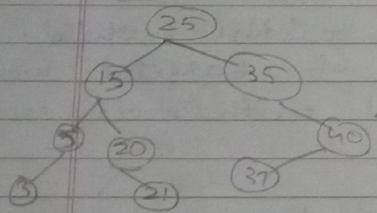
10
5 20.

after dlting 10, we can set it in 2 ways-

ⓐ ① 10-ന്റെ left-ലെ nodes നെ നോക്കുക, അതിൽ നിന്ന് max value നെ root node ആക്കുക.
(i.e) 5, 3, ⑥ → root node.



ⓑ ② 10-ന്റെ right-ലെ nodes നെ നോക്കി min value-നെ root node ആക്കാം.

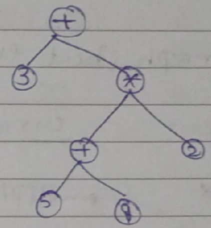


Here, it is a bit complexed case compare to other 2 cases. However, the node which is to be dlted, is replaced with its inorder successor recursively until the node value is placed on the leaf of the tree. After that, replace the node with NULL & free the allocated space.

=> Applications of BST =
1) Implement various searching algorithms.
2) For dynamic sorting.

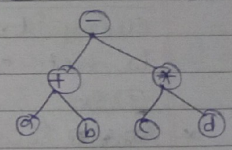
=> Expression tree =
* It is a sple type of b.tree that is used to store algebraic exp.
* In an exp tree, each internal node corresponds to operator & each leaf node corresponds to the operand.
* ~~A node that holds a non is a leaf node~~
* eg → ① $3+(5+9)*2$

→



② $x=(a+b)-(c*d)$

→



```
void insert (int)
struct node {
    int data;
    struct node *left;
    struct node *right;
};
struct node *root;
```