Unit - V

Multi-way search Tree

→ A multiway Search Tree is a multiway Tree in which every node stores atmost "(m-1)" no of values and "m" no of children, where m is degree of Tree

Ex:- B-tree, B+tree, 2-3 tree etc--

Usage:- Usually, the multiway Search Trees can be used widely in Secondary Storage devices, (Especially harddisk drive) to reduce the disk access time while accessing the data.

Properties:-

① A m-way Search Tree is a multiway tree where as the Binary Search Tree is a two-way tree, hence the m-way Search Tree may (or) may not a binary Search Tree.

② The degree (or) order of Tree (m) defines the no of values & no of subtrees per node.

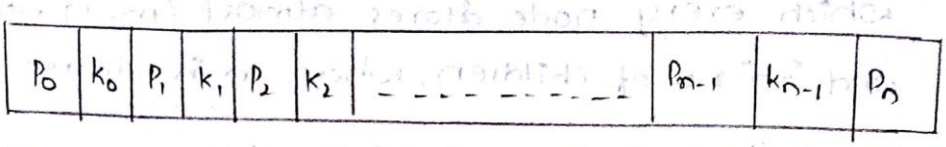
③ It is not compulsory that every node has exactly (m-1) no of values of 'm' subtrees per node.

④ The no of values may be from 1 to (m-1) & the no of subtrees may be from 0 to (i+1) per node, where 'i' is the no of values in that particular node.

⑤ The values in the left subtree are smaller than to their parent & the values in the right subtree are greater than to their parent

⑥ All the values in everynode should be in the

Node Structure

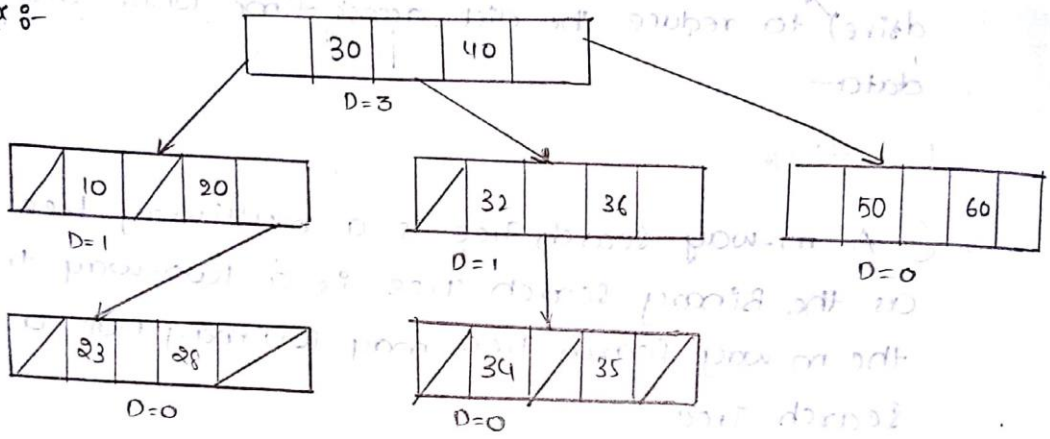


Where,

$P_0, P_1, P_2 \ldots P_n$ are the pointer fields

$k_0, k_1, k_2 \ldots k_n$ are the data (or) key fields

Ex :-



B-Tree

A BTree is one of the multiway Search Tree in which every node stores atmost m-1 number of values and atmost m number of children where m is the degree or order of tree

→ The B-tree was developed by 2 scientists Rudolf Bayer and M.M.C creight in the year 1972.

Properties

→ All the nodes except root node store atleast $(m/2)-1$ no of values & atmost $(m-1)$ no of values. The root may contain minimum one value.

→ All the nodes Except root & leaf node must have atleast m/2 . no of children.

→ Rootnode must have atleast two children if it is not leaf node

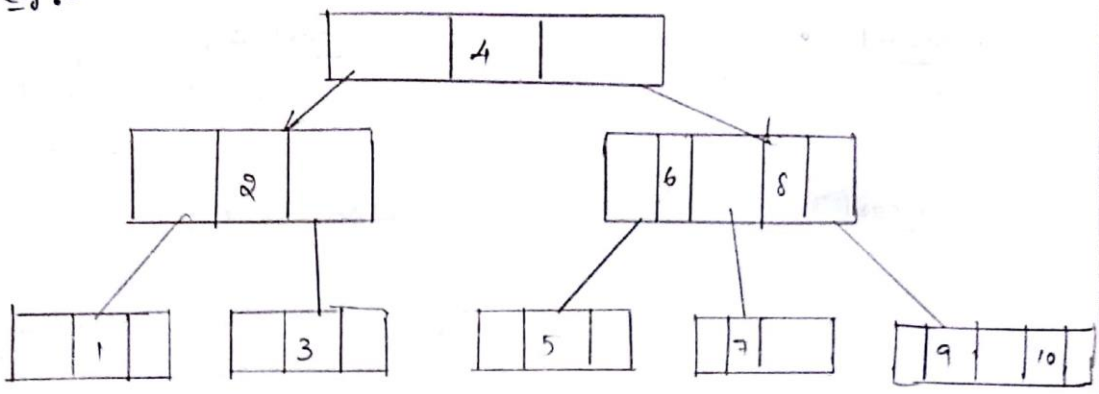
→ If a non-leaf node has 'm' no of children then it must have $(m-1)$ no of values

→ All the leaf nodes should be in same level.

→ All the nodes (Btree) Except rootnode must have atleast (m-1) values and atmost (m-1) values. The root node may contain min one value - Both Insertion & deletion

→ The values in everynode should be in ascending order
The order of BTree always should be odd number

Eg :-



Operations of B-Tree :-

Usally three operations performed on B-Tree they are

1) Insertion
2) Deletion
3) search

Insertion operation in BTree

In Btree, every new value is Inserted at leaf node

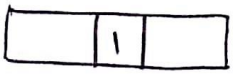
procedure steps :-

1) Check whether the tree is empty (or) not
2) If the tree is Empty then create a newnode & Insert new value into it and that becomes rootnode
3) If the tree is not empty then find the specific leaf node
   a) If the leafnode has Empty datafield then insert new value by following Binary Search Tree logic & ascending order.
   b) else, split it the tree by Sending middle value as parent Repeat the process untill the Sent value is fixed into a suitable node.
   c) If the Spliting is occured at root node then the Sending value becomes newroot and the height of tree is also enhanced.

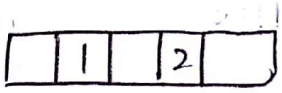
Example :-

construct B-tree of order 1 to 10

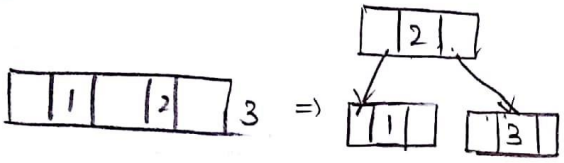
Insert 1 :-



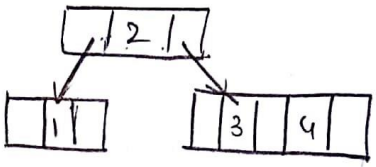
Insert 2 :-



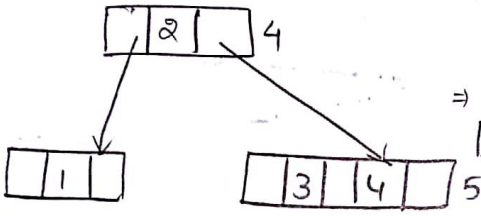
Insert 3 :-



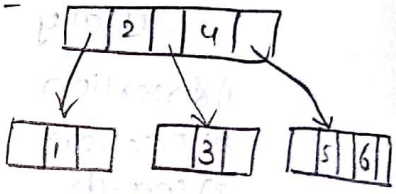
Insert 4 :-



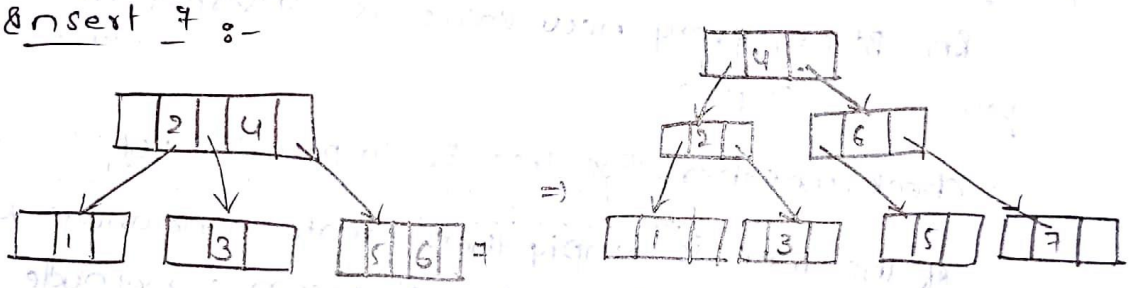
Insert 5 :-



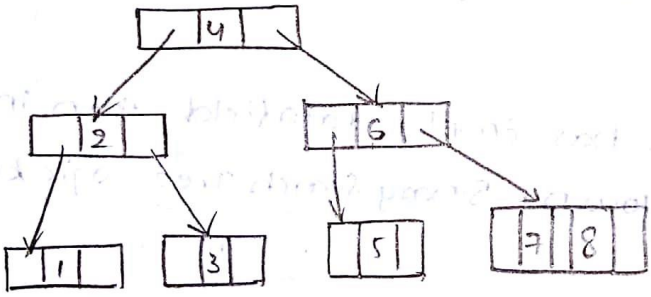
Insert 6 :-



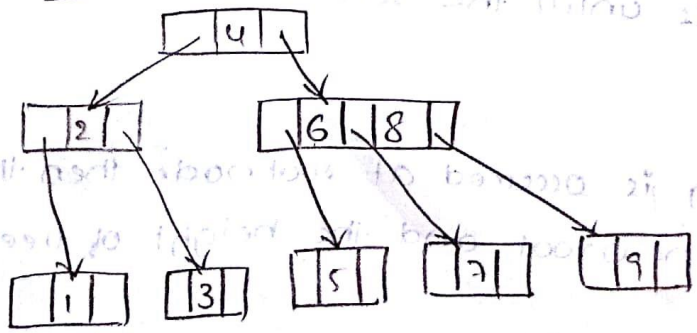
Insert 7 :-



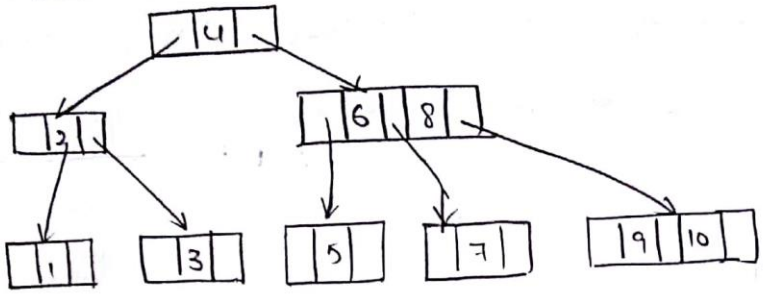
Insert 8 :-



Insert 9 :-



Insert 10:-



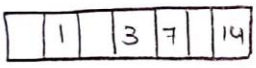
Construct BTree of order m=5 of following elements

3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25 & 19

max = 4, min = 2

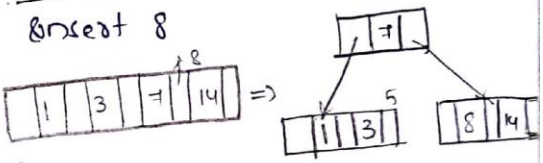
Step ①:-

Insert 3, 14, 7, 1



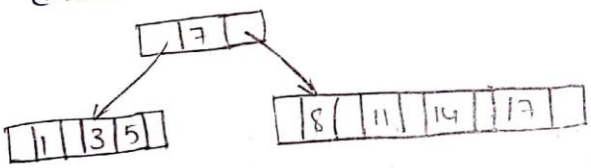
Step ②:-

Insert 8

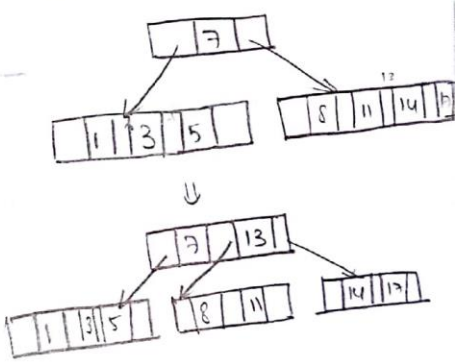


Step ③:-

Insert 5, 11, 17

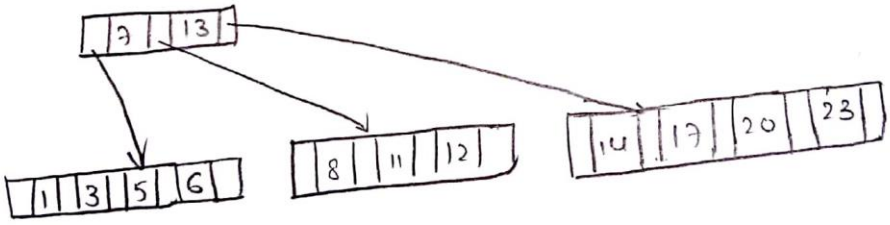


Step ④:- Insert 13

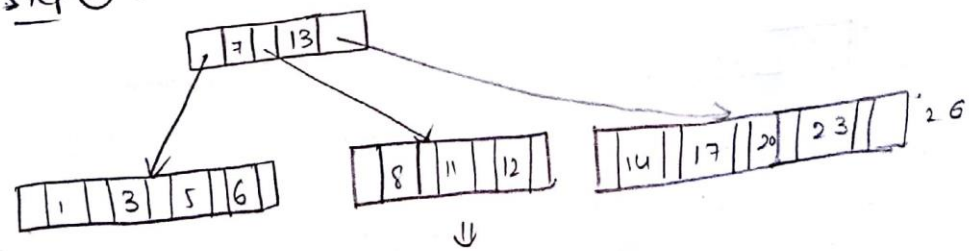


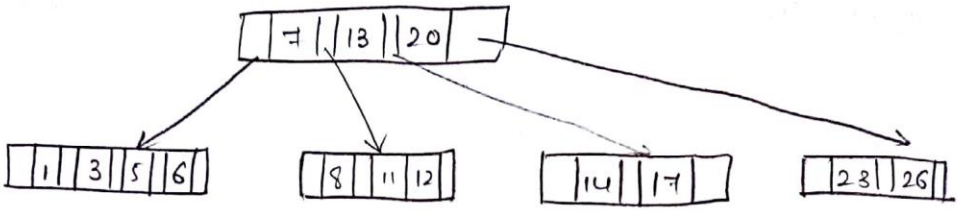
Step ⑤ :-

Insert 6, 23, 12, 20

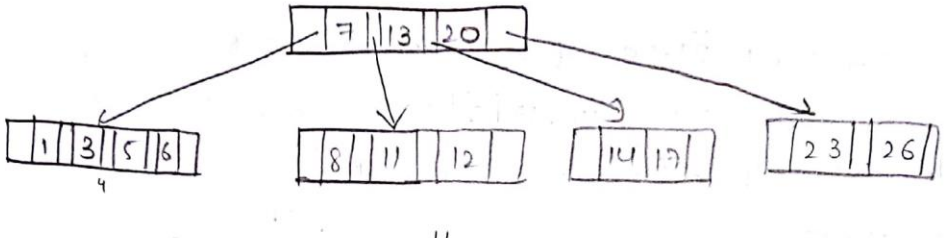


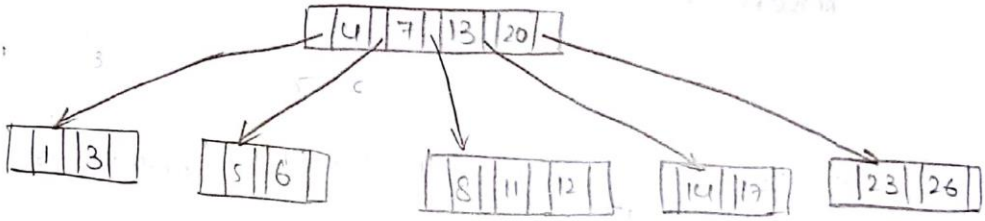
Step ⑥ :- Insert 26



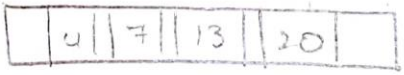


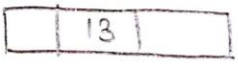
Step(7) :- Insert 4

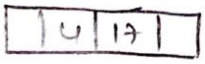


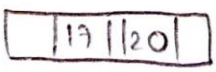


Step(8) :-









Eg:- Construct a B-tree of order 3 by following elements

10, 20, 30, 40, 50, 60, 70, 80, 90, 100

$m = 3$

$min = (m-1)$

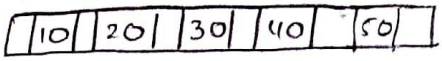
$= 3-1 = 2$

$max = (2m-1)$

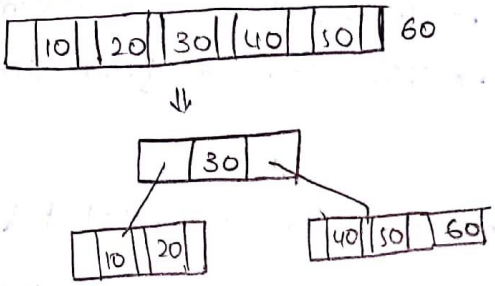
$= (6-1)$

$= 5$

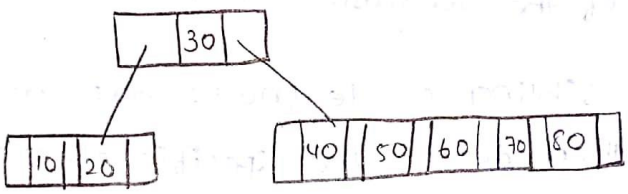
Insert :- 10, 20, 30, 40, 50



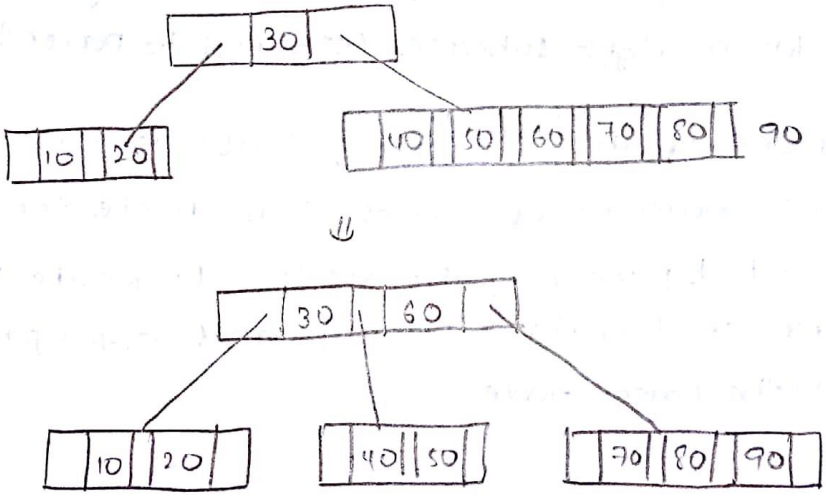
Insert :- 60



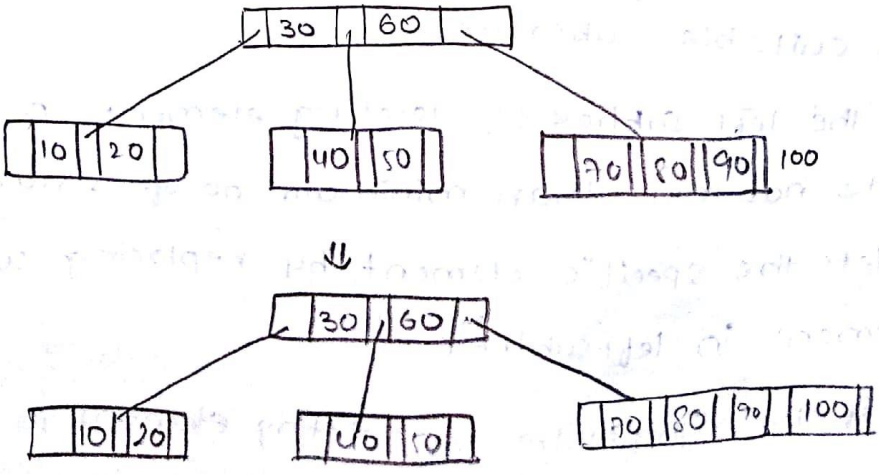
Insert :- 70, 80



Insert :- 90



Insert :- 100



Deletion in the B-Tree

To delete element from B-tree we must follow certain cases

Case(i):-

If an element is to be deleted from the leafnode and the leafnode has more than the minimum no of elements then delete the specific element from the leafnode.

Case(ii):-

If an element is to be deleted from the leafnode and the leafnode has only minimum no of values then follow one of the suitable subcases.

a) If either sibling of leafnode has more than minimum no of values then delete the specific element by moving appropriate value from parent node to specific leafnode and move largest value from left sibling (or) smallest value from right subcases (sibling) to parent node

b) If none of the siblings of leafnode has more than the minimum no of values then delete the specific element by merging the specific leafnode with either left (or) right sibling and appropriate value from the parent node.

Case(iii):

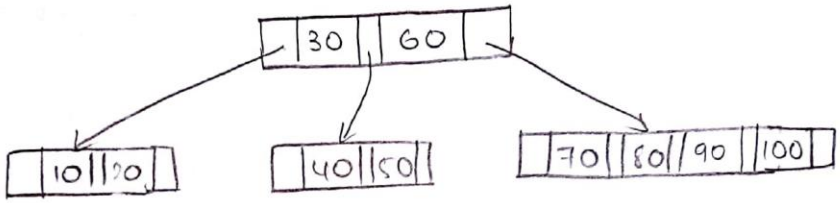
If an element is to be deleted from the internal node (including root node) then follow one of the three suitable subcases

a) If the left subtree of deleting element in the internal node has more than minimum no of values then delete the specific element by replacing with largest element in leftsubtree

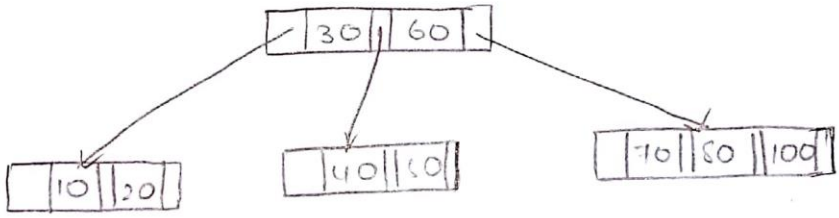
b) If the right subtree of deleting element in internal

node has more than the minimum no of values then delete the specific element by replacing with smallest element from the right subtree.

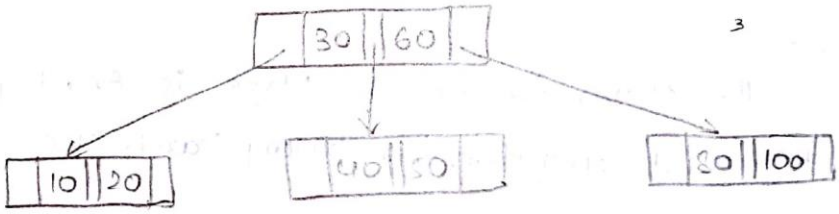
c) If both left & right subtrees of deleting element in the internal node no of values then delete the specific element by merging both left & right subtrees of deleting element.



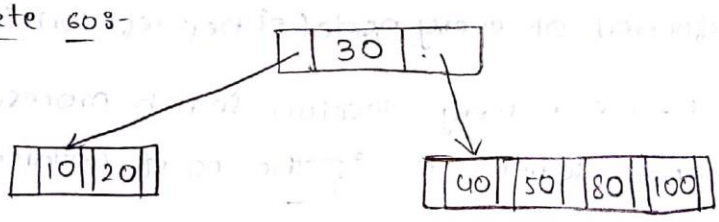
Delete 90 :-



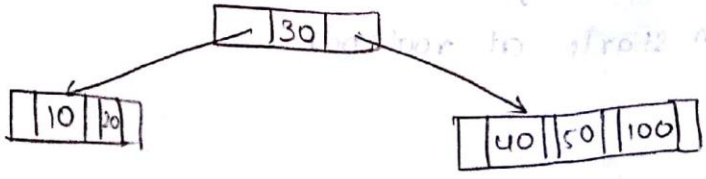
Delete 70 :-



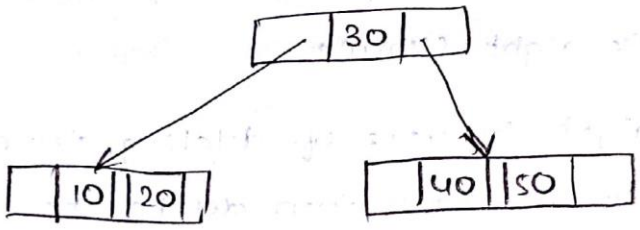
Delete 60 :-



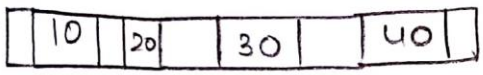
Delete 80 :-



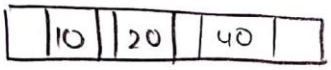
delete 100:-



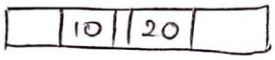
delete 50:-



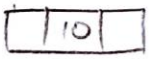
delete 30 :-



delete 40:-



delete 20:-



Search Operation

→ The search operation in Btree is exactly similar to the search operation in Binary search Tree

→ In the Binary search Tree, two way decision search process will be performed at every node (either left (or) right subtree)

→ In the B-Tree, n-way decision search process will Perform at every node where 'n' is the no of children of specific node.

→ In both Binary search Tree and BTree, the search operation starts at rootnode.

procedure steps

→ check whether the tree is empty (or) not

→ If the tree is Empty then the search element will not be Found. Otherwise,

→ If the tree is not empty, the search process starts with Rootnode. Initially, the search element is compared with first element in the rootnode. If the search element is matched with first element then the element is found at rootnode

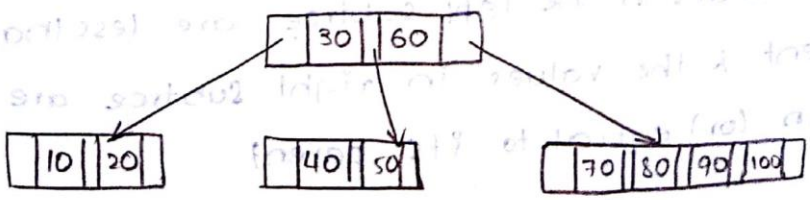
→ If the search element is lessthan first element in the Rootnode then the search operation will be performed recursively on the leftsubtree of first element in the Root node

→ If the search element is greater than the first element in the rootnode then the search element is compared with next element in the rootnode. If both are matched then the search element is found at rootnode.

→ If the rearch element is less than the next element in the rootnode then search operation will be performed recursively on the leftsubtree of the next element in the rootnode.

→ If the search element is greater than next element in the rootnode then the search element is compared with another element in the rootnode if any. otherwise the search operation will be performed recursively on the right subtree of next element in rootnode

→ If the Search element is not matched with anyelement in the B-tree then that indicates the search element is not available in the entire B-tree.



B+ tree

→ The B+ tree is another multiway search tree in which everynode also contains m no of children, where 'm' is degree (or) order of tree

→ The B+ tree is an advanced tree of B-tree

→ The B+ tree was developed by one of the foriegn scientist Douglas Comer in the year 1973

→ In General, the B+ trees can be used mostly in the implementation of databases. and also used in secondary storage devices. (Especially in Hard disk drive)

→ The operations like insertion, deletion, and retrival can be performed effeciently on the B+ tree. than B+tree

→ usually, the B+ tree occupies more space than B-tree

→ the B+ tree follows the combination of ISAM (Index sequential Access method) & B+tree properties

Properties of B+ tree

→ All the properties of B-tree are inherited into B+ tree

→ The elements stored in the leaf nodes are treated as actual elements and the elements stored in the internal nodes are treated as index values which are used to identify the actual elements in leaf nodes

→ B+ tree may store redundant data

→ All the leafnodes are connected to one another in the form of linked list for faster data access

→ The values in the left subtree are less than to its parent & the values in right subtree are greater than (or) equal to its parent

→ the operations of B+ tree are 1) Insertion

2) deletion

3) search

Insertion

m = 3

min = m - 1

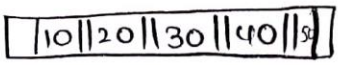
= 3 - 1 = 2

max = 2m - 1

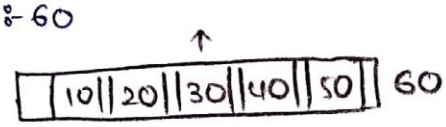
= 2(3) - 1

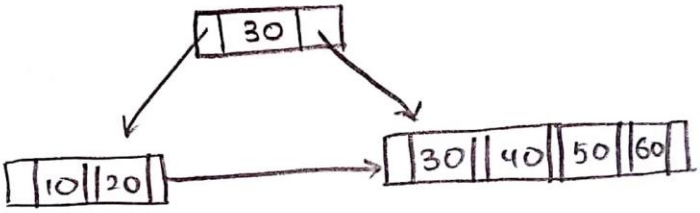
= 5

Insert :- 10, 20, 30, 40, 50

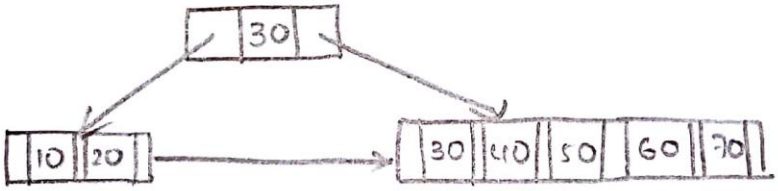


Insert :- 60

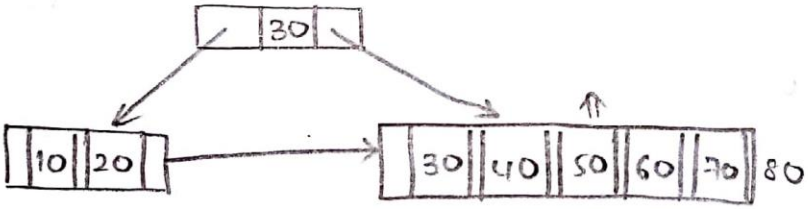


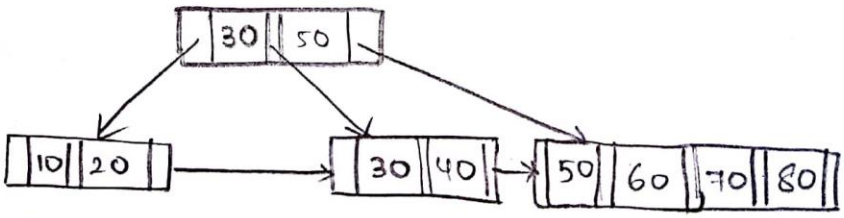


Insert :- 70

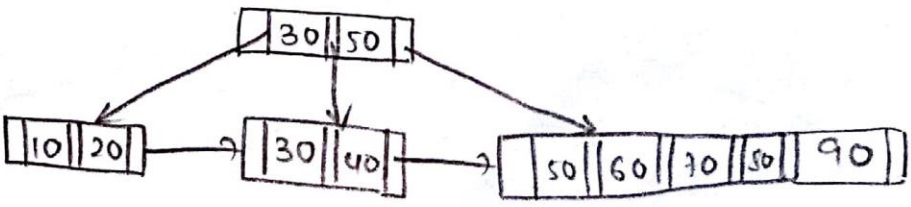


Insert :- 80

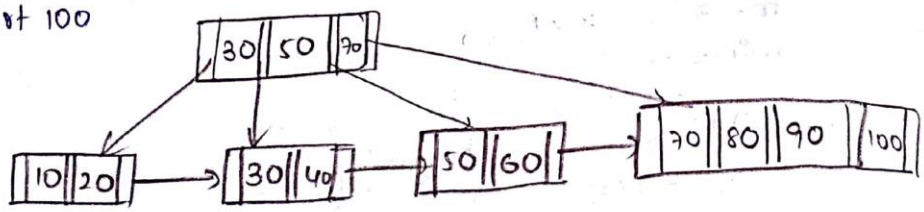




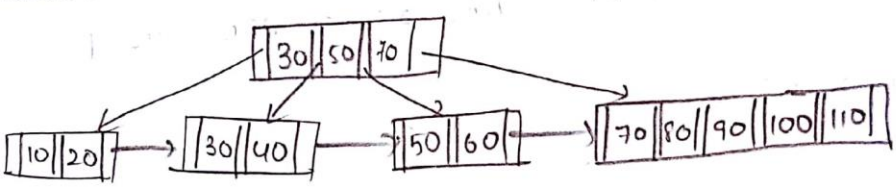
Insert :- 90



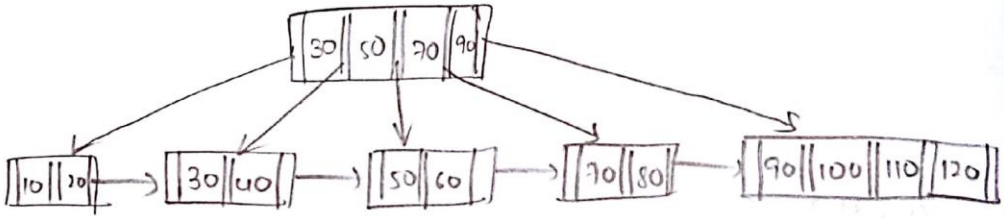
Insert 100



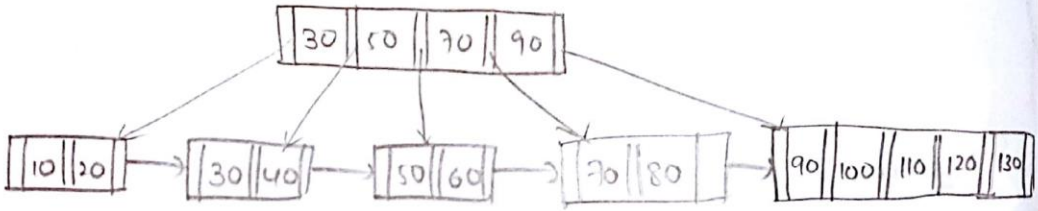
Insert 110



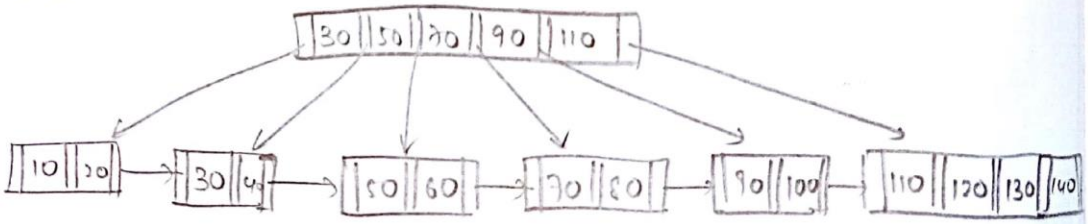
Insert 120



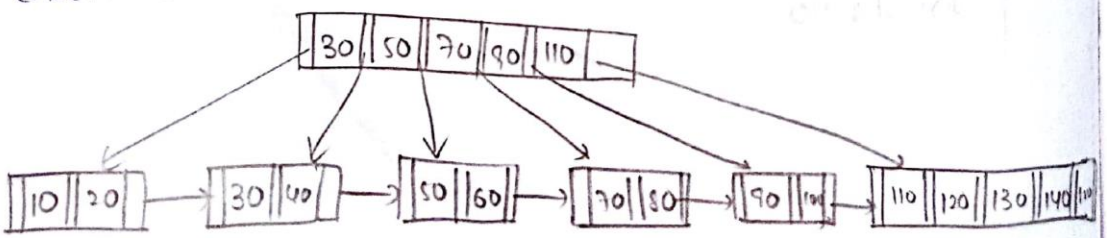
Insert 130



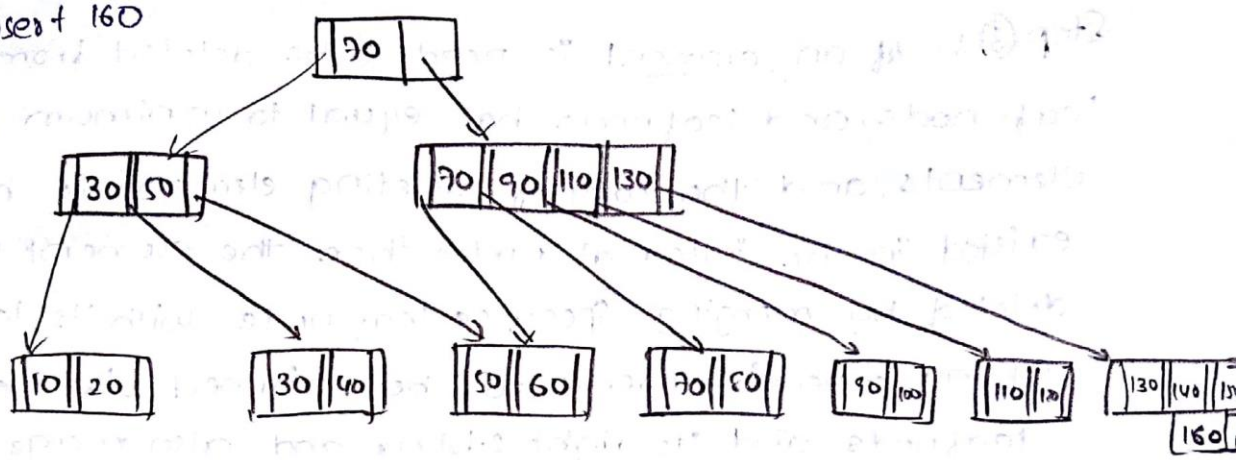
Insert 140



Insert 150



Insert 160



Note:- During insertion operation in Bt tree, when a node is to be splited into two parts the middle value is stored into second splitted part and also stored into its parent node

Deletion operation in Bt tree:-

In Bt tree the deletion is performed at leaf node and copy of deleting element in internal node will be removed by default

To performed, the deletion operation in Bt tree we follow certain cases

Step①:- If an element to be deleted from leafnode and the leafnode has more than minimum number of elements and the copy of deleting element is not existed in internal nodes then the element can be directly from specific leafnode

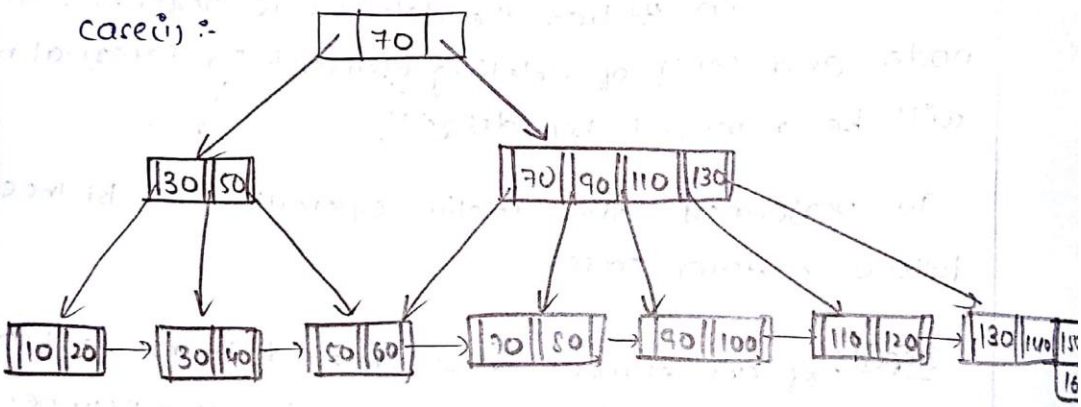
Step②:- If an element is need to be deleted from the leafnode and the leafnode has more than minimum number of elements (or) equal to minimum no of elements and copy of deleting element is existed in internal nodes then the element can be deleted from specific leafnode by merging specific leafnode with its leaf sibling and the internal nodes should be adjusted (merge will perform b/w internal nodes if require)

Step ③:- If an element is need to be deleted from the leaf node and leafnode has equal to minimum no of elements and the copy of deleting element is not existed in the internal nodes then the element can be deleted by merging specifice leaf node with its left sibling otherwise merge will be performed b/w specific leafnode and its right sibling and also merge will be performed b/w the internal nodes, (if required) to adjust the B+ tree

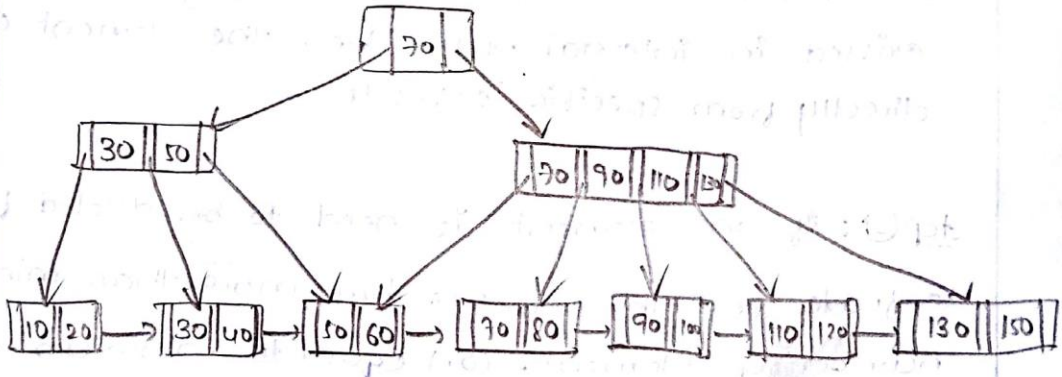
Eg:-

Step ①:-

Case(i):-

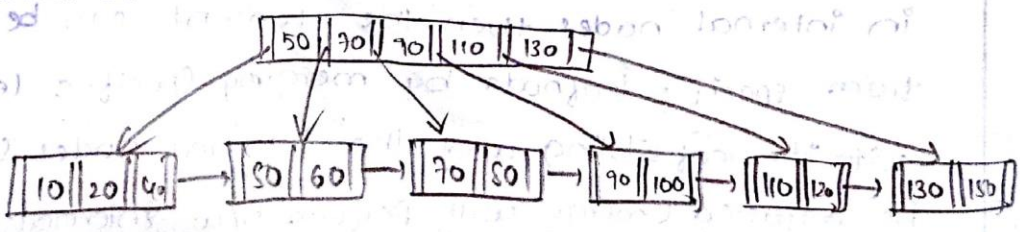


Delete 140 & 160

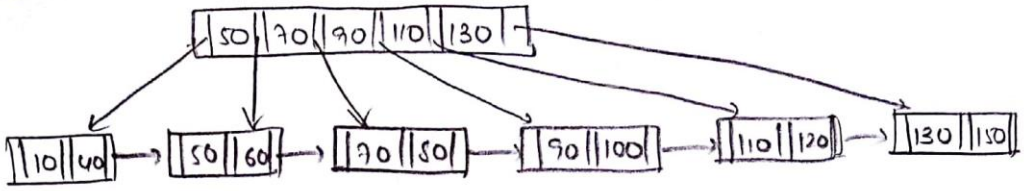


Step ②:-

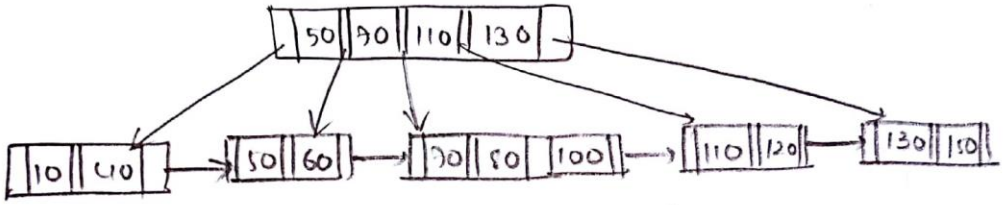
Delete 30



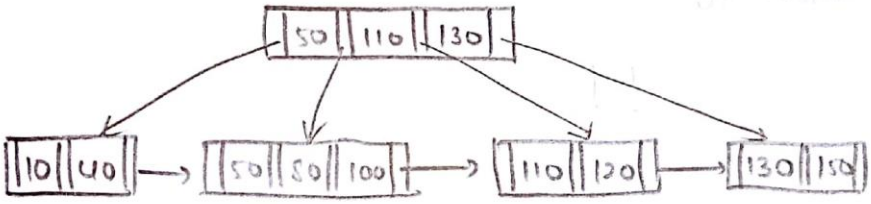
Step③ :- delete 20



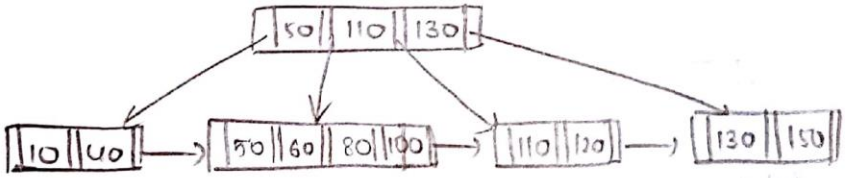
delete 90



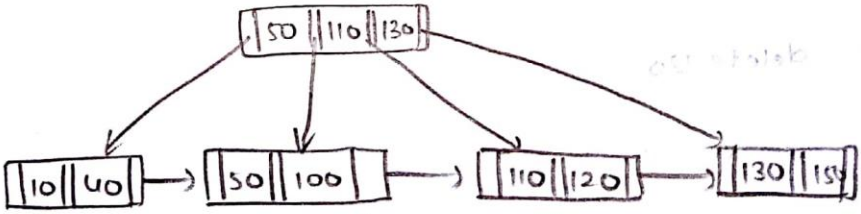
4. delete 60



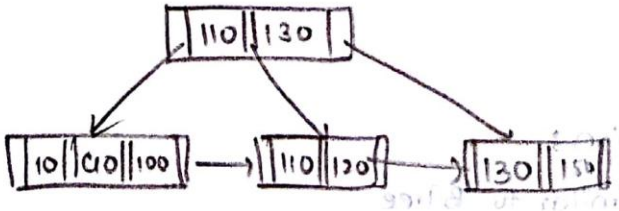
70 delete 70



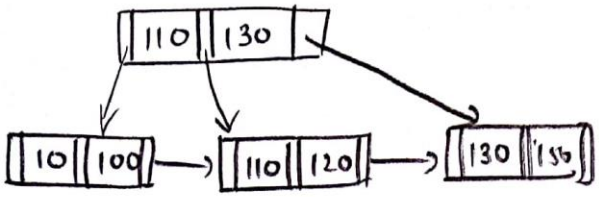
80 delete 80



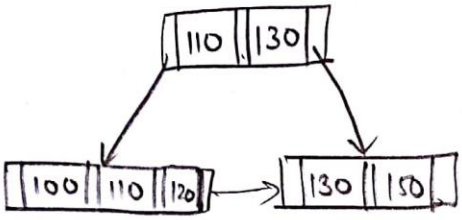
50 delete 50



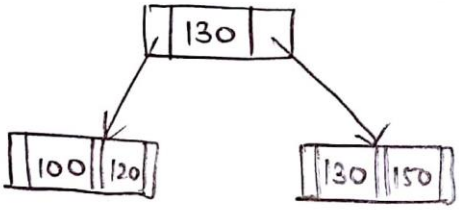
delete 40



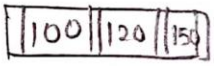
10 delete 10



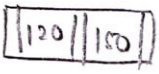
110 delete 110



130 delete 130

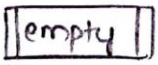


100 delete 100



120 delete 120

150 delete 150



Search operation:-

Similar to BTree