B-TREE.

The tree structure is best suitable for maintaining the indices of elements in it.

The main purpose of this indexing is to accelerate the search procedure.

Binary Search Tree (BST) uses the concept of TREE INDEXING, where each node contains a key value, pointers to the left subtree and right subtree.

Binary Search Tree → is in fact, a 2-way Search Tree & this concept of Tree Indexing can be generalised for an m-way ($m \geq 2$) search tree ($m=2$ is a special case for BST) with the following

definition :→

1. An m-way search tree T is a tree in which all the nodes are of degree $\leq m$.
2. Each node in the tree contains the following attributes.

P_0	K_1	P_1	K_2	P_2	......	K_n	P_n

where

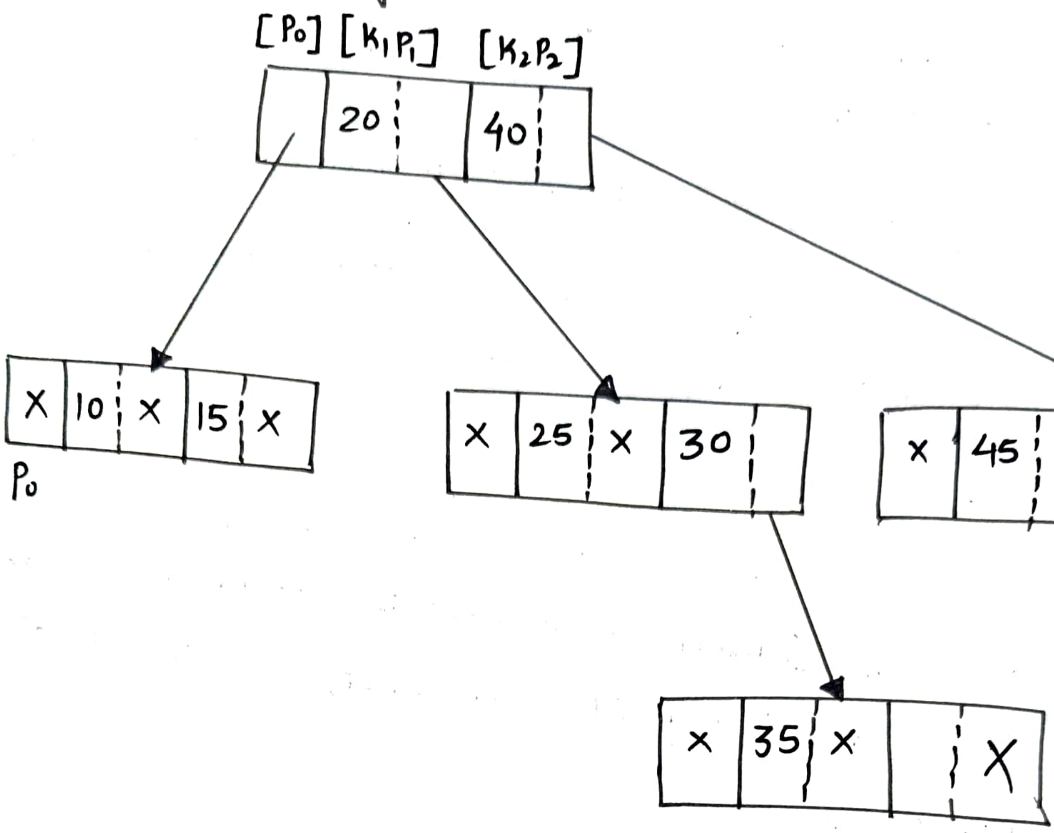
$1 \leq n < m$

K_i ($1 \leq i \leq n$) are the key value in the node.

P_i ($0 \leq i \leq n$) are the pointers to the subtrees of T.

3. $K_i < K_{i+1}$, $1 \leq i < n$
4. All the key values in the subtree pointed by P_i are less than the key value K_{i+1}, $0 \leq i < n$.
5. All the key values in the subtree pointed by P_n is greater than K_n.
6. All the subtrees pointed by P_i ($0 \leq i \leq n$) are also the m-way search tree

A-3-Way Search Tree



B-Tree Indexing

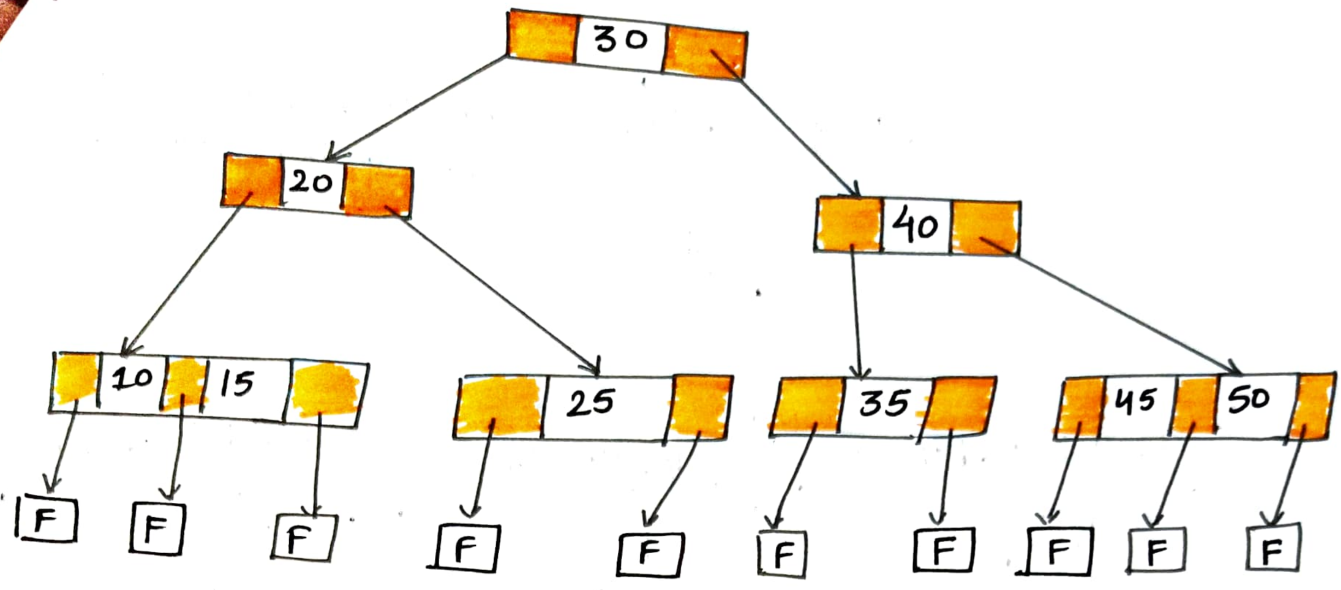
→ B-Tree is an extension of the m-way search tree. In fact, in order to improve the search efficiency, the tree should be balanced. and if the m-way search tree is height Balanced then it is termed B Tree.

Definition :→ A B-Tree T of order m is an m-way search tree, that is, either it is empty or it satisfy the following property →

1. The root node has atleast 2 children.
2. All nodes other than the root node have atleast $\lceil m/2 \rceil$ children
3. All failure nodes are at the same level.

A Failure Node :→ represents a node which can be reached during a search only if the value say, x, being searched for, is not in the Tree. For convenience, these empty sub-trees are replaced by hypothetical nodes called Failure Nodes.

B-Tree of order 3.



Operation on A B-Tree

(1). INSERTION.

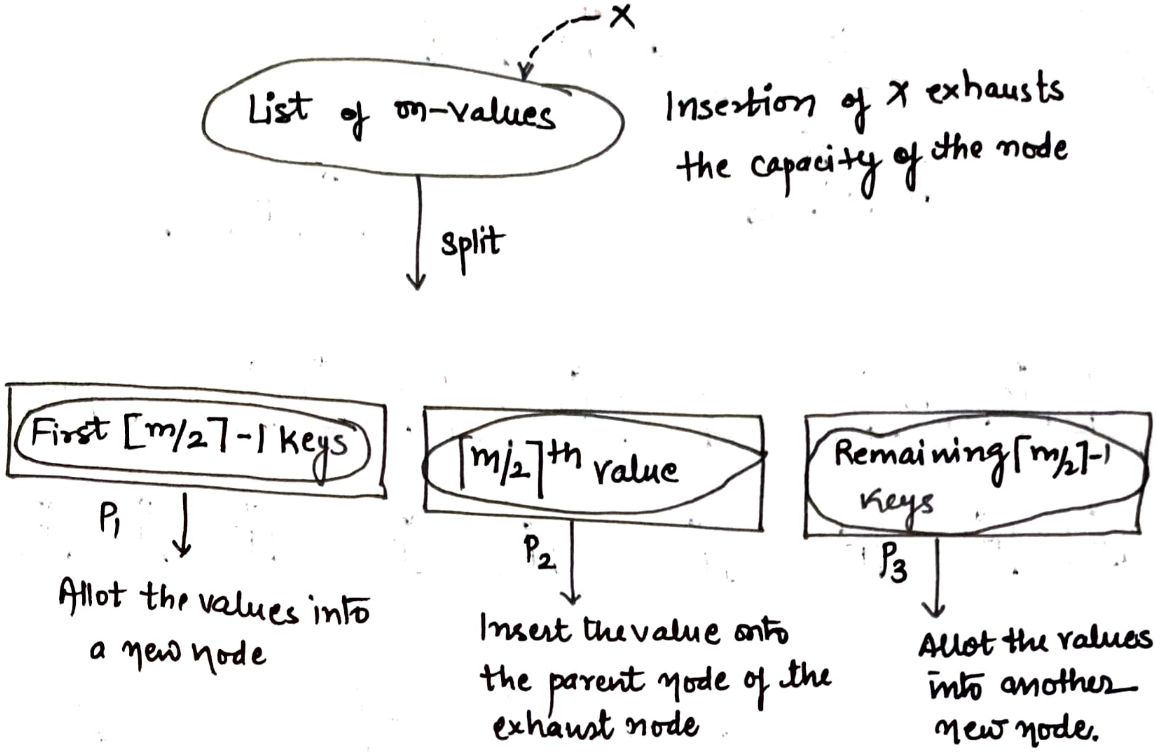
* Note:

When a key value is to be inserted which has already max. number of key value THAT IS $(m-1)$ for a BTree of m-order.

1. Insert the value, say x, into the lists of values in the node is ASCENDING ORDER

2. Split list of values into three part.
   - P_1 = contain the first $\lceil m/2 \rceil - 1$ key values
   - P_3 = contain $\lceil m/2 \rceil + 1 \ldots, m^{th}$ values
   - P_2 = value contain $\lceil m/2 \rceil$th value

3. With this splitting, the $\lceil m/2 \rceil^{th}$ value is to be inserted into parent node

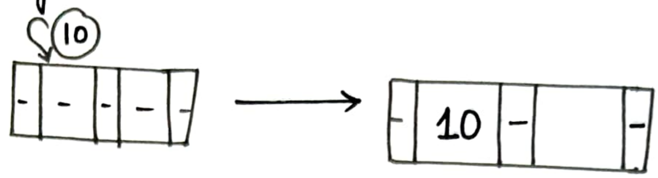

x
List of m-values
Insertion of x exhausts the capacity of the node
split
First ⌈m/2⌉-1 keys
⌈m/2⌉th value
Remaining ⌈m/2⌉-1 keys
P1
P2
P3
Allot the values into a new node
Insert the value onto the parent node of the exhaust node
Allot the values into another new node.

INSERTING INTO A B-TREE

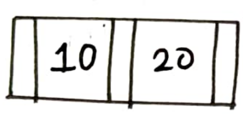
10, 20, 30, 40, 50, 60, 70, 80, 90

Assume that the order of the B-Tree is 3?

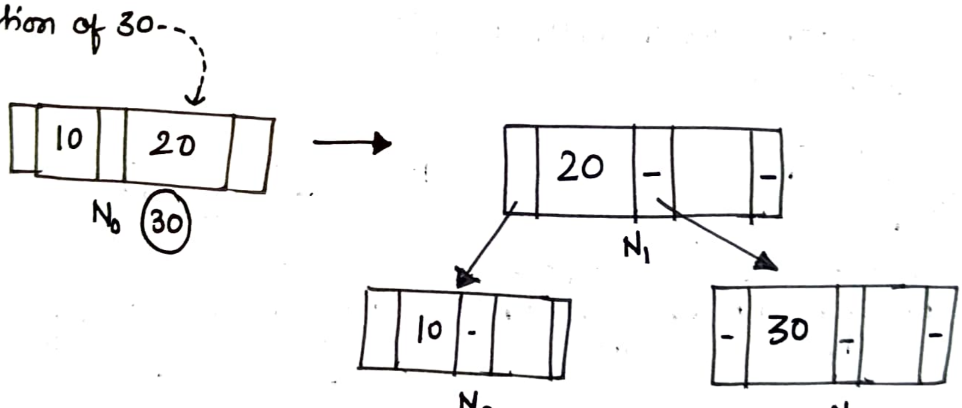
(a) Insertion of 10.



(b) Insertion of 20.

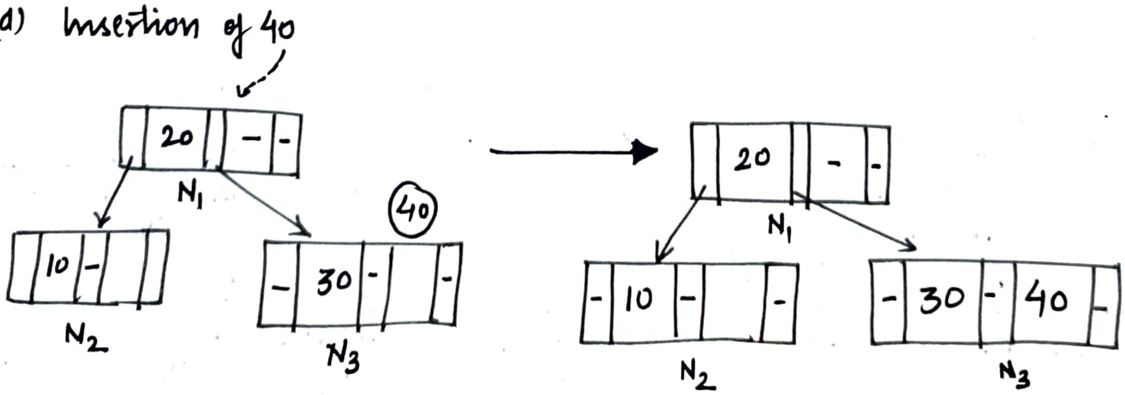


(c) Insertion of 30



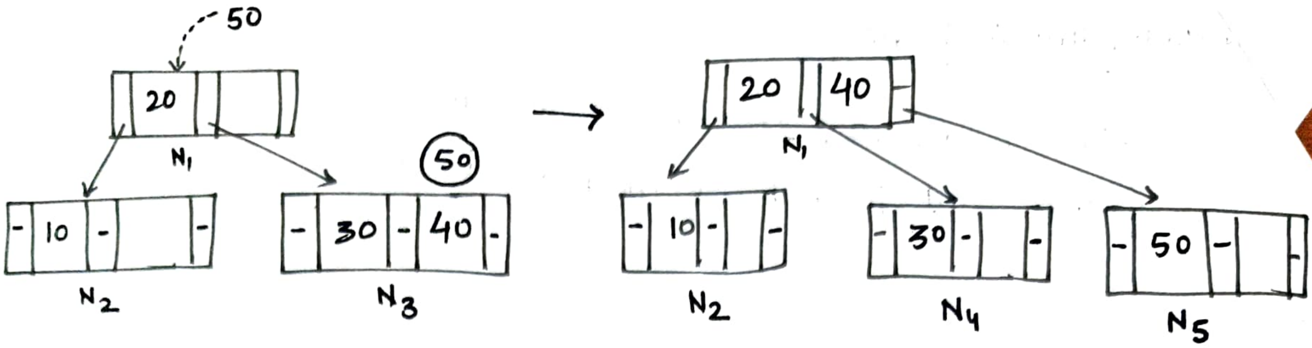
(c) Insertion of 30, Bursting of Nodes.

(d) Insertion of 40

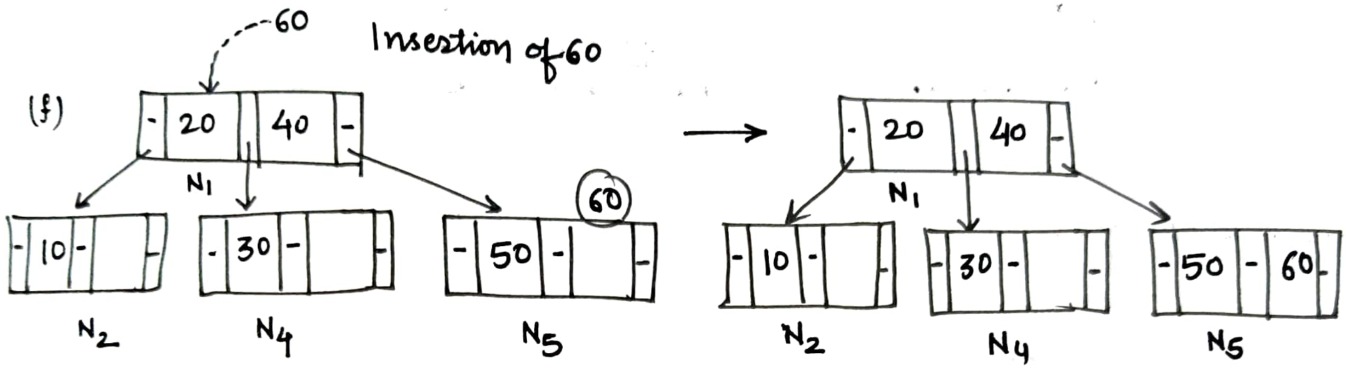


(d) Insertion of 40, it goes to the N_3 Node

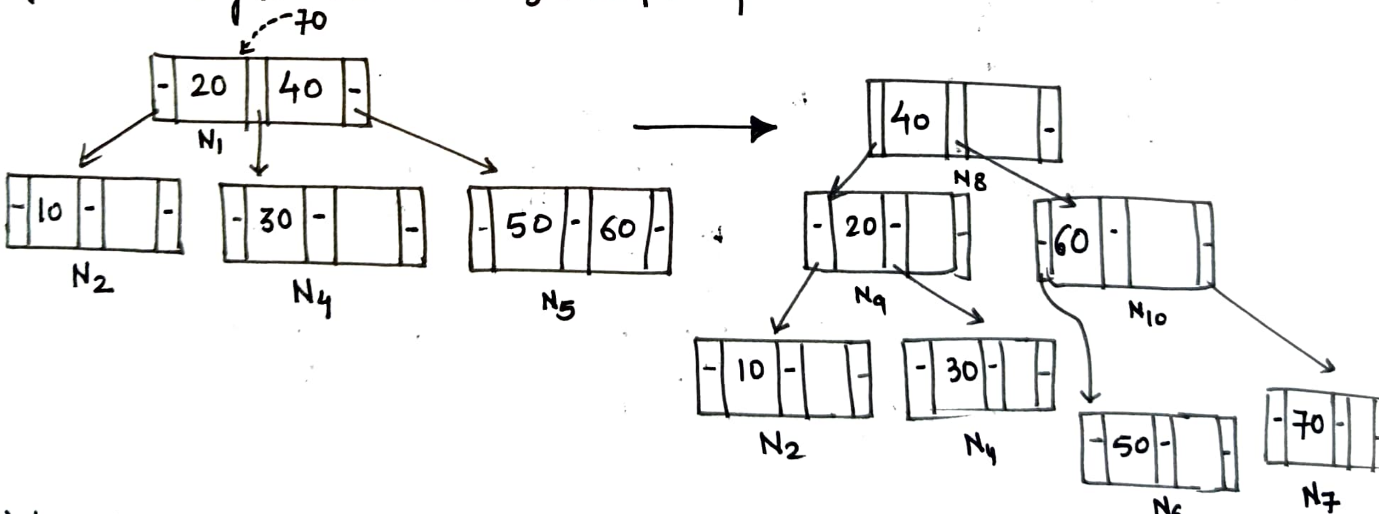
(e) Insertion of 50, splits N_3 into N_4 and N_5



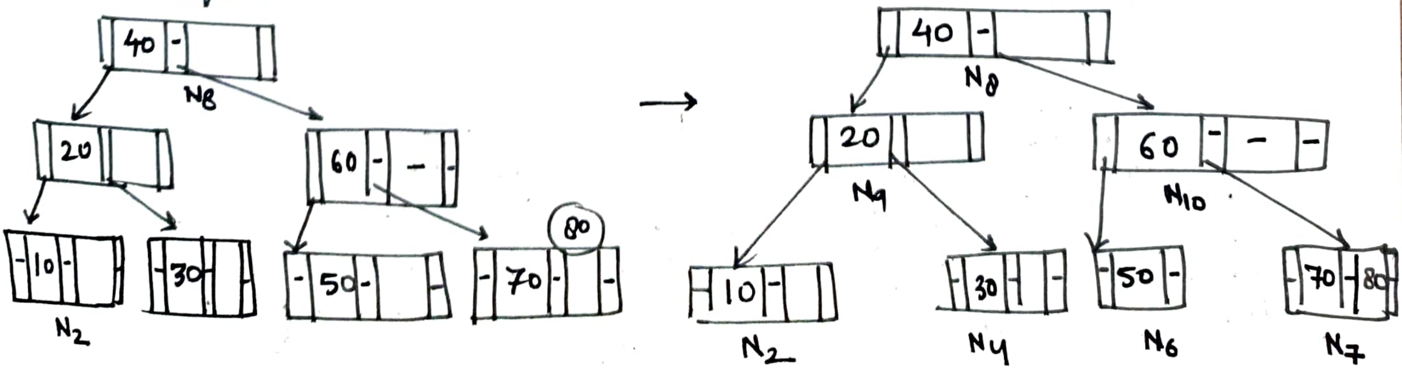
(f) Insertion of 60



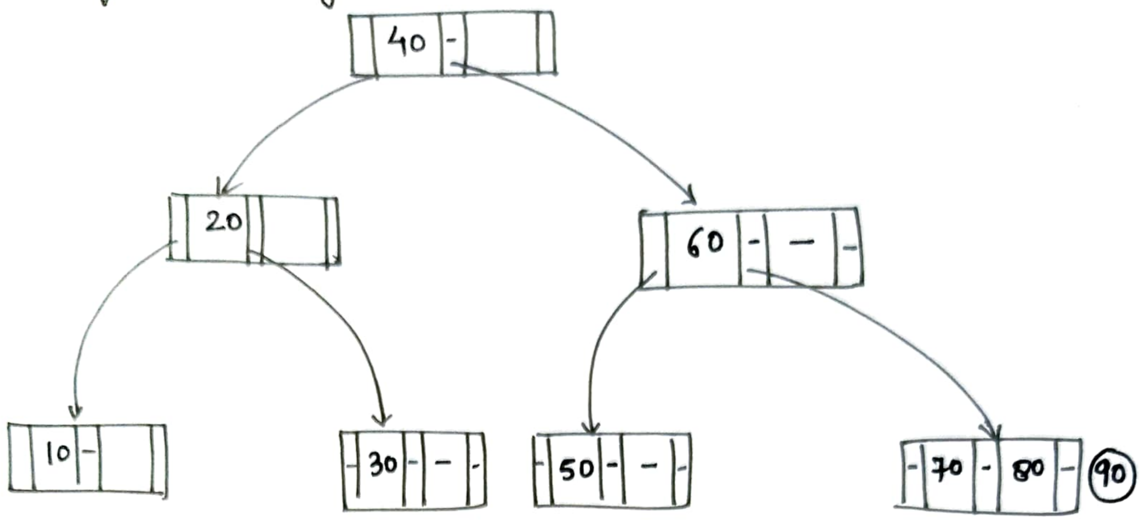
(g) Insertion of 70 : two nodes N_5 and N_1 are split

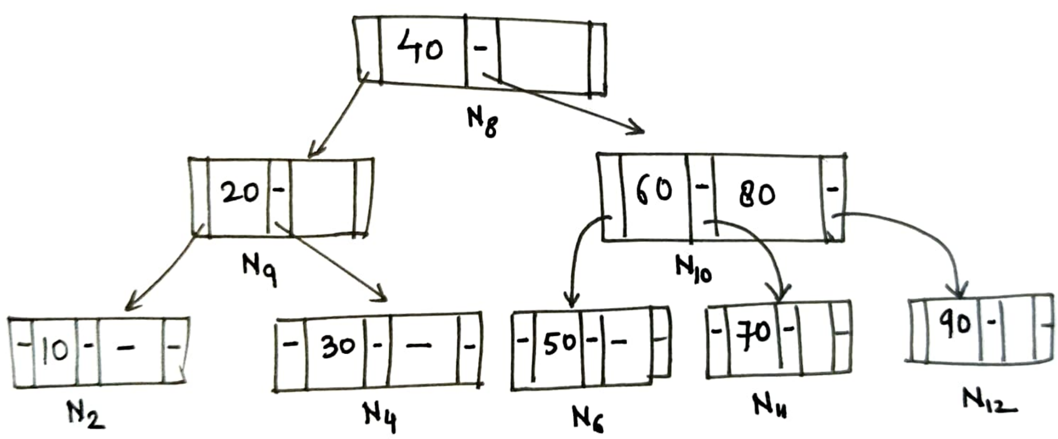


(h) Insertion of 80



insertion of 90 ; splitting of N_7 into N_{11} and N_{12} & then insertion of 80 into N_{10}

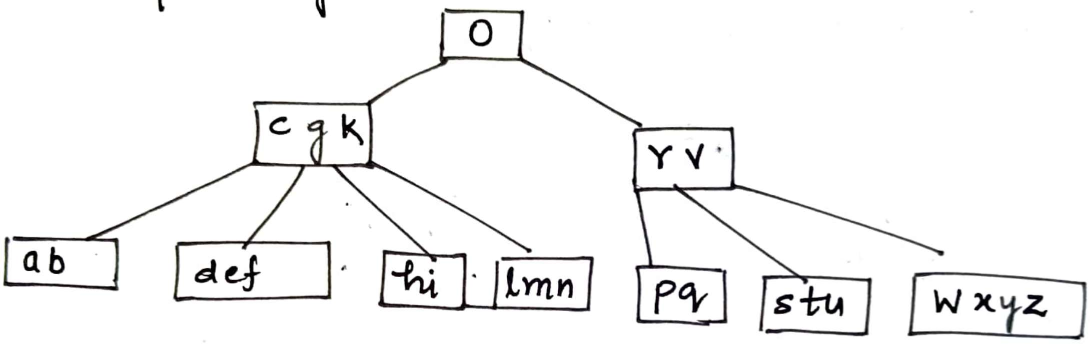




Deleting B-Tree

Whatever be the situation, the deletion of a node must ensure the properties of a B-Tree. The main two properties that need to be taken care of during deletion are stated below:-

- The root node must have atleast one key value.
- All other nodes (except the root node) must have atleast $\lceil m/2 \rceil - 1$ key values.
- $m \rightarrow$ order of a B-Tree.

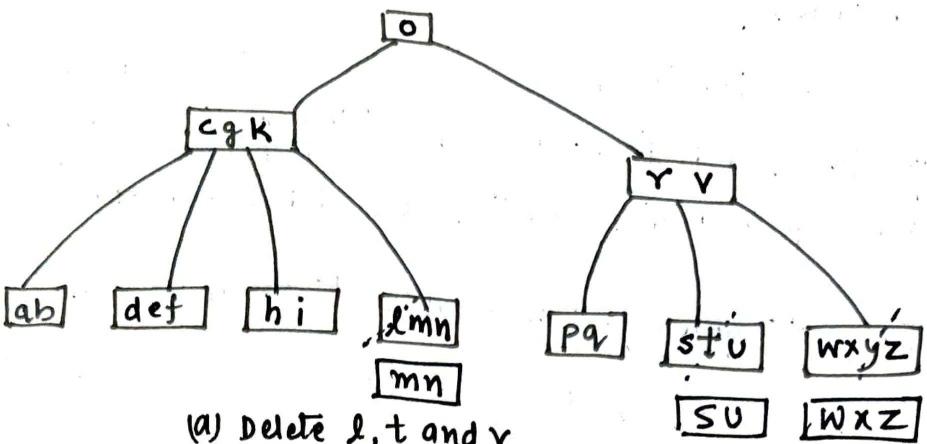


CASE 1: Deletion of a key value from a leaf node.

In a B-Tree, a leaf node is the node which does not have any children. As we have already pointed out that all the nodes other than the root node have atleast $\lceil m/2 \rceil - 1$ key values. After deletion, depending on whether a node (from which a key value has to be deleted) contain minimum number of nodes or not, there are two sub-cases :→

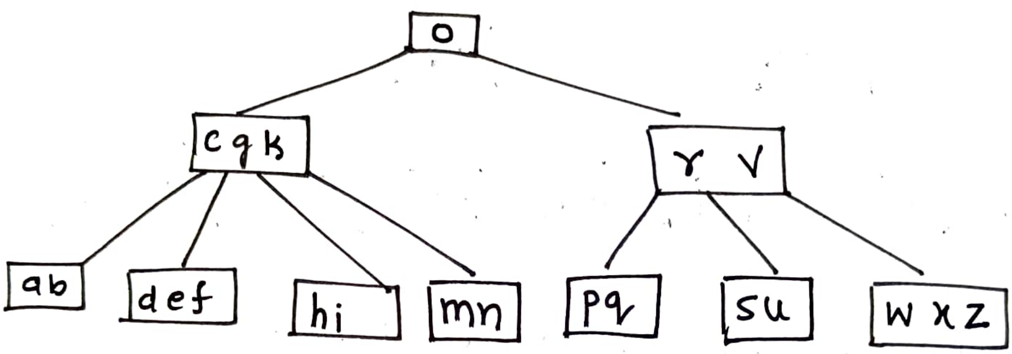
Case 1.1 Removal of a key value leads to the number of keys $\geq \lceil m/2 \rceil - 1$.

In this case removal of a key value from the leaf node which does not disturb the requirement of minimum number of key values in that node.



(a) Delete l, t and y.

(b) After deletion of l, t and y



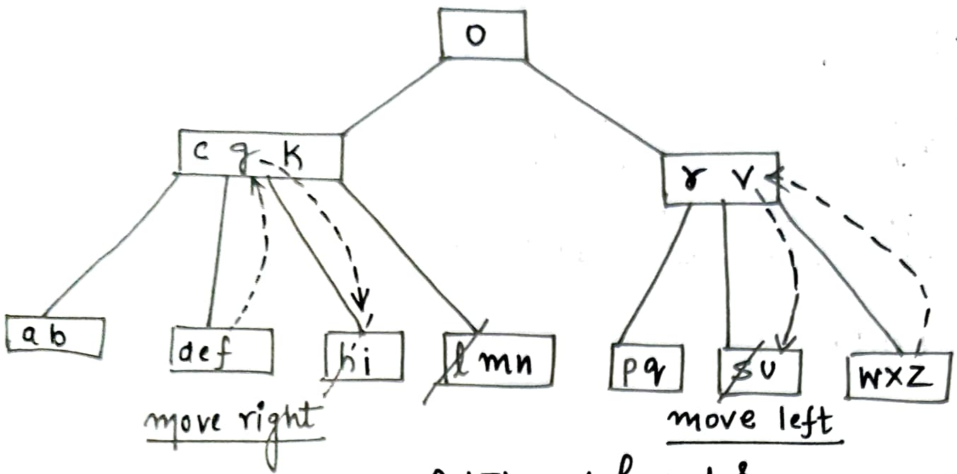
Case 1.2 : Removal of key value leads to the number of keys < $\lceil m/2 \rceil - 1$.

In this case, when a key value is removed, the no. of key value will be < $\lceil m/2 \rceil - 1$ (and thus violating the requirement of minimum number of key values in that node).

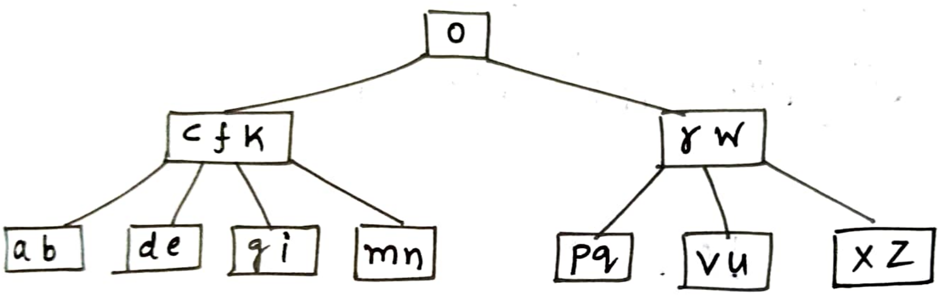
If such a case occur, then we have to move the key value from the sibling (left or right) of the node. Three situation may be possible in this case :-

1. The nearest right sibling contains more than $\lceil m/2 \rceil - 1$ key values.
2. The nearest left sibling contains more than $\lceil m/2 \rceil - 1$ key values.
3. Neither the nearest left sibling nor the right sibling contain more than $\lceil m/2 \rceil - 1$ key values.

onsider the deletion of s and h from the B-Tree



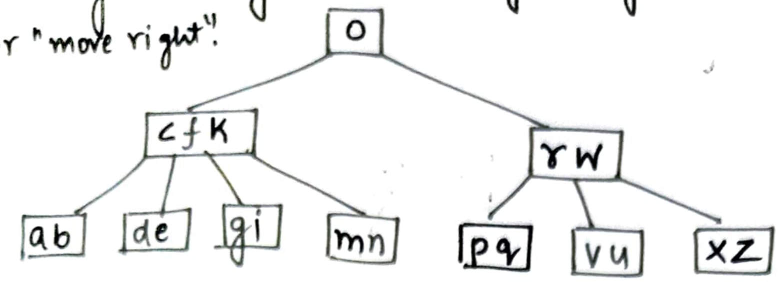
(a) Deletion of h and s.



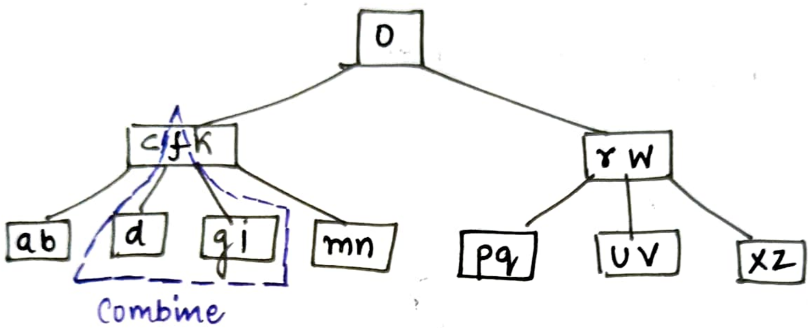
(b) After deletion of h and s.

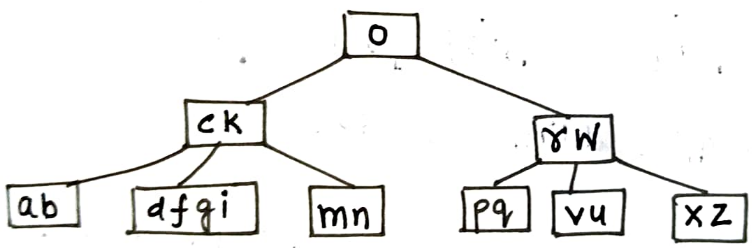
Note: Deletion of key values from nodes result in the number of key values < $\lceil m/2 \rceil - 1$ but either its right or left sibling possessess more than $\lceil m/2 \rceil - 1$ key values.

Next, let us consider the deletion of e from the B tree. Now Both of e left and right sibling contain only 2 key values in each, so no "mov left" or "move right".

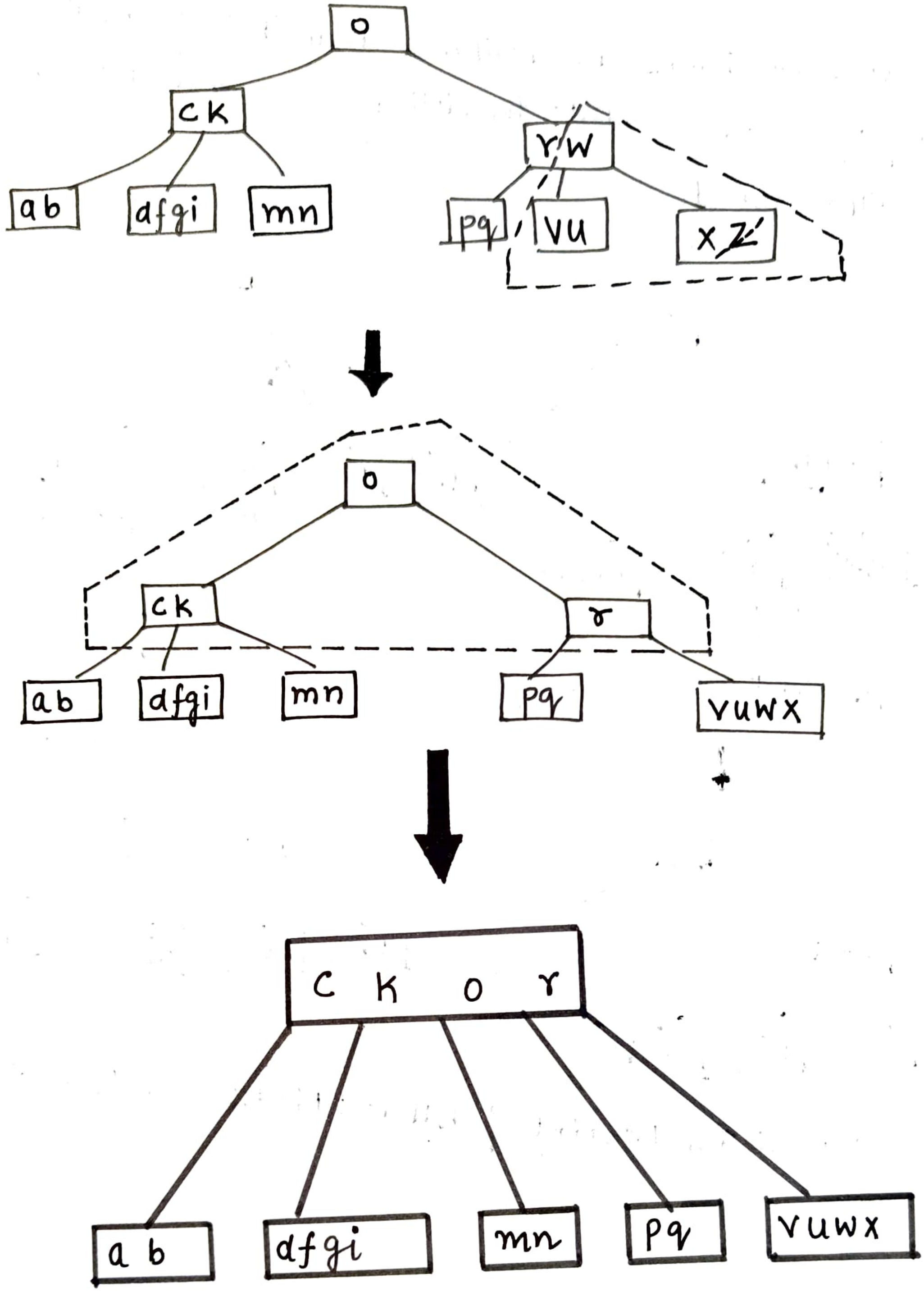


(a) Deletion of e



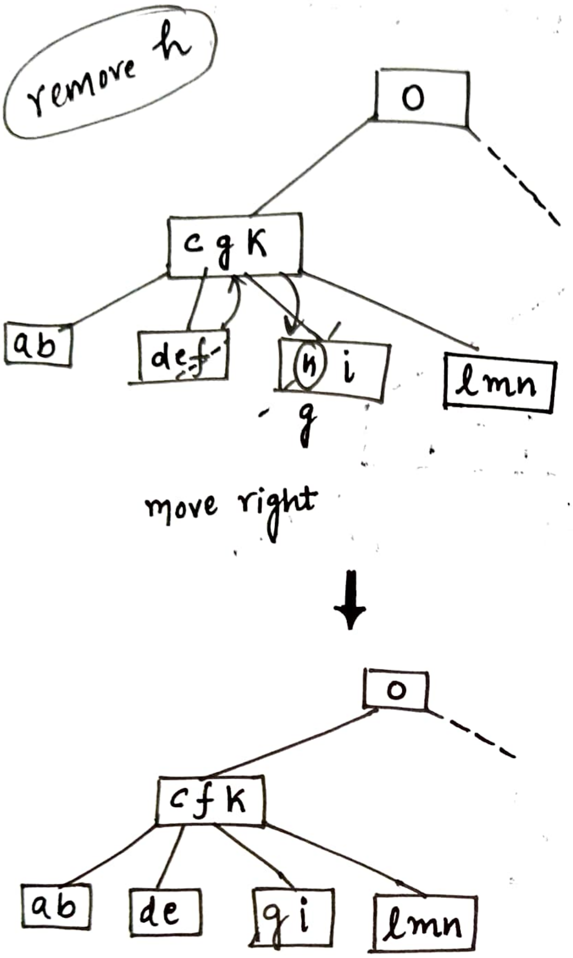


(b) After deletion of e

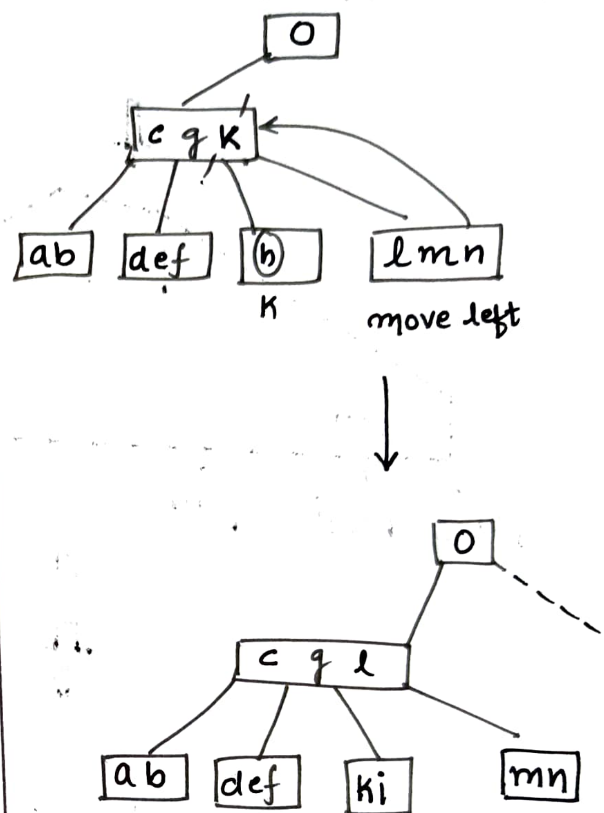


deletion of key values from nodes whose neither left nor right sibling has more than ⌈m/2⌉-1 key values.

When both the siblings are available then it is prerogative of the programmer to select the sibling from which the key value will be moved.

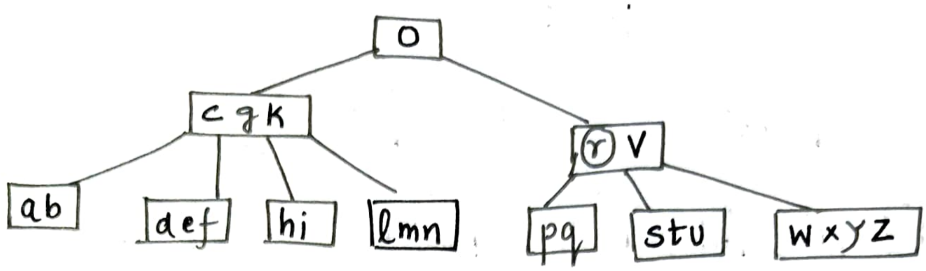


(a) Nearest left Sibling - Move Right

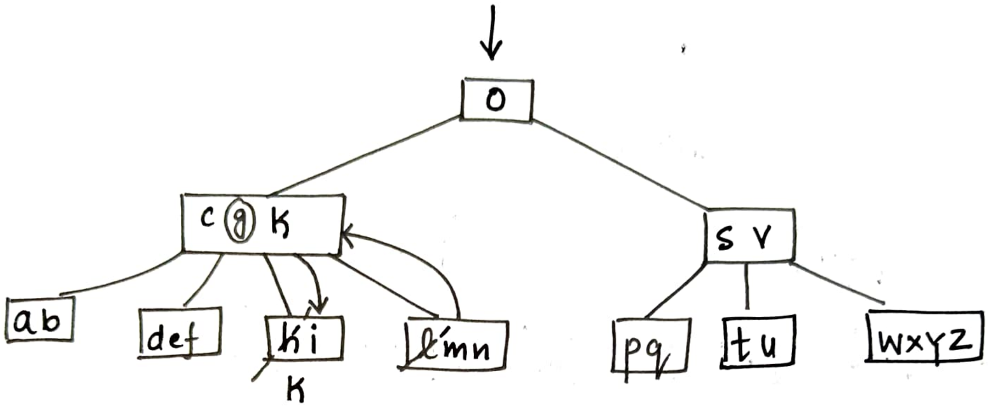


(b) Nearest Right Sibling - Move left.

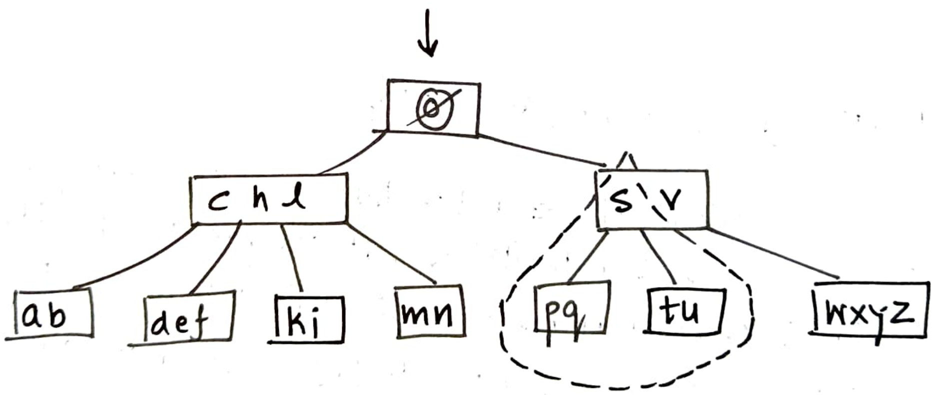
Case 2. Deletion of key value from a Non-leaf node



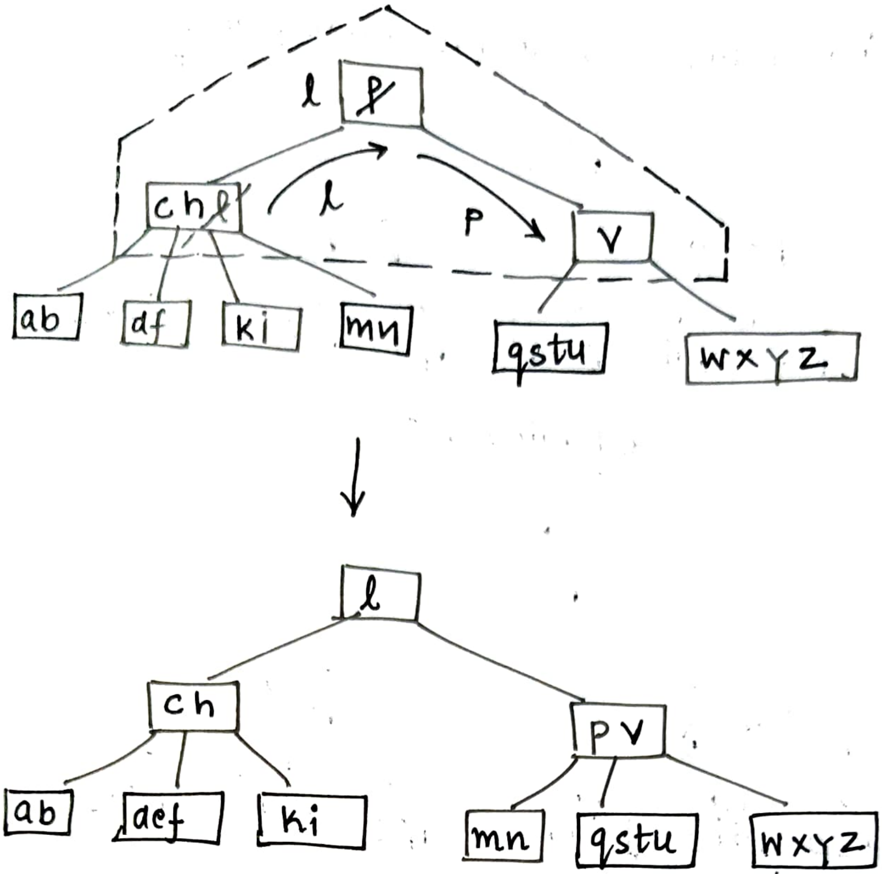
(a) Deletion of r



(b) After deletion of r and then deletion of g



(c) After deletion of g & then deletion of o



(d) After deletion of O

Deletion from a non-leaf node.

In the case of O deletions ~~of r, g and o~~, two combine took place. First combine is as usual but it has left the parent node with one key, which is v. So another 'combine' in this level with the node containing [chl], the root node [p], & the node with [v]. Here the 'combine' gives [chlpv], but a node can hold at most 4 key values. So splitting is required. Here the splitting that takes place is [ch], [l], [p v], [l] is being pushed up & becomes the root of [ch] and [pv], the left and right child respectively.

LOWER AND UPPER BOUND OF A B-TREE

Upper Bound of a B-Tree

- A B tree is always a height Balanced Tree.
- The degree of a B-Tree of order m is m, that is, the max. number of Branches that can emanate from a node is m.
- In a B-Tree of order m and height h,

$$\text{the maximum no. of nodes possible} = \sum_{i=0}^{h-1} m^i = \frac{m^h - 1}{m-1}$$

- The maximum no. of key values that a node in a B-Tree of order m can have is $m-1$.
- The maximum no. of key values that is possible in a B-Tree of order m is :

$$\frac{m^h - 1}{m-1} \times (m-1) = m^h - 1$$

Lower Bound of a B-Tree

- The root node contains atleast 2 children.
- All nodes other than the root node can have atleast $\lceil m/2 \rceil$ children.
- The minimum no. of key values in the root node is 1 (if the B-Tree is not empty)
- The minimum no. of key values in any node other than the root node is $\lceil m/2 \rceil - 1$.

The minimum no. of key values in a B-Tree of order m can be calculated as shown below:-

Level	Minimum No. of Nodes	Remark
0	1	Root node
1	2	Root node atleast 2 children
2	$2 \times \lceil m/2 \rceil$	Each node other than the root node has atleast $\lceil m/2 \rceil$ children.
3	$2 \times \lceil m/2 \rceil \times \lceil m/2 \rceil$	
⋮		
$l-1$	$2 \times \lceil m/2 \rceil^{l-1}$	Minimum no. of nodes in the last level.

All the above node so counted are non-failure nodes.

Now suppose there are atleast N no. of key values $K_1, K_2, K_3, \ldots, K_N$ where $K_i < K_{i+1}$ for $1 \leq i \leq n$.

Then the number of failure nodes $= N+1$. This is because the failure occurs for $K_i < X < K_{i+1}$, $0 < i \leq n$. This result, in the total number of failure nodes $= N+1$, hence we state the same thing as

$$N+1 = \text{No. of failure nodes in BTree}$$
$$= \text{No. of nodes in level } l$$
$$\geq 2 \times \lceil m/2 \rceil^{l-1+1}$$
$$\geq 2 \times \lceil m/2 \rceil^{l}$$
$$N \geq 2 * \lceil m/2 \rceil - 1$$

Therefore, a B-Tree of order m may contain atleast $2 * \lceil m/2 \rceil - 1$ no. of nodes.