M-WAY SEARCH TREES : All the data structures discussed so far favor data stored in the internal memory & hence support internal information retrieval. However, to favor retrieval & manipulation of data stored in external memory, viz., storage devices such as disks etc; there is a need for some special data structures. m-way search trees, B trees and B+ trees are examples of such data structures which find application in problems such as file indexing.

m-way search trees are generalized versions of binary search trees. The goal of m-way search tree is to minimize the accesses while retrieving a key from a file. However, an m-way search tree of height h calls for $O(h)$ number of accesses for an insert/delete/retrieval operation. Hence it pays to ensure that the height h is close to $\log_m(n+1)$, because the number of elements in an m-way search tree of height h ranges from a minimum of h to maximum of $m^h - 1$. This implies that an m-way search tree of n elements ranges from a minimum height of $\log_m(n+1)$ to a maximum height of n. Therefore there arises the need to maintain balanced m-way search trees. B trees are balanced m-way search trees.

Definition: An m-way search tree T may be an empty tree. If T is non empty, it satisfies the following properties:

(i) For some integer m, known as order of the tree, each node is of degree which can reach a maximum of m, in other words, each node has, at most m child nodes. A node may be represented as A_0, $(K_1, A_1), (K_2, A_2) \ldots (K_{m-1}, A_{m-1})$ where, K_i, $1 \le i \le m-1$ are the keys and A_i, $0 \le i \le m-1$ are the pointers to subtrees of T.

(2)

(ii) If a node has K child nodes where $K \le m$, then the node can have only $(K-1)$ keys $K_1, K_2 \dots K_{K-1}$ contained in the node such that $K_i < K_{i+1}$ and each of the keys partitions all the keys in the subtrees into K subsets.

(iii) For a node $A_0, (K_1, A_1), (K_2, A_2) \dots (K_{m-1}, A_{m-1})$, all key values in the subtree pointed to by A_i are less than the key K_{i+1}, $0 \le i \le m-2$, and all key values in the subtree pointed to by A_{m-1} are greater than K_{m-1}.

(iv) Each of the subtrees A_i, $0 \le i \le m-1$ are also m-way search trees.

Example 5-way search tree. Observe, each node has at most 5 child nodes & therefore has at most 4 keys contained in it.

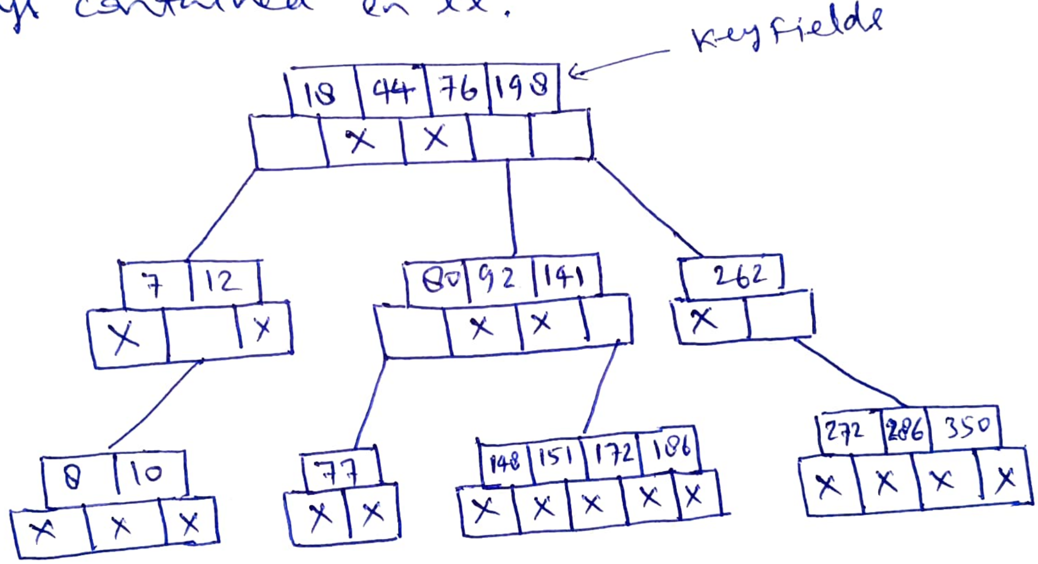


Fig 1

☐ Pointer to subtree

☒ Null Pointers.

SEARCHING an m-way Search Tree. Searching for a key in an m-way search tree is similar to that of binary search trees. To search for 77 in the 5-way search tree shown in above fig 1, we begin at the root and as $77 > 76 > 44 > 18$, move to the fourth subtree. In the root node of the fourth subtree, $77 < 80$ and therefore we move to the first subtree of the node. Since 77 is available in the only node of the subtree, we claim 77 successfully searched.

③

Insertion in an m-way Search Tree :

To insert a new element into an m-way search tree we proceed in the same way as one would in order to search for the element. To insert 6 into the 5-way search tree shown in fig 2, we proceed to search for 6 and find that we fall off the tree at the node [7, 12] with the first child node showing a null pointer. Since the node has only two keys and a 5-way search tree can accommodate up to 4 keys in a node, 6 is inserted into the node as [6, 7, 12]. But to insert 146, ~~into it~~ the node [148, 151, 172, 186] is already full, hence we open a new child node and insert 146 into it.

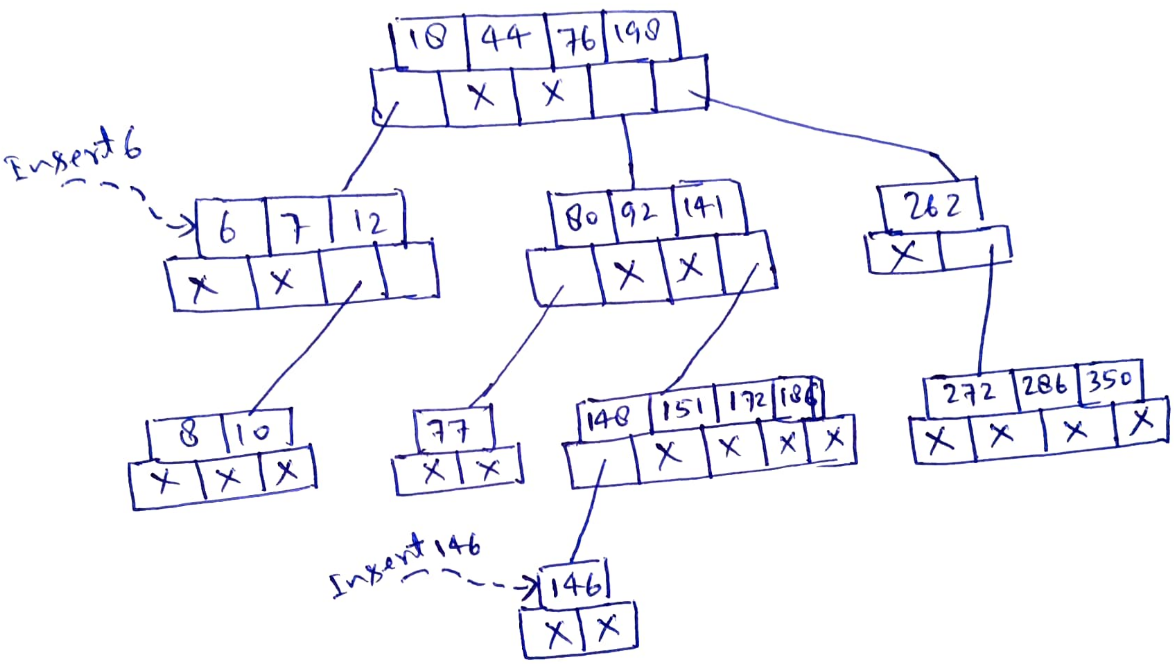
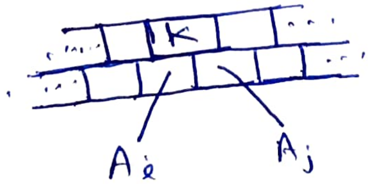


Fig 2

Deletion in an m-way Search Tree :

Let K be a key to be deleted from the m-way search tree. To delete the key we proceed as one would to search for the key. Let the node accommodating the key be as ...



K = Key

A_i, A_j = Pointers to subtrees.

If ($A_i = A_j =$ NULL) then delete K.

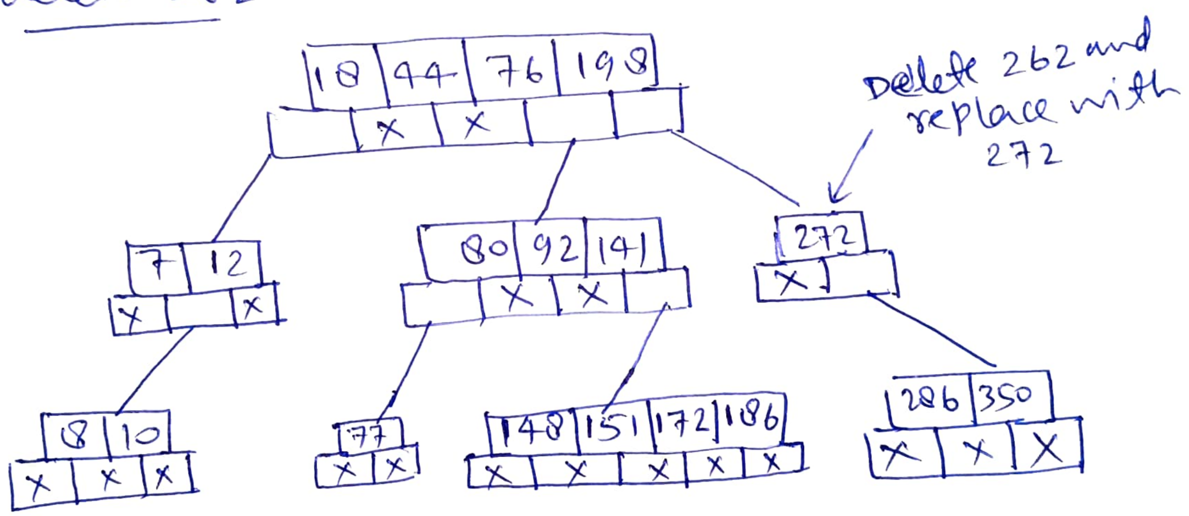
If ($A_i \neq$ NULL, $A_j =$ NULL) then choose the largest of the key elements K′ in the child node pointed to by A_i, delete the key K' and replace K by K'.

If (A_i = NULL, $A_j \neq$ NULL) then choose the smallest of the key elements K'' from the subtree pointed to by A_j, delete K'' and replace K by K''.

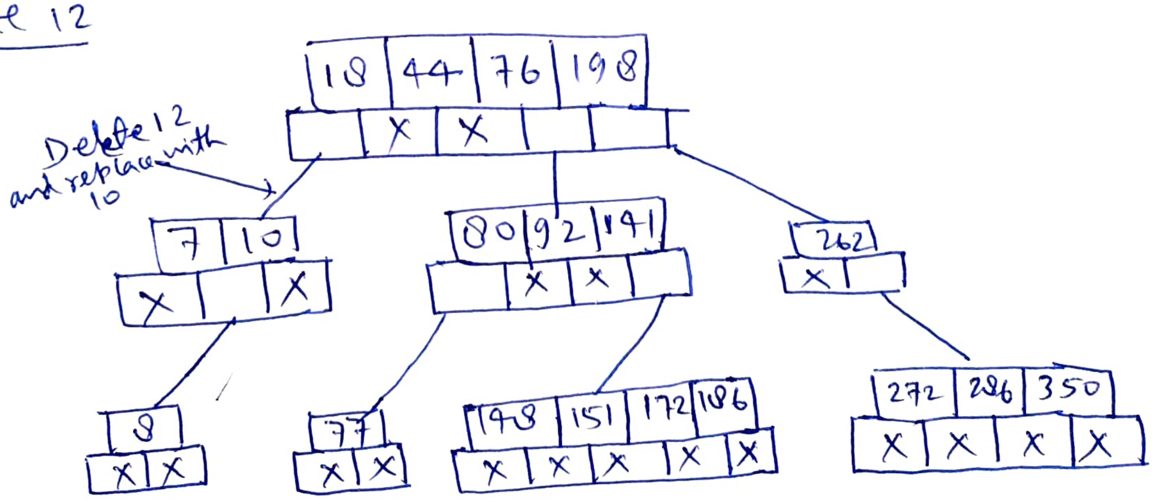
If ($A_i \neq$ NULL, $A_j \neq$ NULL) then choose either the largest of the key elements K' in the subtree pointed to by A_i or the smallest of the key elements K'' from the subtree pointed to by A_j to replace K.

→ We illustrate deletions on the 5-way search tree shown in fig1 (as above). To delete 151, we search for 151 and observe that in the leaf node [148, 151, 172, 186] where it is present, both its left subtree pointer and right subtree pointer are such that ($A_i = A_j$ = NULL). Therefore simply delete 151 and the node becomes [148, 172, 186]. Deletion of 92 also follows a similar process.

→ ~~To~~ delete 262 —



→ Delete 12

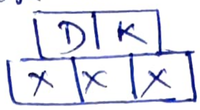


Example Construct a 3-way search tree with the following keys in the order shown as -

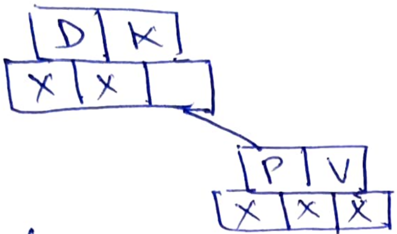
D, K, P, V, A, G

Insert D

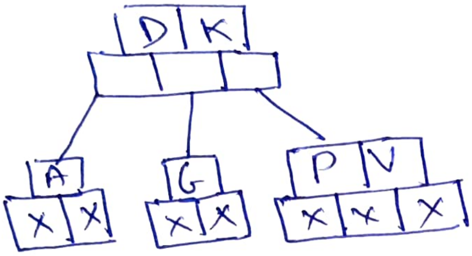
Insert K



Insert P, V



Insert A, G



B-TREES: m-way search trees have the advantage of minimizing file accesses due to their restricted height. However it is essential that the height of the tree be kept as low as possible and therefore there arises the need to maintain balanced m-way search trees. Such a balanced m-way search tree is what is defined as a B-tree.

Definition: A B-tree of order m, if non empty, is an m-way search tree in which:

(i) the root has at least two child nodes and at most m child nodes

(ii) the internal nodes except the root have at least $\lceil m/2 \rceil$ child nodes and at most m child nodes.

(iii) the number of keys in each internal node is one less than the number of child nodes and these keys partitions the keys in the subtrees of the node in a manner similar to that of m-way search trees.

(iv) All leaf nodes are on the same level.

[A B-tree of order 3 is referred to as 2-3 tree since the internal nodes are of degree 2 or 3 only]

Example B-tree of order 5.

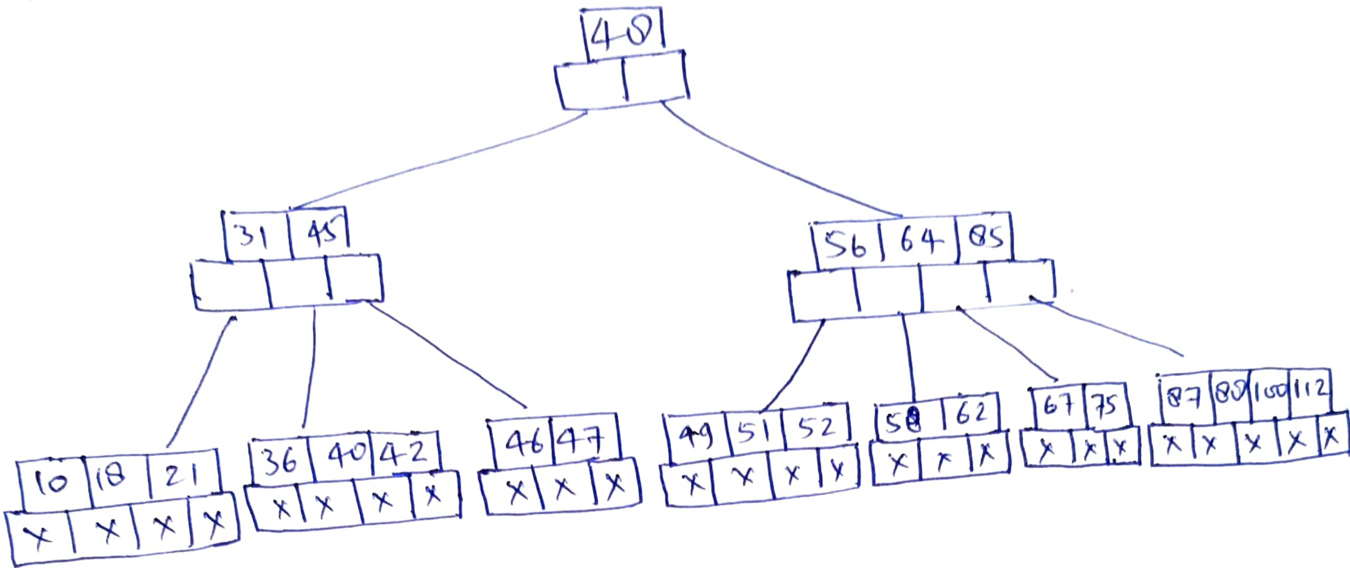


Fig 1

▭ Pointers to subtrees ▭ key fields ☒ Null Pointers

Searching a B-tree: Searching for a key in a B-tree is similar to one on an m-way search tree. The number of accesses depends on the height h of the B-tree.

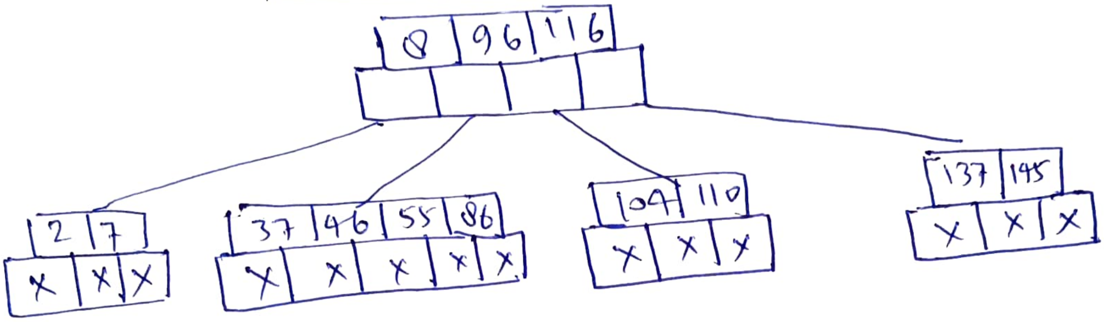
Insertion in a B-tree: The insertion of a key in a B-tree proceeds as if one were searching for the key in the tree. When the search terminates in a failure at a leaf node and tends to fall off the tree, the key is inserted according to the following procedure:

If the leaf node in which the key is to be inserted is not full, then the insertion is done in the node. A node is said to be full if it contains a maximum of (m−1) keys, given the order of the B-tree to be m.

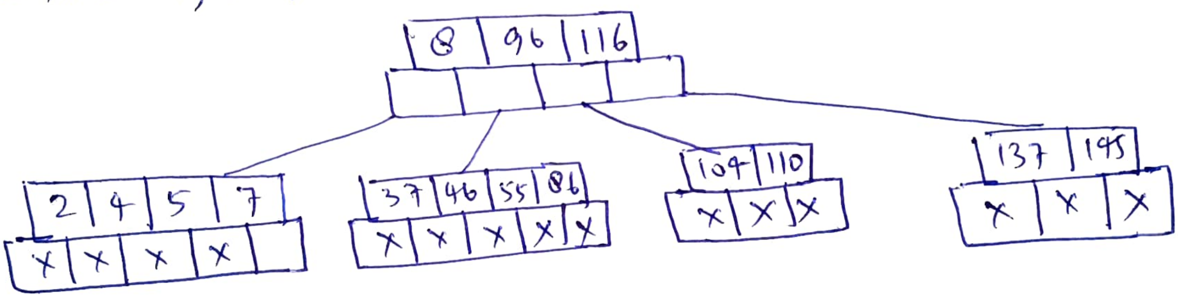
If the node were to be full, then insert the key in order into the existing set of keys in the node, split the node at its median into two nodes at the same level, pushing the median element up by one level. Note that the split nodes are only half full. Accommodate the median element in the parent node if it is not full. Otherwise repeat the same procedure and this may even call for rearrangement of the keys in the root node or the formation of a new root itself.

Thus a major observation pertaining to insertion in a B-tree is that, since the leaf nodes are all at the same level, unlike m-way search trees, the tree grows upwards.

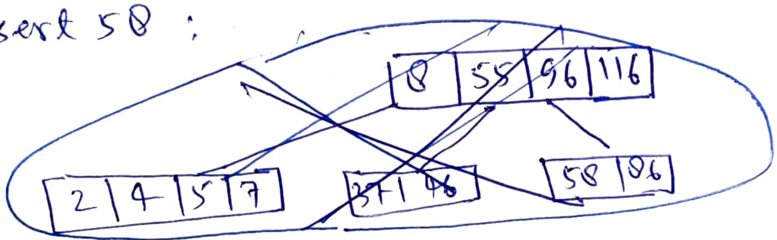
<u>Example</u> Consider the B-tree of order 5 as shown below. Insert the elements 4, 5, 58, 6 in the order given.

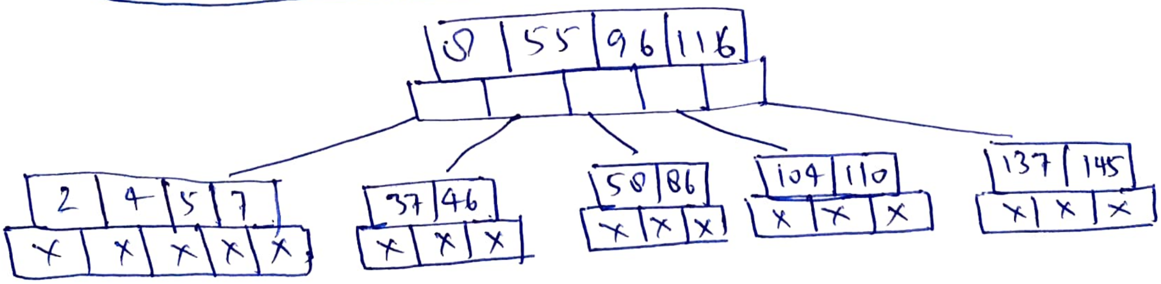


Insert 4, 5 :

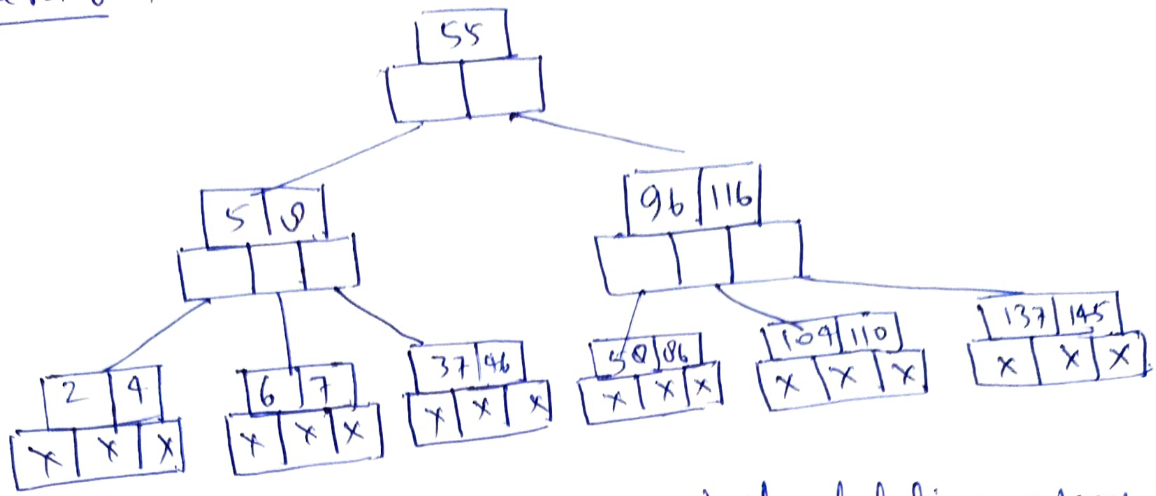


Insert 58 :





Insert 6 :

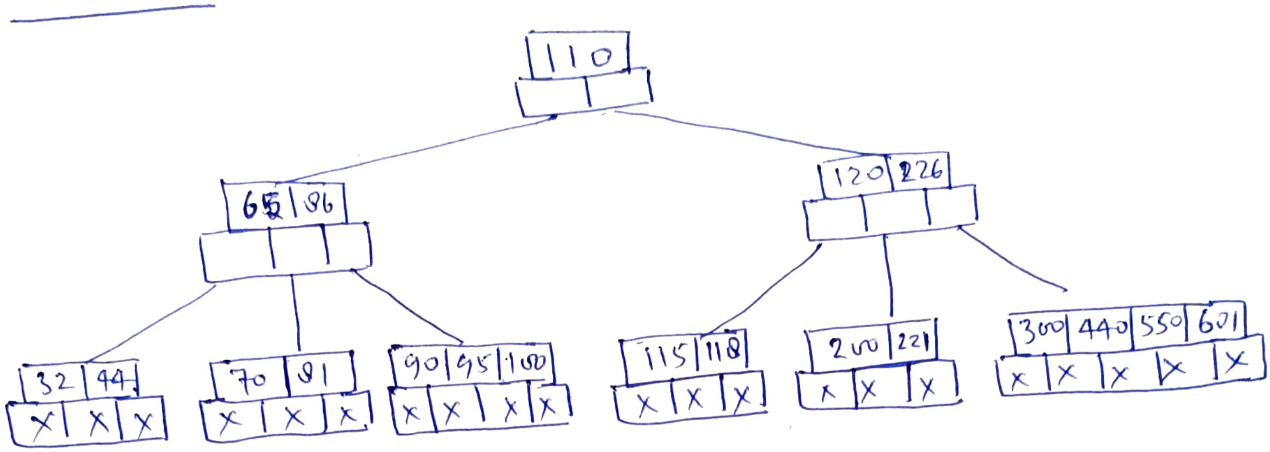


DELETION in a B-TREE :- While deleting a key it is desirable that the keys in the leaf node are removed. However when a case arises forcing a key that is an internal node to be deleted, then we promote a successor or a predecessor of the key to be deleted, to occupy the position of the deleted key and such a key is ~~bound~~ bound to occur in a leaf node.

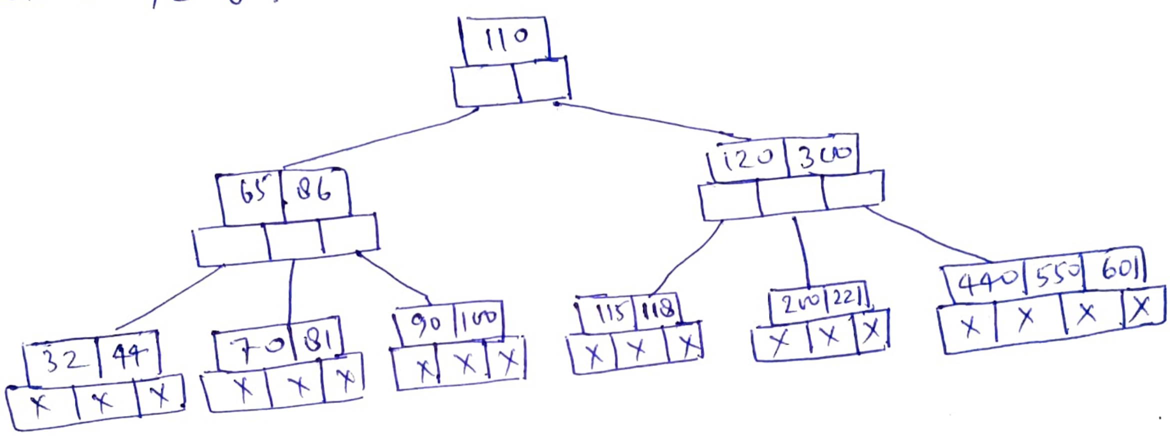
While removing a key from a leaf node, if the node contains more than the minimum number of elements, then the key can be easily removed. However, if the leaf node contains just the minimum number of elements, then scout for an element from either the left sibling node or right sibling node to fill the vacancy. If the left sibling node has more than the minimum number of keys, pull the largest key up into the parent node and move down the intervening entry from the parent node to the leaf node where the key is to be deleted. Otherwise, pull the smallest key of the right sibling node to the parent node and move down the intervening parent element to the leaf node.

If both the sibling nodes have only minimum number of entries, then create a new leaf node out of the two leaf nodes and the intervening element of the parent node, ensuring that the total number does not exceed the maximum limit for a node. If while borrowing the intervening element from the parent node, it leaves the number of keys in the parent node to be below the minimum number, then we propagate the process upwards ultimately resulting in a reduction of height of the B-tree. (9)

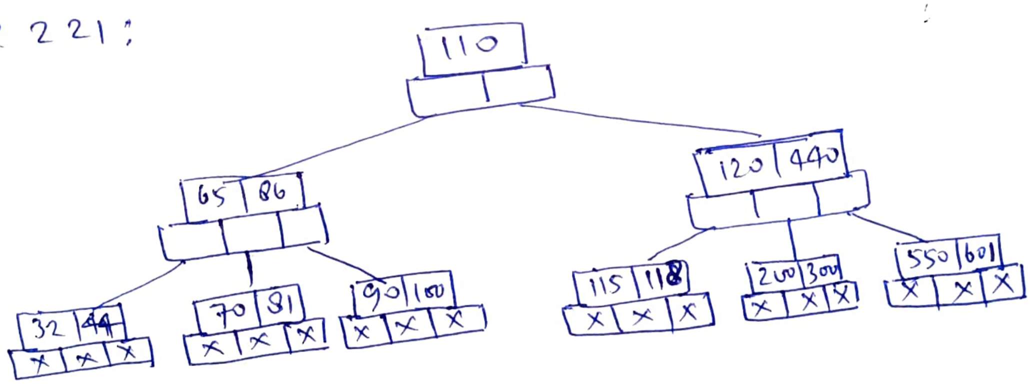
Example : B-tree of order 5 :



Delete 95, 226 :



Delete 221 :



Delete 70 :