Tree

Defn: It is a nonlinear D.S. which shows a hierarchical relationship among the data items.

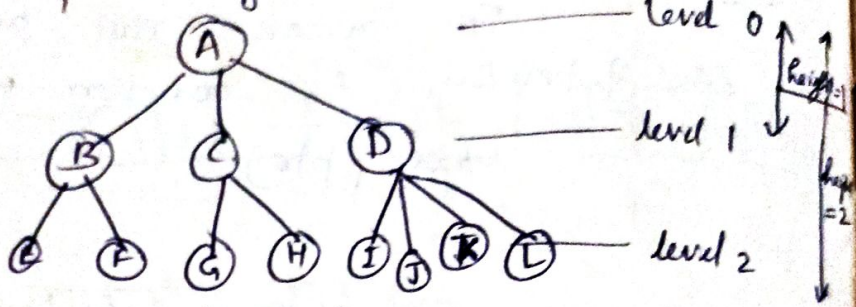


fig: A tree

Basic Terminologies:

Root – A (first item in hierachical arrangement)

Node – A, B, C, D, E, F -- L (Each item is a node)

Degree of node – It is No. of subtree in a given tree

The degree of node A = 3
B = 2
C = 2
D = 4
E = 0

Degree of a tree – It is Max degree of nodes in a given tree

Here degree of node D is max(4)
So degree of tree = 4

Terminal node (or leaf node or external node) – A node with degree zero (no further branches)

Ex. E F G H, I, J, K, L

Non-terminal node (or internal node) – A node whose degree is not zero (except root node)
i.e. It is intermediate node between root & leaf

Ex. B, C, D

Non-linear DS - tree, graph ... in linear fashion ... usually

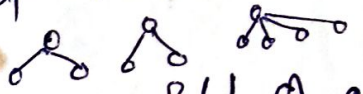
Siblings (Brothers) — The children node of a given parent node are called siblings.

Eg. E, F are siblings in parent node B
G, H ——— C
I, J, K, L ——— D

Level — Root node — level 0
immediate children — level 1
their intermediate children — level 2 ....

Edge/branch — It is a connecting line of 2 nodes

Forest — It is a set of disjoint tree.
If root node is removed, it would be a forest.



Height/depth — The h/d of a tree is the max level of any node in a given tree.
i.e. no of levels, one can descend the tree from its root to the leaves
Here Height of the tree = 2

Path — It is a sequence of consecutive edges from some node to destination node.

Binary Tree

In B.T. max degree of any node is at most two.

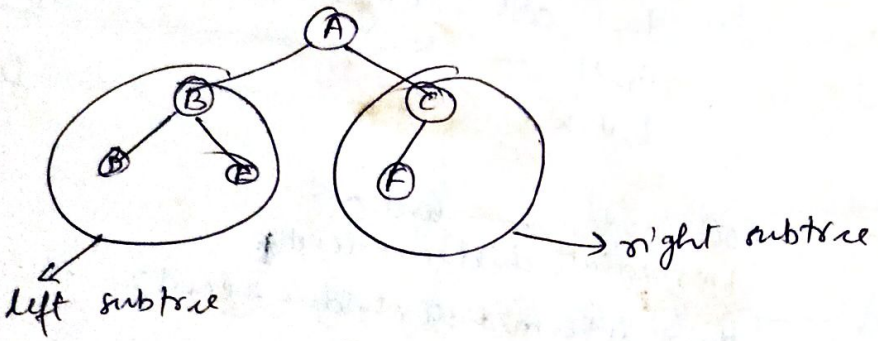


fig: BinaryTree

Strictly Binary tree -

If every non-terminal node consist non-empty left & right subtree.

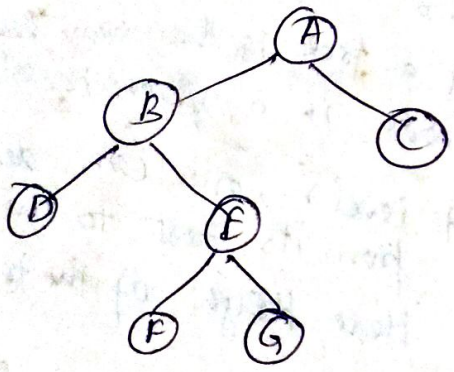
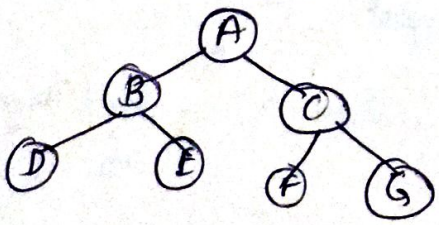


fig: A Strictly Binary tree

complete Binary Tree -



Skewed Binary Tree

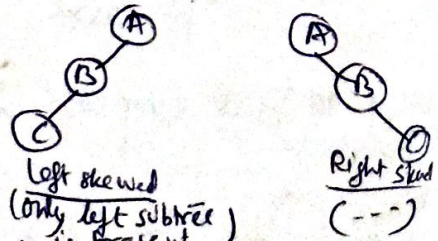
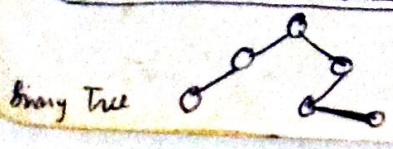
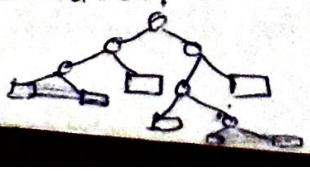


fig: A complete Binary tree with depth 2

Extended Binary tree / 2-tree

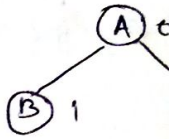
: If each node has either 0 or 2 children.

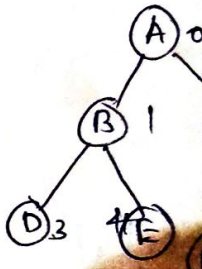




Binary T

① Array Rep^n





Array rep^n is not suit unnecessary

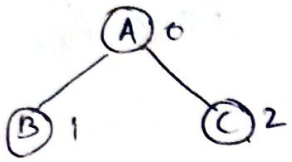
3

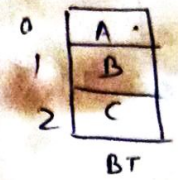
1. Linear DS - Array, Linked list, Stack, ...

NC - Tree, Graph

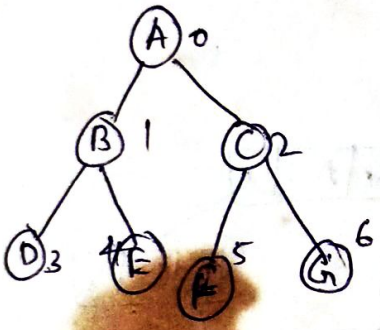
Binary Tree Representation :- (8.4)

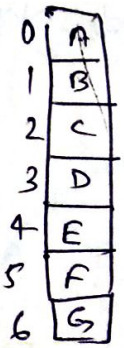
(1) Array Rep^n :-



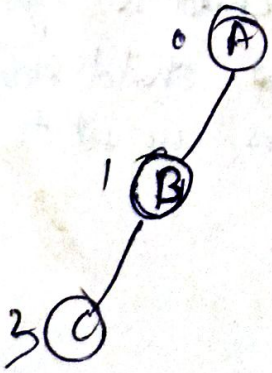


BT[0] = A
BT[1] = B
BT[2] = C

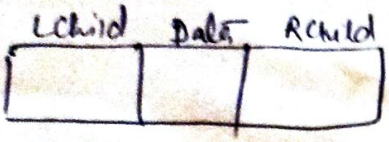




Array rep^n is better for complete Binary Tree. But it is not suitable for others because it results in unnecessary wastage of memory space. for Ex -



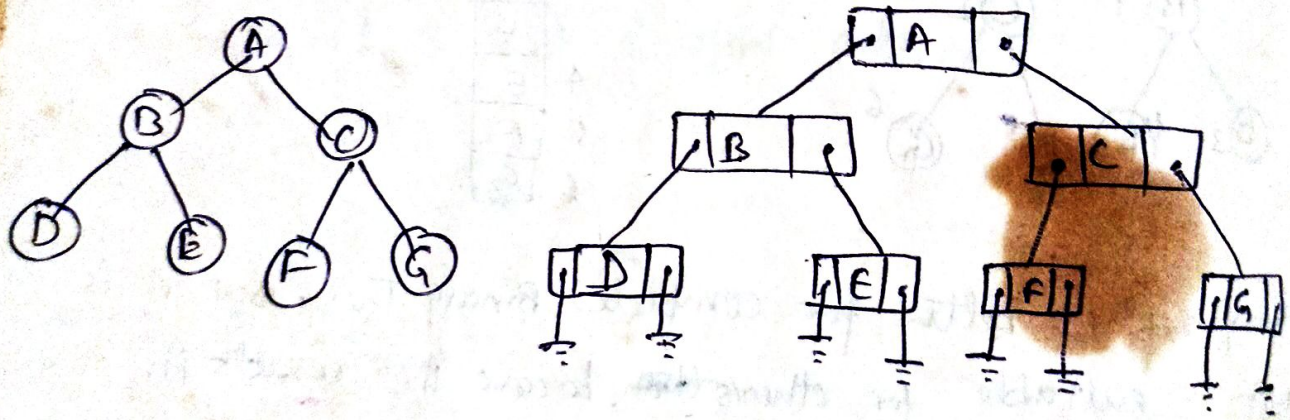
(2) Linked List Repn.



A node of a BT

Lchild = left child
Rchild = Right child

```
struct node
{
    char data;
    struct node * lchild;
    struct node * rchild;
};
typedef struct node NODE;
```



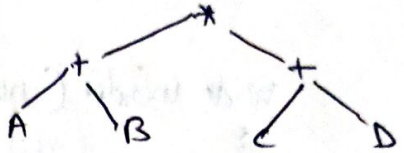
Root & Non-terminal nodes have their left child & right child nodes.
Terminal nodes don't have left or right child nodes.
So their lchild & rchild pointers are set to NULL.

Algebraic Expressions :-

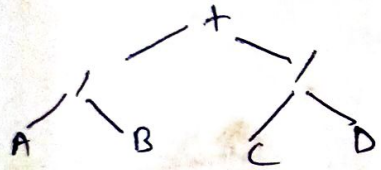
Conversion of an algebraic expression into BT is done according to order of precedence (BODMAS).

① A+B

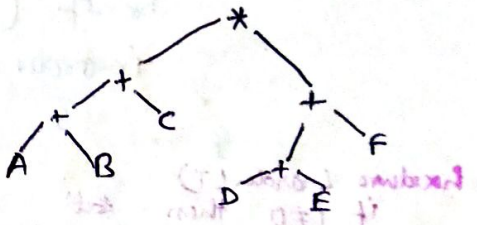
② (A+B) * (C+D)



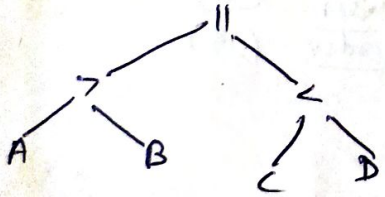
③ A/B + C/D



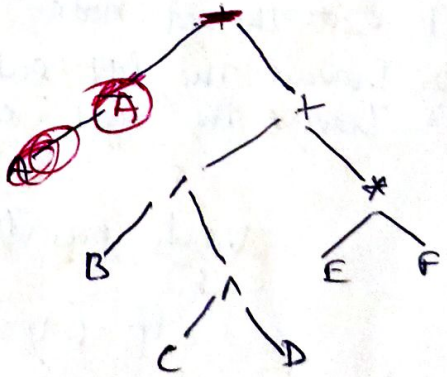
④ (A+B+C) * (D+ E+ F)



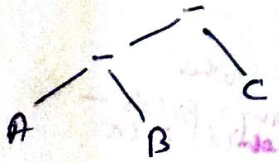
⑤ (A>B) || (C<D)



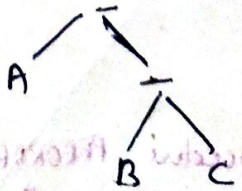
⑥ A - B/(C^D) + (E* F)



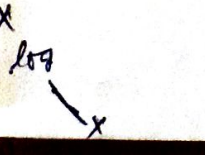
⑦ (A-B) - C



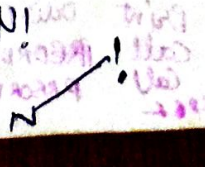
⑧ A - (B-C)



⑨ log x



⑩ N!



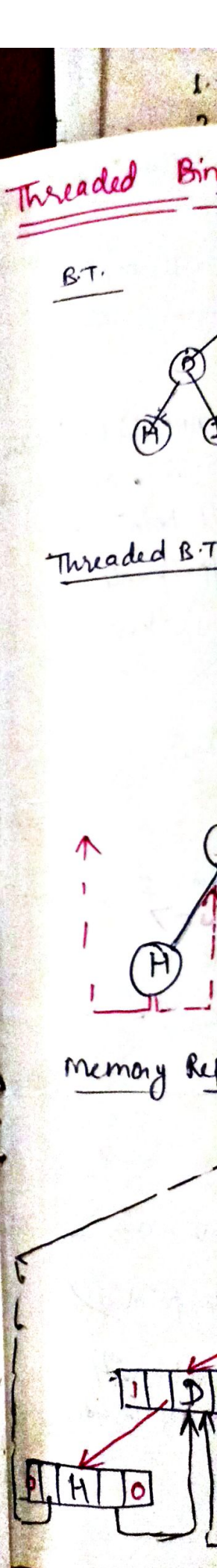
Traversal of B.T. (Non-Recursive : using stack)

```
Procedure INORDER(T)
    i←0                          // initialize stack
    Loop
      while T≠0  do
                 i←i+1
                 if i>n  then Print "STACK FULL"

                 STACK(i)←T
                 T←LChild(T)

      end

      if i=0 then return
      T← STACK(i)
      i←i-1
      Print(DATA(T))
      T←RChild(T)
    forever
end INORDER
```

Procedure Inorder(P)

1. P←ROOT
   i←0
2. Proceed down the leftmost path, ~~no~~ pushing each node onto STACK
3. STOP when a node with no left child is pushed
4. (9) POP & Process the nodes on STACK until NULL is popped
5. Print 'DATA' , if appears
6. If RChild node ~~is processed~~ appears then process it also
7. Follow the ~~above~~ process to rightmost path also

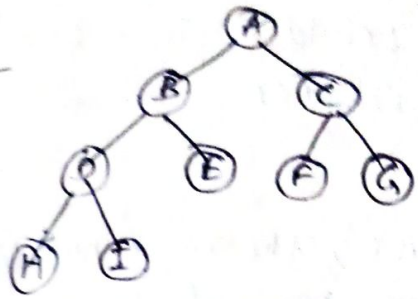
End



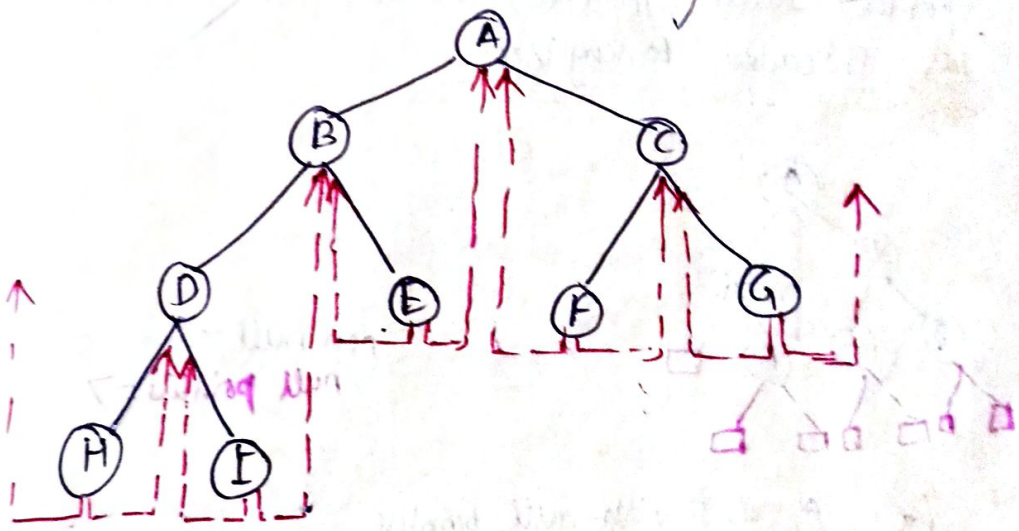
1. Linear DS - Array, Linked ...
2. Non-linear DS - Tree, graph ... in linear fashion ... sequentially

Threaded Binary Tree:-

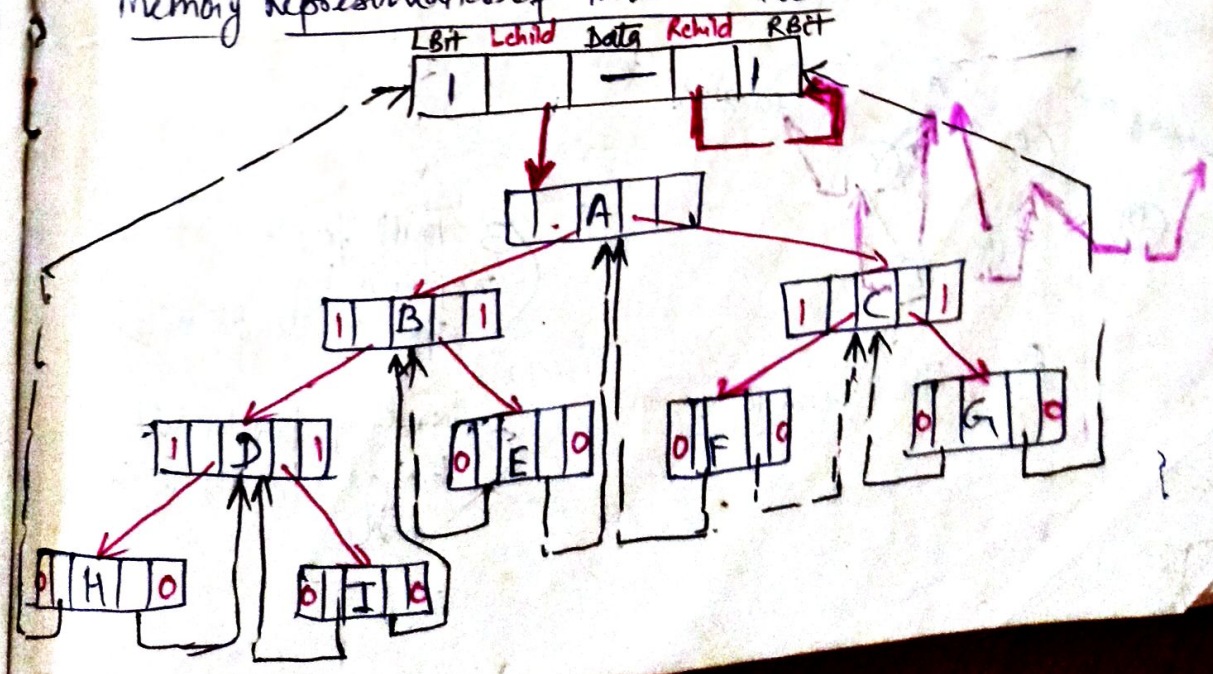
B.T.



Threaded B.T.



Memory Representation of Threaded Tree



Traversal of a BT (Recursive)

(A) Inorder Traversal

1. Traverse the left subtree in inorder (L)
2. Visit the root node (N)
3. Traverse the right subtree in inorder (R)

```
void inorder (node *P)                    // name of root
{
  if (tree != NULL)
    {
    inorder (P->left);
    printf ("%d\n", P->num)
    inorder (P->right);
    }
}
```

```
Procedure Inorder(T)
  if T≠0 then
      call INORDER (LCHILD(T))
      Print (DATA(T))
      call INORDER (RCHILD(T))
end Inorder
```

(B) Preorder Traversal -

1. Visit the root node (N)
2. Traverse the left subtree in preorder (L)
3. Traverse the right subtree in preorder (R)

```
Void preorder (node *P)
{
  if (tree != NULL)
    {
    printf ("%d\n", P->num);
    preorder (P->left);
    preorder (P->right);
    }
}
```

```
Procedure PREORDER (T)
  if T≠0 then
      Print DATA(T)
      Call PREORDER (LCHILD(T))
      Call PREORDER (RCHILD(T))
end PREORDER
```

// T = Binary Tree where each node has 3 fields — Lchild, Rchild, Data

1. Linear DS - Array, Linked list, stack, Queue

.. -DS - Tree, graph

(c) Postorder Traversal -

1. Traverse the left subtree in postorder (L)
2. Traverse the right subtree in postorder (R)
3. Visit the root node (N)

```
void postorder (node * P)
{
    if (tree != NULL)
    {
        postorder (P->left);
        postorder (P->right);
        printf ("%d \n", P->num);
    }
}
```

Procedure Postorder (T)
if T ≠ 0 then
— call Postorder (LCHILD(T))
— call Postorder (RCHILD(T))
— Print (DATA(T))
end Postorder

Ex.

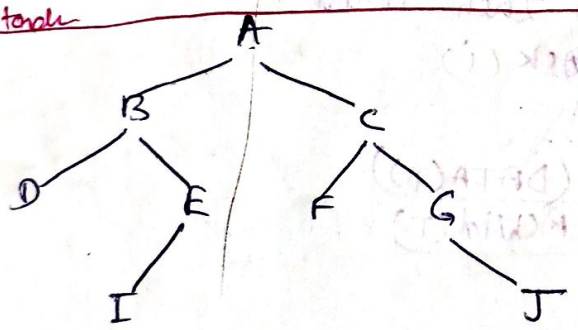


fig: A BT

Inorder - D B I E | A | F C G J
(L) | Root | (R)

Preorder - A | B D E I | C F G J
Root | L | R

Postorder - D I E B | F J G C | A
L | R | Root

Traversal of B.T. (Non-Recursive : using stack)

```
Procedure INORDER(T)
        i←0                          // initialize stack
        Loop
          while T≠0  do
                     i←i+1
                     if i>n  then Print "STACK FULL"

                     STACK(i)←T
                     T←LChild(T)

          end

          if i=0 then return
          T← STACK(i)
          i←i-1
          Print(DATA(T))
          T←RChild(T)
        forever
end INORDER
```

```
Procedure Inorder(P)
```

① P←ROOT
   i←0

② Proceed down the leftmost path, pushing each node onto STACK

③ STOP when a node with no left child is pushed

④ POP & Process the nodes on STACK until NULL is popped

⑤ Print 'DATA'

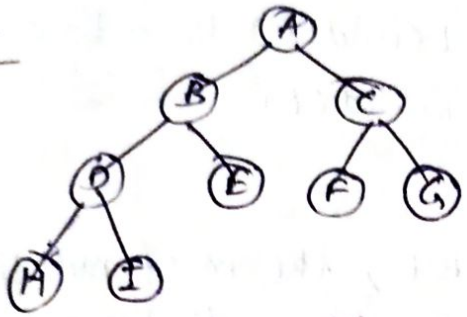
⑥ If RChild node appears, then process it also

⑦ Follow the above process to rightmost path also

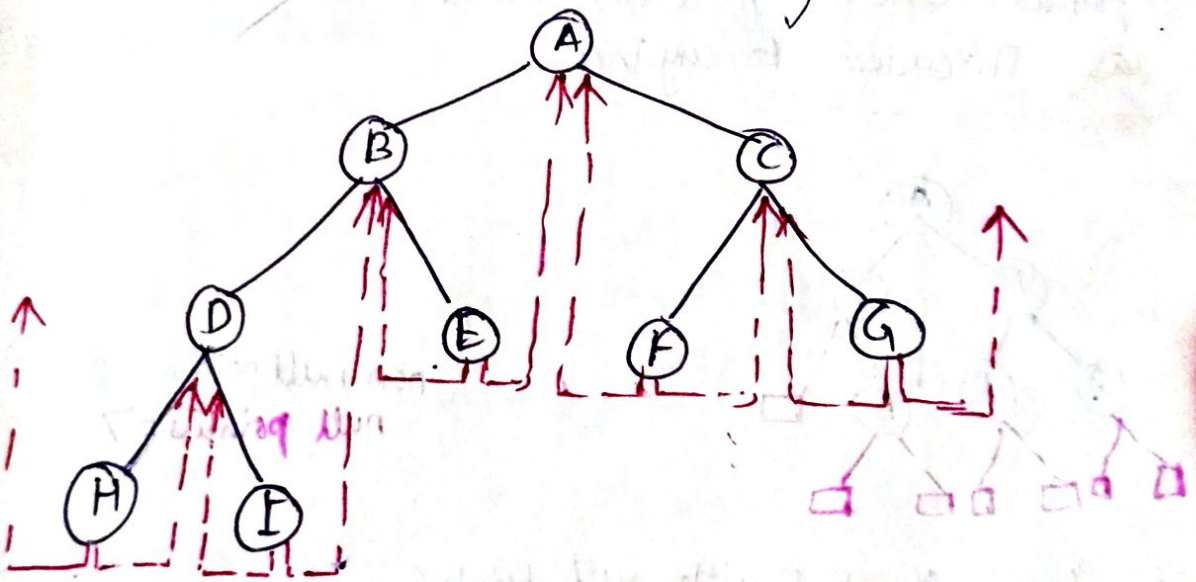
End

Threaded Binary Tree:-

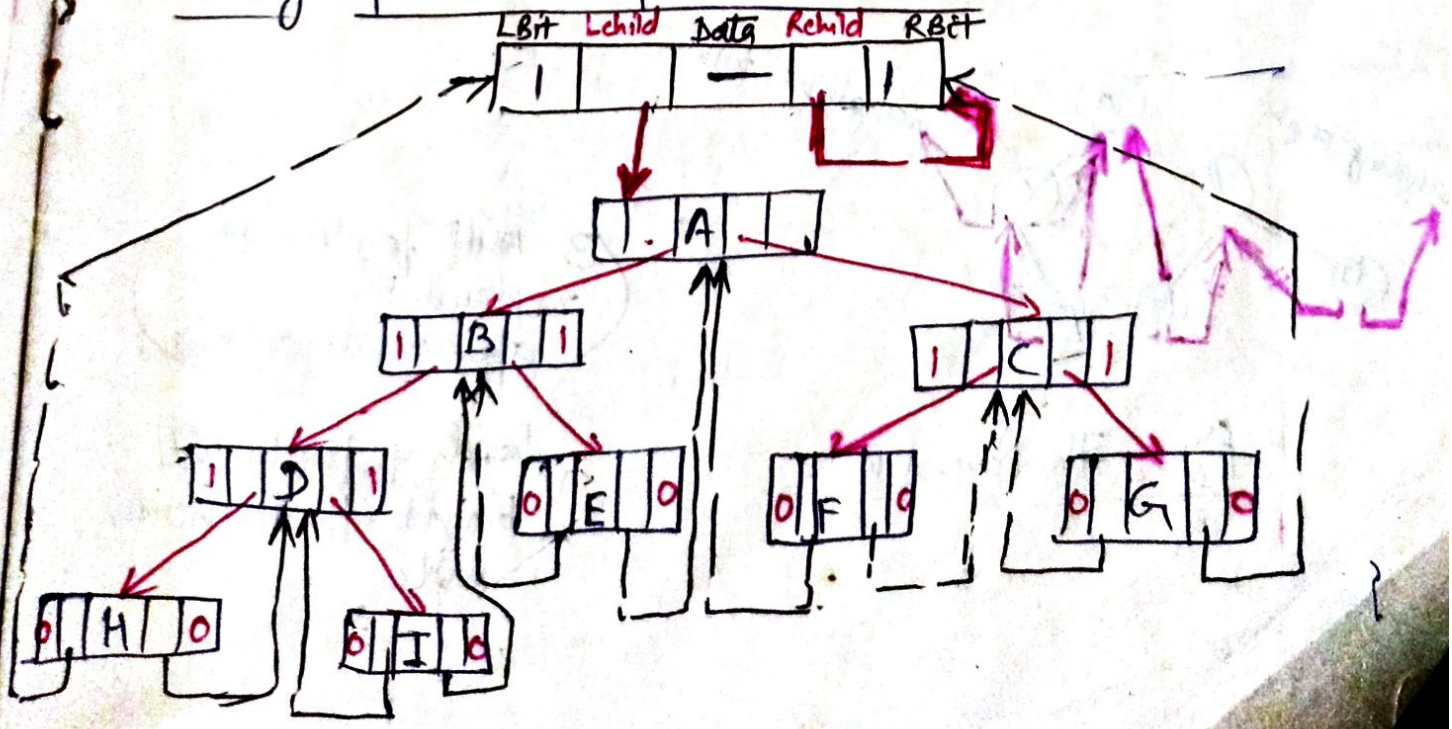
B.T.



Threaded B.T.



Memory Representation of Threaded Tree



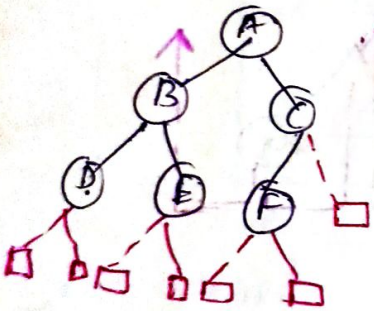
LBit (P) = 1 if Lchild (P) is a normal pointer

RBit (P) = 1 if Rchild (P) ———

LBit (P) = 0 if Lchild (P) is a thread

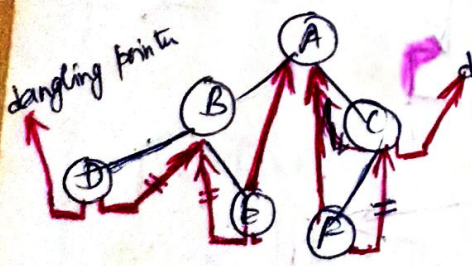
RBit (P) = 0 if Rchild (P) ———

In linked list rep^n of BT, the no of null links/pointers is actually more than non-null pointers. So to make an effective use of these null pointers, replace all the null pointers by the appropriate pointer called Threads. Such tree is known as Threaded Binary Tree.



non null pointers = 5
null pointers = 7

fig: A B.T. with null pointers



(3 null pointers are replaced by appropriate pointers)

(dangling pointers will point to the header node)

fig: Threaded B.T.

Traversing T

Procedure INSUCC (x)
S ← Rchild(x)
if RBit(x) = 1

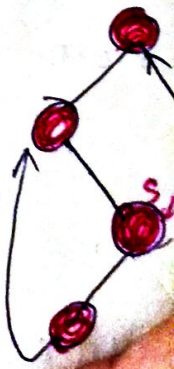
return (S)
end INSUCC

Procedure TINO
Head ←
loop

forev
end TINORDE

Insertion

(1)



1. Linear DS - Array, Linked list, Stack, ...

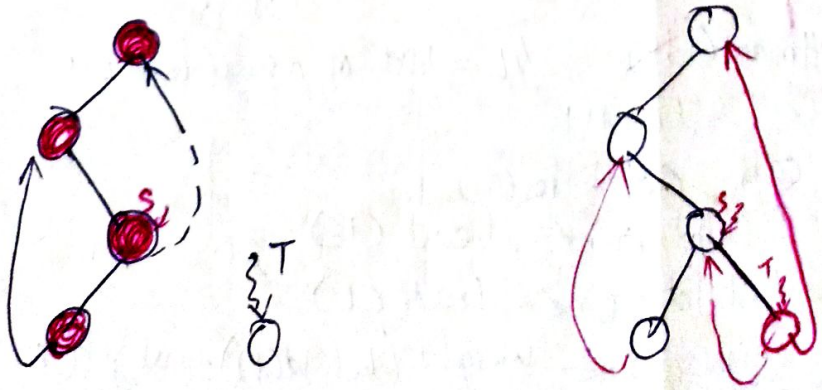
Traversing Threaded B.T. :-

```
Procedure INSUCC (x)          // Finding the inorder successor of x
    S ← Rchild(x)
    if RBit(x) = 1  then
        while LBit(S) = 1
            S ← Lchild(S)
    return(S)
end INSUCC
```

```
Procedure TINORDER(T)         // Traverse the threaded B.T. 'T' in inorder
    Head ← T
    loop
        T ← INSUCC(T)
        if T = Head then return
        Print(DATA(T))
    forever
end TINORDER
```

Insertion of 'T' as a Right child of 'S' in a Threaded B.T.

(1)



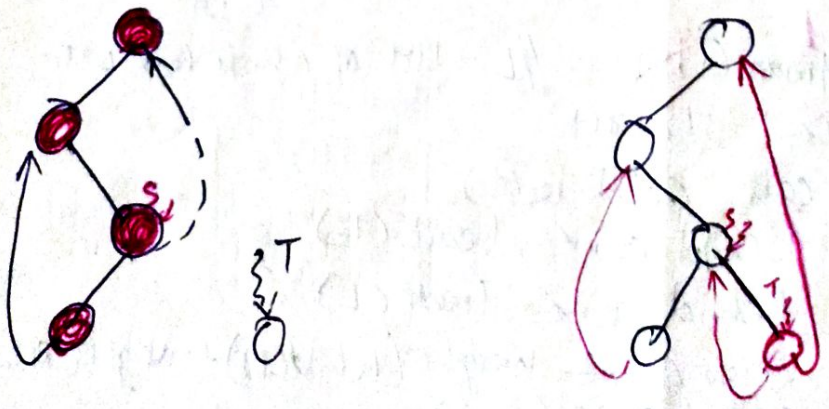
Traversing Threaded B.T. :-

```
Procedure INSUCC (x)        // Finding the inorder successor of x
    S ← Rchild(x)
    if RBit (x) = 1 then
                while LBit (S) = 1
                        S ← Lchild (S)
    return (S)
end INSUCC
```

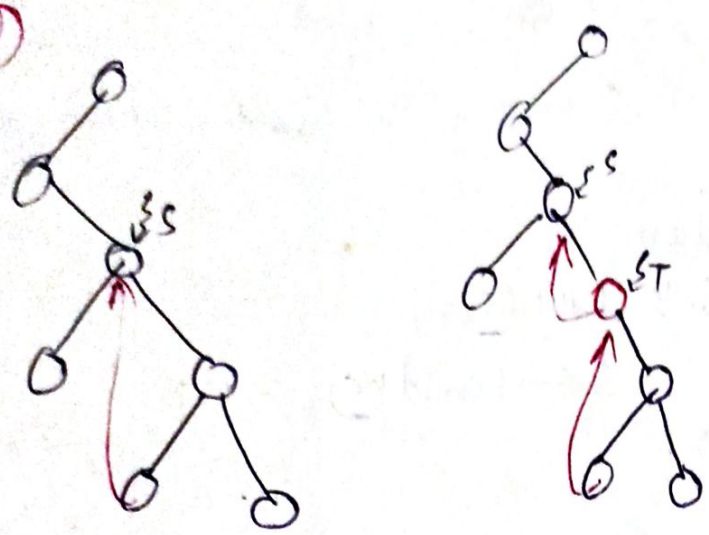
```
Procedure TINORDER (T)      // Traverse the threaded B.T. 'T' in inorder
        Head ← T
        loop
            T ← INSUCC (T)
            if T = Head then return
            Print (DATA(T))
        forever
end TINORDER
```

Insertion of 'T' as a Right child of 'S' in a Threaded B.T.

(1)

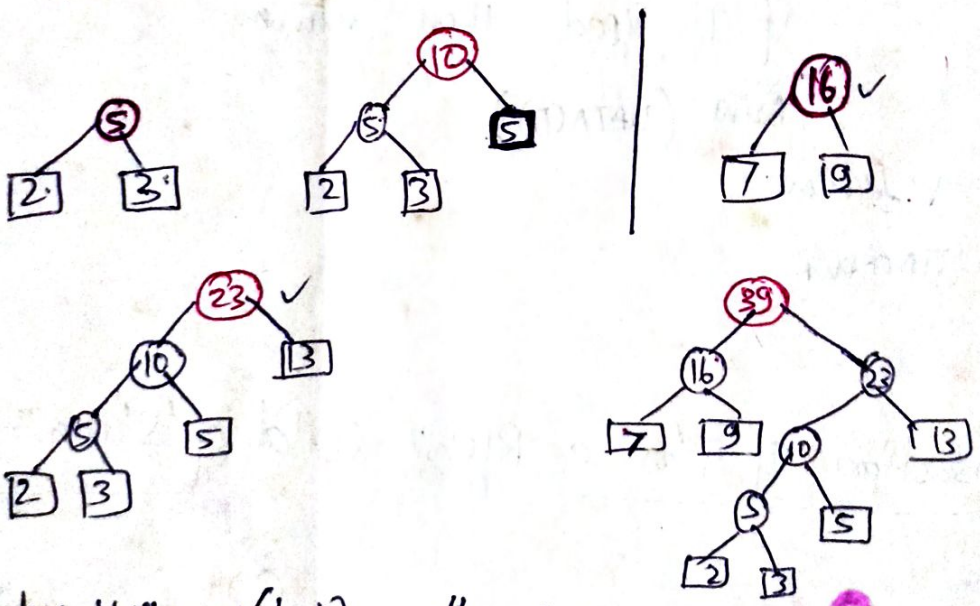


(2)



Huffman Algorithm :—

Let L = 2, 3, 5, 7, 9, 13.




```
Procedure Huffman (L, n)      //L = list of n single node B.T.
    for i ← 1 to n-1
        call GetNode (T)
        LChild (T) ← Least (L)
        RChild (T) ← least (L)
        Weight (T) ← Weight (LChild(T)) + Weight (RChild(T))
        call INSERT (L, T)
end
```

end Huffman

BST

complexity of searching algo:—

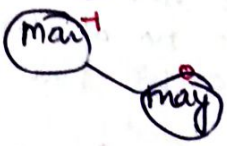
= O(logn)

[But it may reduced to a linear search [O(n)] when the tree would be unbalanced]

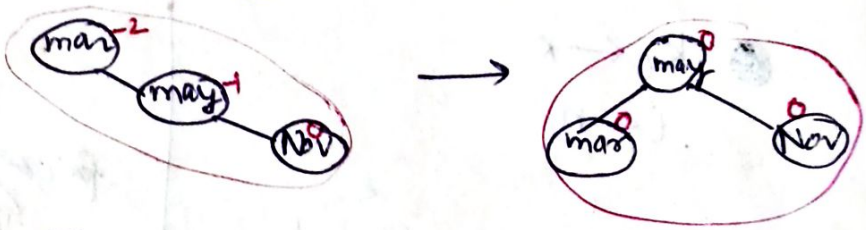
AVL Tree (Height Balanced Binary Tree) :- Adelson-Velskii & Landis (1962)

March

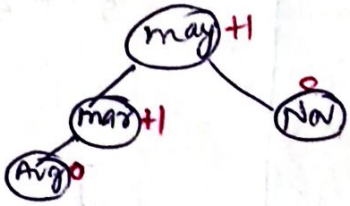
May



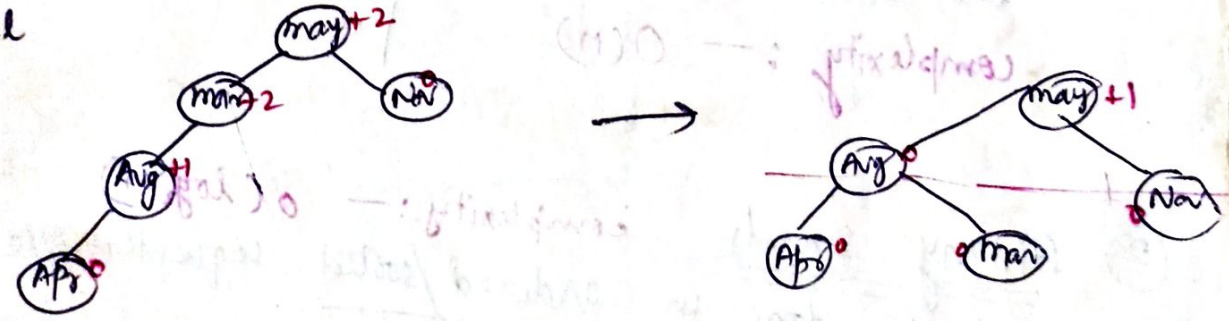
Nov.



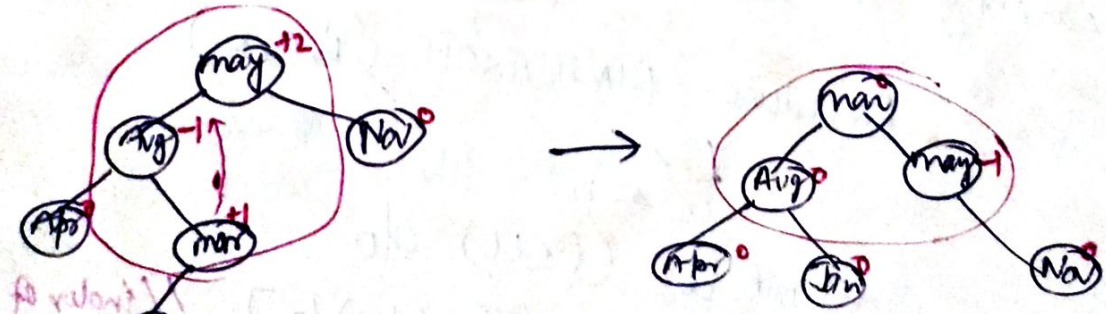
Avg



April



Jan

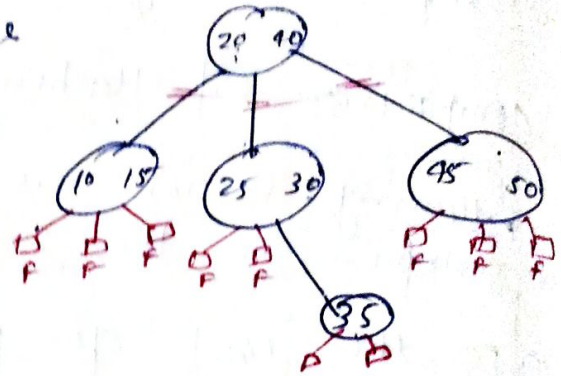


Dec

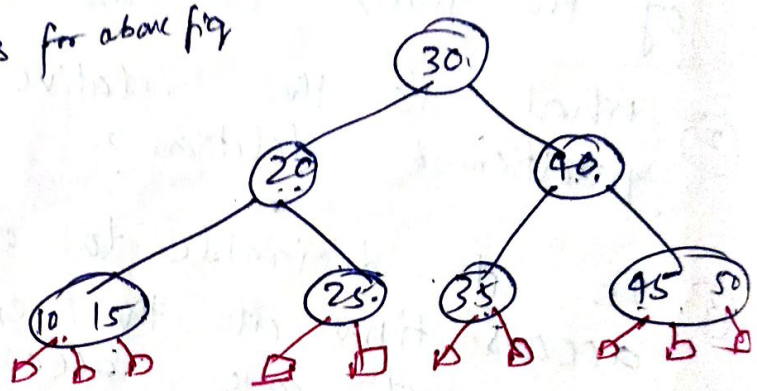
B+ Tree Index files / B Tree index file

B Tree Indexing –

A - 3 way search Tree



B Tree of order 3 for above fig

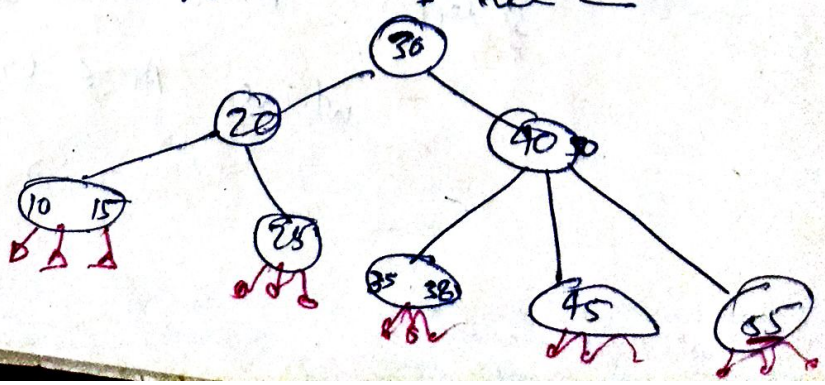


B Tree of order M is a M-way search Tree which satisfies following things

1. The root has at least 2 children
2. All nodes other than root & failure nodes have at least $(m/2)$ children
3. All the failure nodes are at the same level.

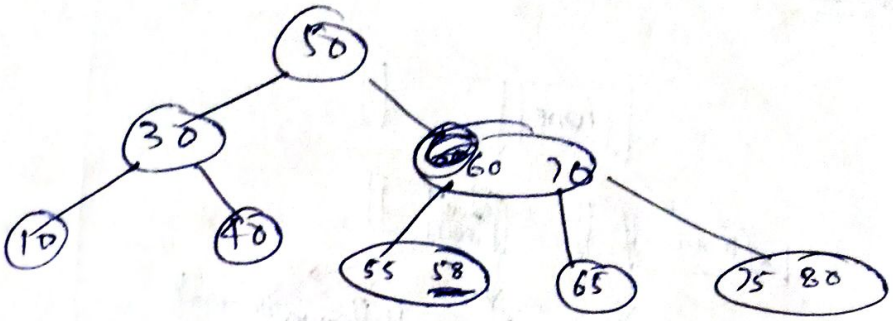
← Insert

Insertion of 55 into, above B-Tree –

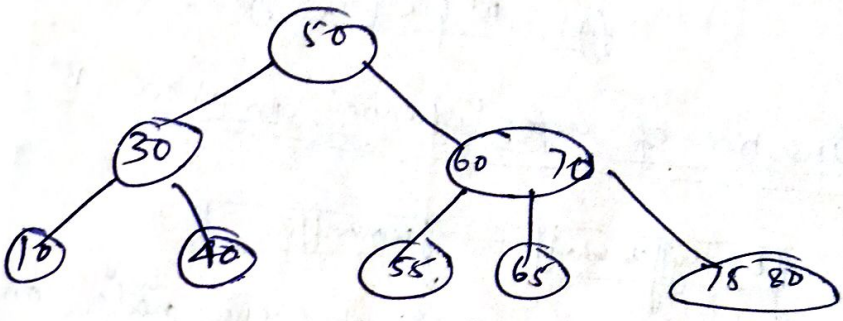


particular org. of data is called DS.

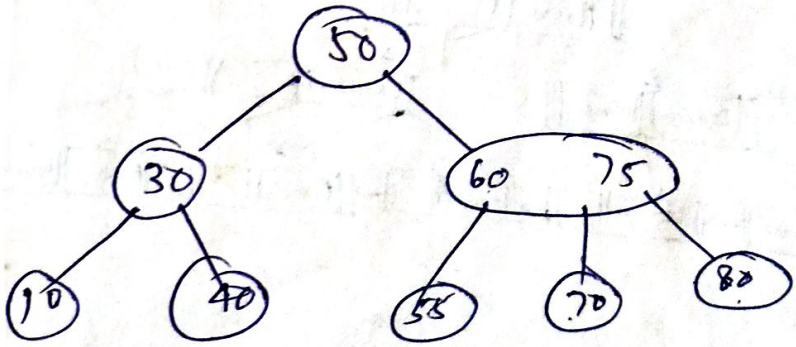
Deletion from a B-Tree of order 3



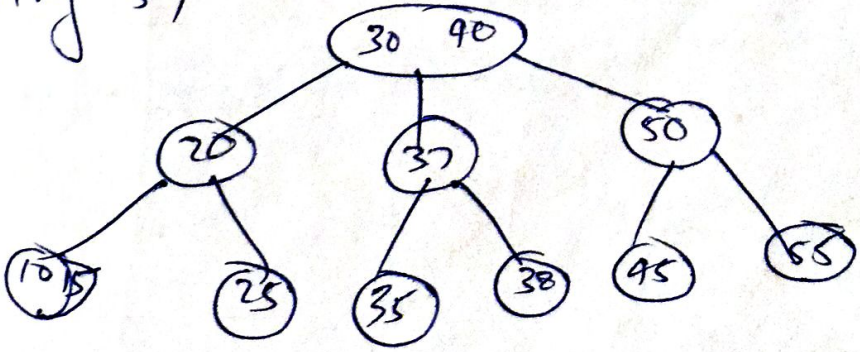
After deleting 58 —



After deleting 65 —



Inserting 37 in above Tree



B^+-Tree Indexing.

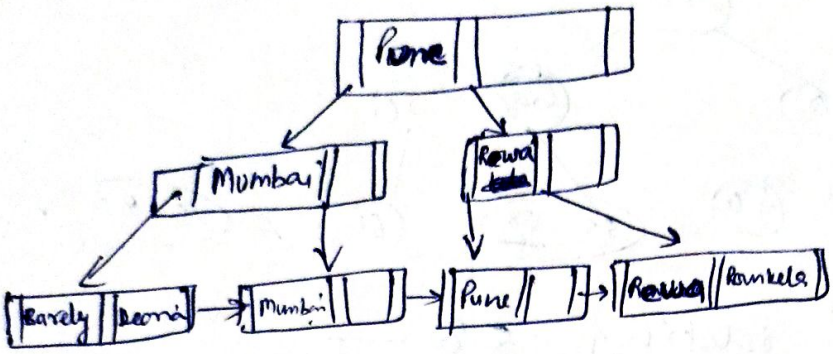
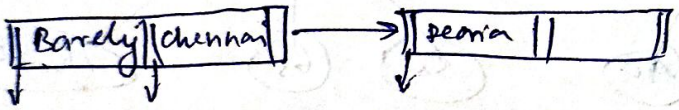


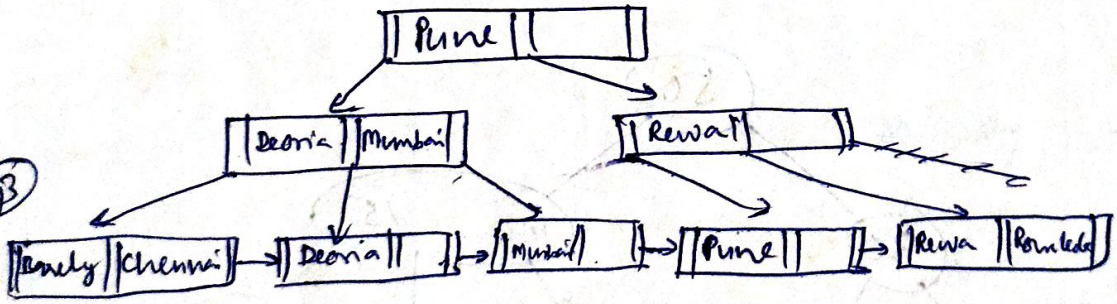
fig 1 B^+ Tree for given cities

Insertion of 'chennai' in above B^+-Tree

(A)



splitting of the leaf node on insertion of chennai



(B)

fig 2 B^+ Tree after insertion of chennai in fig 1

... ...sation of data is called DS.

Deletion of 'Deoria' from above B+ Tree (fig 2)

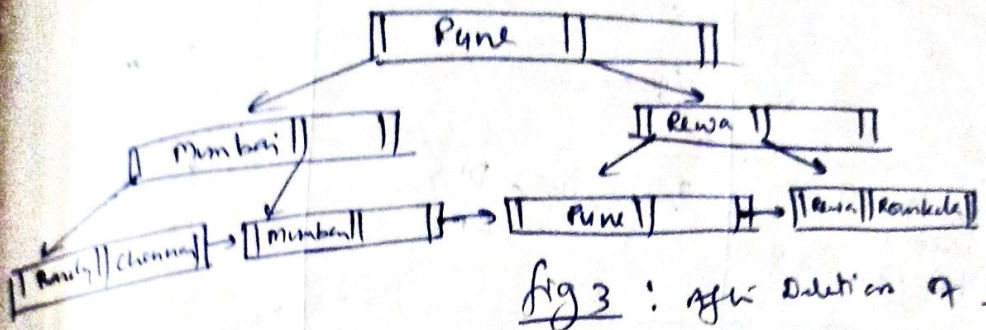


fig 3 : After Deletion of Deoria

Deletion of "Pune" from from above B+ Tree (fig 3)

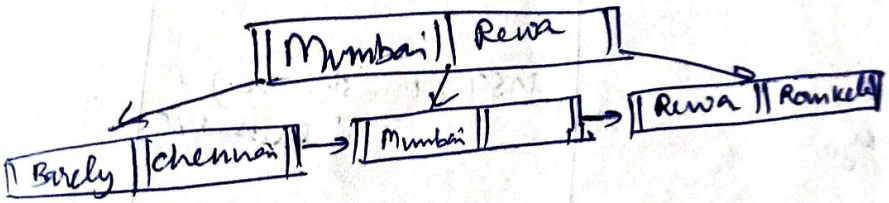


fig 4: After Deletion of Pune

Deletion of 'Pune' from B+ Tree of fig 2

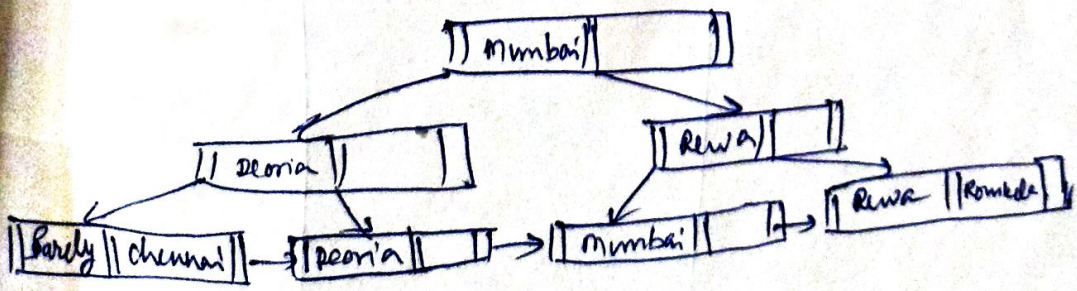


fig 5 : After Del^n of Pune