B-Tree

Introduction

B-Tree :

A B tree is a specialized M-way tree developed by Rudolf Bayer and Ed McCreight in 1970 that is widely used for disk access.

B-Tree is a self-balanced search tree in which every node contains multiple keys and has more than two children.

the number of keys in a node and number of children for a node depends on the order of B-Tree. Every B-Tree has an m order. A B-Tree of order m is an m-way search tree.

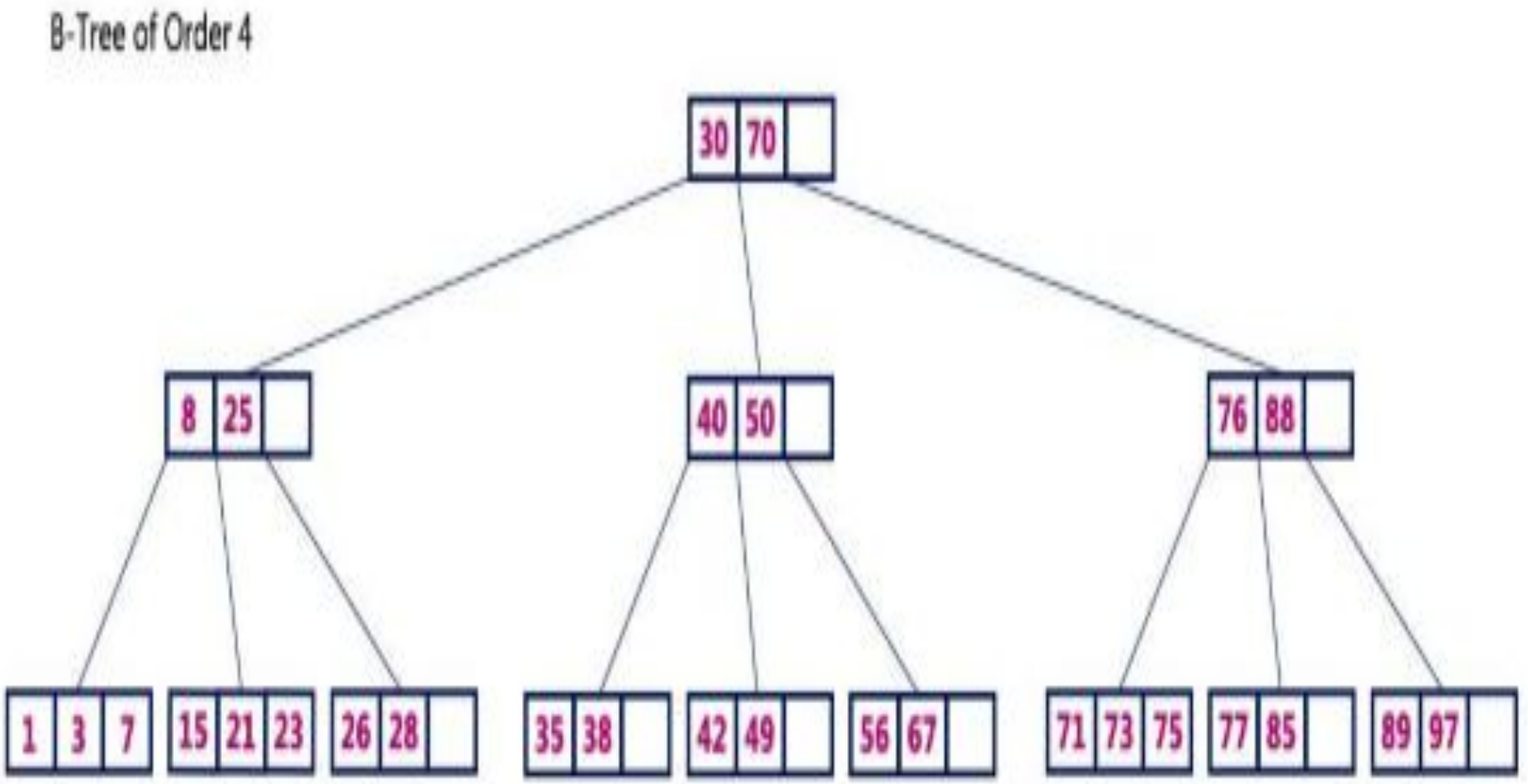
B-Tree properties

B-Tree of Order m has the following properties...

- All **leaf nodes** must be **at same level**.
- All nodes except root must have at least **[m/2] to m** no of children or **[m/2]-1** keys and maximum of **m-1** keys.
- If the root node is a non leaf node, then it must have **atleast 2** children.
- All the **key values in a node** must be in **Ascending Order**.

For example, B-Tree of Order 4 is a 4-way search tree which contains a maximum of 3 key values in a node and maximum of 4 children for a node.

Example for 4 way B-Tree



B-Tree

Operations on a B-Tree

The following operations are performed on a B-Tree...

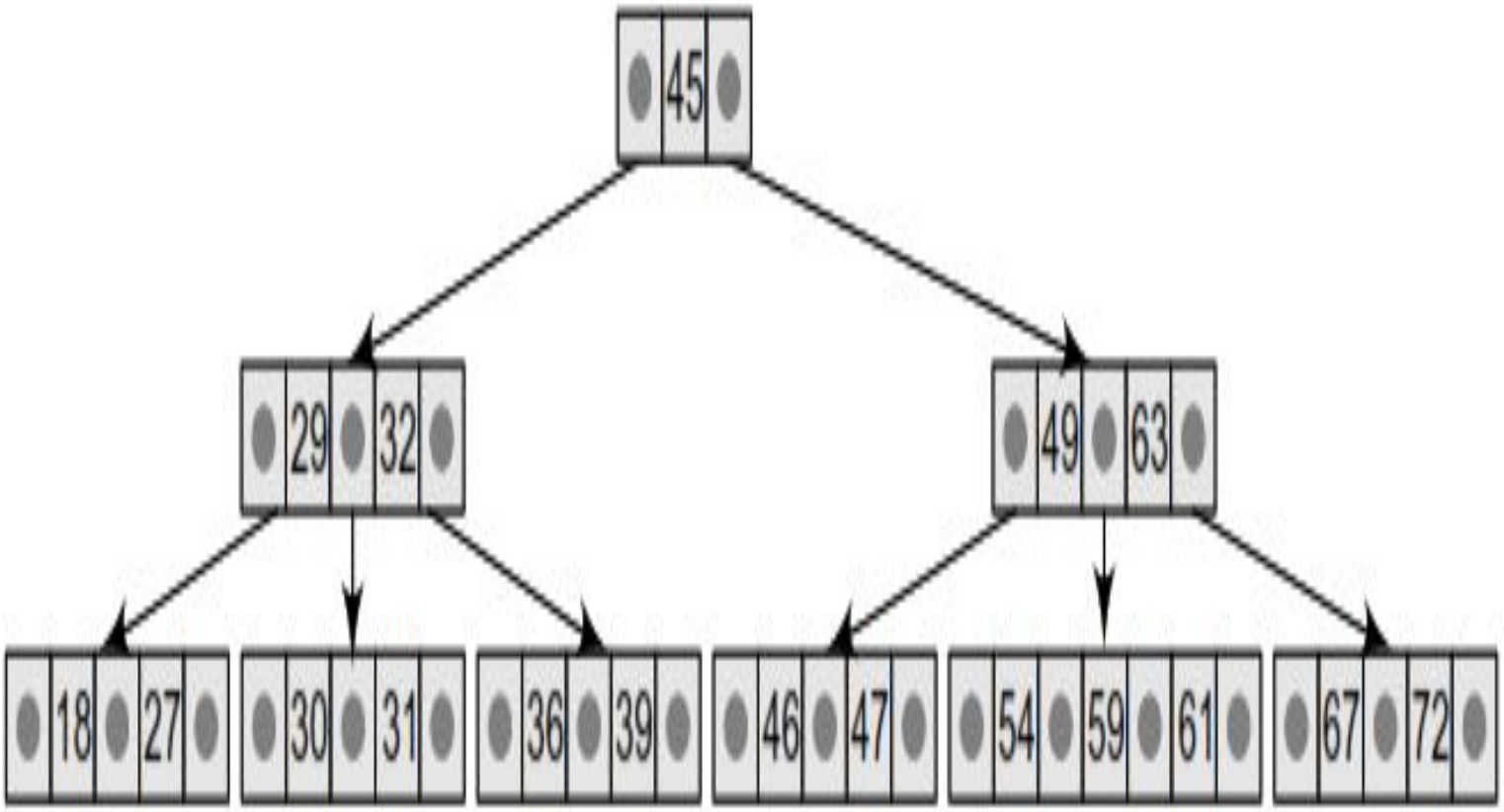
1. Search
2. Insertion
3. Deletion

Operations on B-Tree

1. Search Operation in B-Tree

The search operation in B-Tree is similar to the search operation in Binary Search Tree. In a Binary search tree, the search process starts from the root node and we make a 2-way decision every time (we go to either left subtree or right subtree). In B-Tree also search process starts from the root node but here we make an n-way decision every time. Where 'n' is the total number of children the node has.

Search in a B-Tree



Insertion in a B-Tree

In a B tree, all insertions are done at the leaf node level. A new value is inserted in the B tree using the algorithm given below.

1. Search the B tree to find the leaf node where the new key value should be inserted.
2. If the leaf node is not full, that is, it contains less than `m-1` key values, then insert the new element in the node keeping the node's elements ordered.
3. If the leaf node is full, that is, the leaf node already contains `m-1` key values, then

   (a) insert the new value in order into the existing set of keys,
   (b) split the node at its median into two nodes (note that the split nodes are half full), and
   (c) push the median element up to its parent's node. If the parent's node is already full, then split the parent node by following the same steps.

Insertion in a B-Tree

Example: Construct a **B-Tree of Order 3** by inserting the following set of numbers.

{ 1,2, 3,4,5,6,7,8,9,10}

1. Find the minimum elements in a internal and leaf nodes
   m/2 to m number of children or **m/2-1 to m-1** number of data items

insert(1)

Since '1' is the first element into the tree that is inserted into a new node. It acts as the root node.

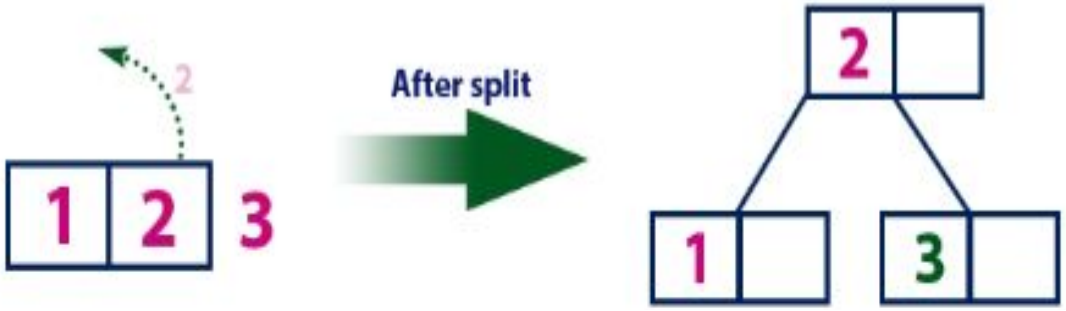
insert(2)

Element '2' is added to existing leaf node. Here, we have only one node and that node acts as root and also leaf. This leaf node has an empty position. So, new element (2) can be inserted at that empty position.

Insertion in a B-Tree

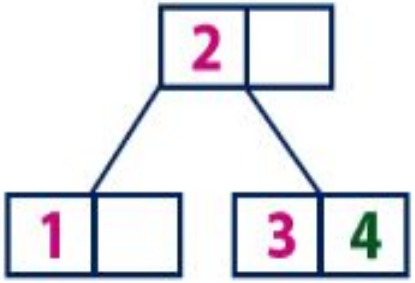
insert(3)

Element '3' is added to existing leaf node. Here, we have only one node and that node acts as root and also leaf. This leaf node doesn't has an empty position. So, we split that node by sending middle value (2) to its parent node. But here, this node doesn't has parent. So, this middle value becomes a new root node for the tree.



insert(4)

Element '4' is larger than root node '2' and it is not a leaf node. So, we move to the right of '2'. We reach to a leaf node with value '3' and it has an empty position. So, new element (4) can be inserted at that empty position.



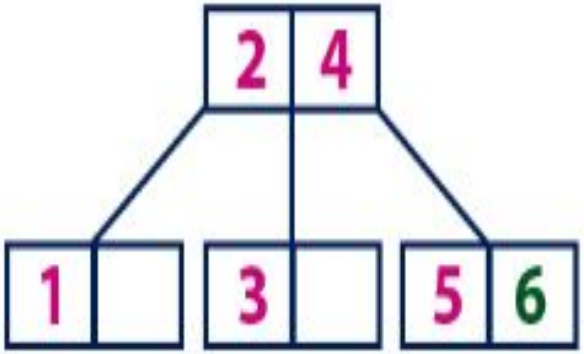
Insertion in a B-Tree

insert(5)

Element '5' is larger than root node '2' and it is not a leaf node. So, we move to the right of '2'. We reach to a leaf node and it is already full. So, we split that node by sending middle value (4) to its parent node (2). There is an empty position in its parent node. So, value '4' is added to node with value '2' and new element '5' added as new leaf node.

insert(6)

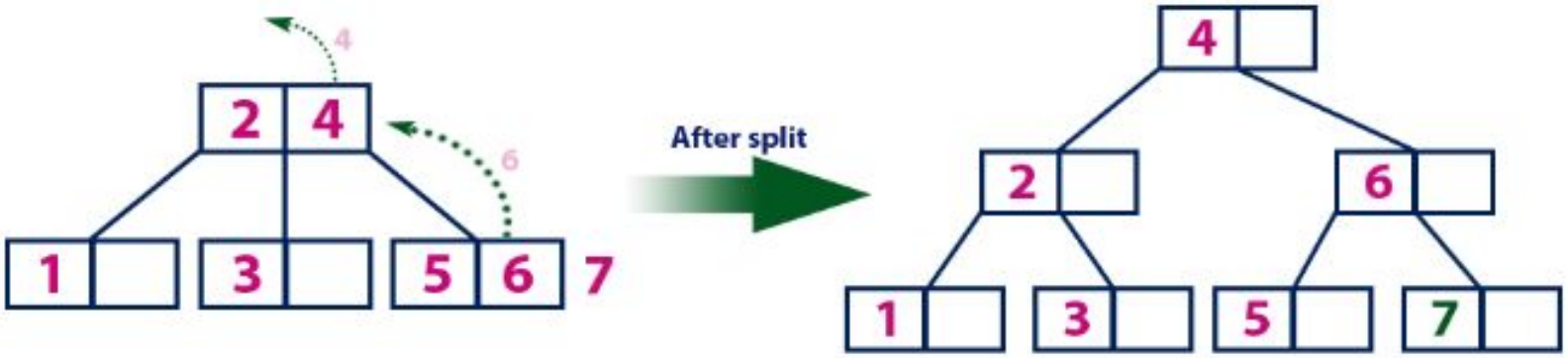
Element '6' is larger than root node '2' & '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a leaf node with value '5' and it has an empty position. So, new element (6) can be inserted at that empty position.



Insertion in a B-Tree

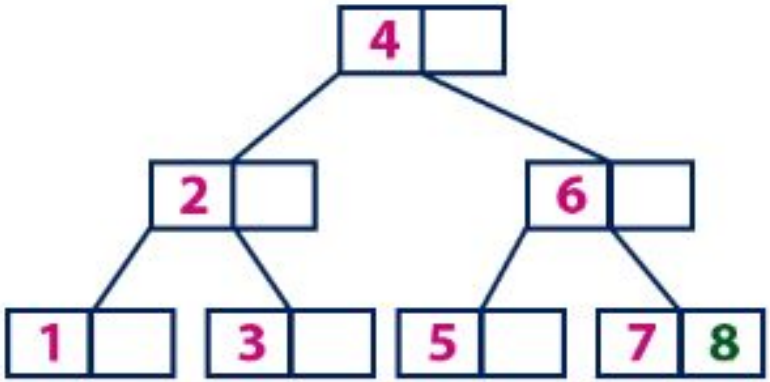
insert(7)

Element '7' is larger than root node '2' & '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a leaf node and it is already full. So, we split that node by sending middle value (6) to its parent node (2&4). But the parent (2&4) is also full. So, again we split the node (2&4) by sending middle value '4' to its parent but this node doesn't have parent. So, the element '4' becomes new root node for the tree.



insert(8)

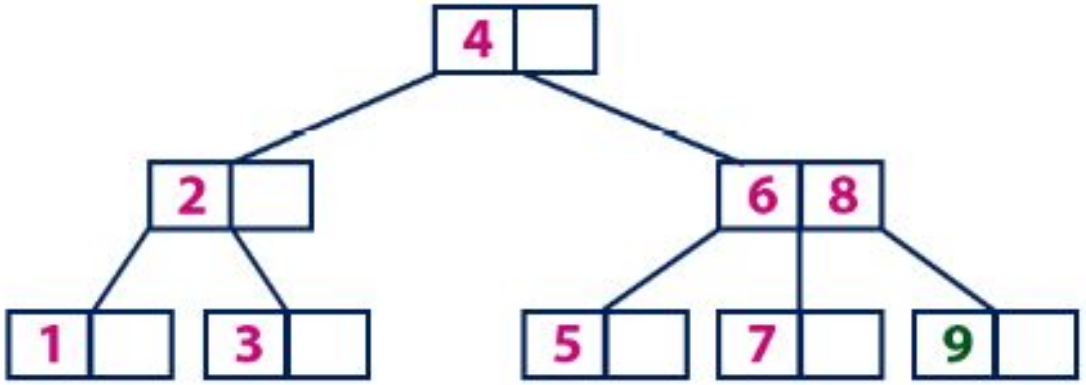
Element '8' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with value '6'. '8' is larger than '6' and it is also not a leaf node. So, we move to the right of '6'. We reach to a leaf node (7) and it has an empty position. So, new element (8) can be inserted at that empty position.



Insertion in a B-Tree

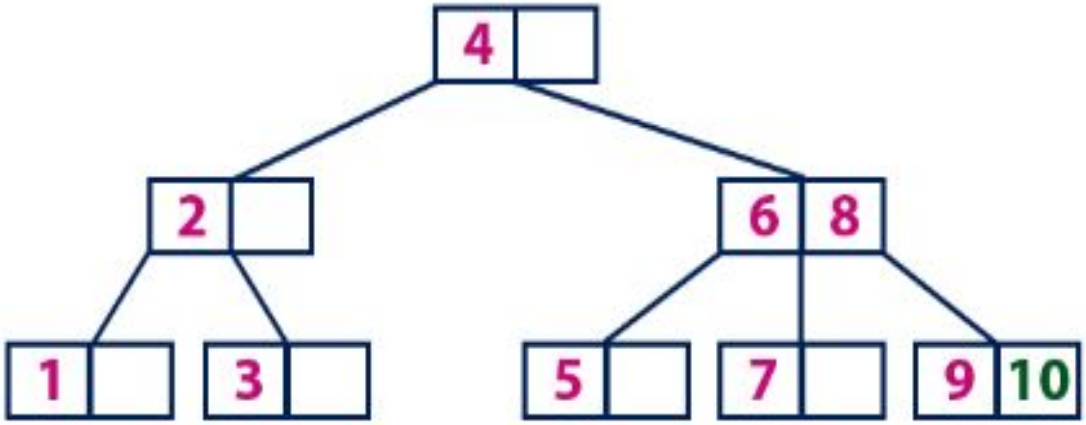
insert(9)

Element '9' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with value '6'. '9' is larger than '6' and it is also not a leaf node. So, we move to the right of '6'. We reach to a leaf node (7 & 8). This leaf node is already full. So, we split this node by sending middle value (8) to its parent node. The parent node (6) has an empty position. So, '8' is added at that position. And new element is added as a new leaf node.



insert(10)

Element '10' is larger than root node '4' and it is not a leaf node. So, we move to the right of '4'. We reach to a node with values '6 & 8 '. '10' is larger than '6 & 8' and it is also not a leaf node. So, we move to the right of '8'. We reach to a leaf node (9). This leaf node has an empty position. So, new element '10' is added at that empty position.



Operation in a B-Tree

3. Search/Deletion Operation in B-Tree

First Search for the value to delete and if it is found then delete the same as per the following. Otherwise search is unsuccessful and the deletion not possible.

a) Deletion from a leaf node

1. If the value is in a leaf node or terminal node, simply delete it from the node.
2. If underflow happens, try to get the value from right sibling if it contains more than the minimum number, if any. Otherwise try to take the value from left sibling if it has more than minimum number.
3. If it is under flow both left or right, then merge left and right siblings as one node.

Deletion in a B-Tree

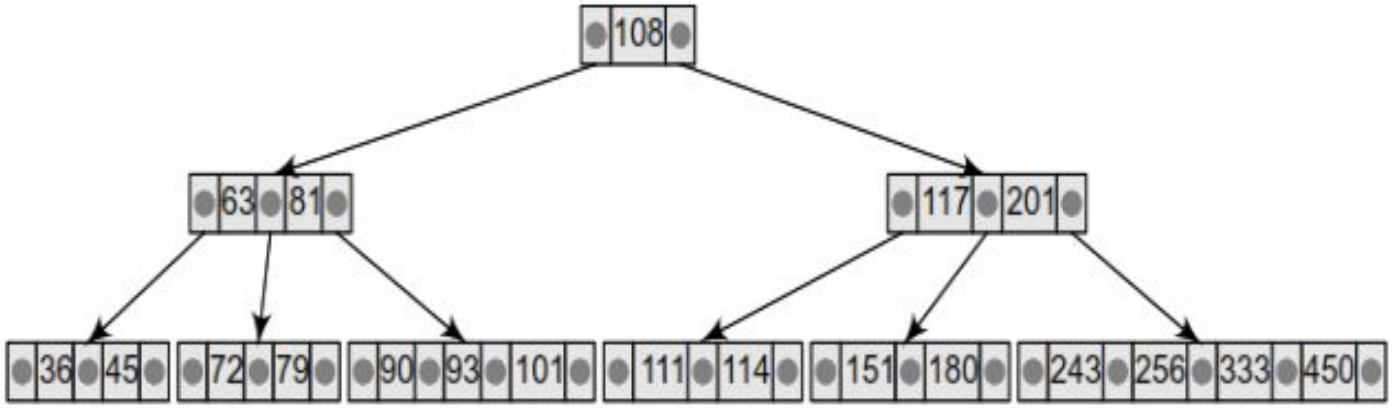
b) Deletion from an internal node

Each element in an internal node acts as a separation value for two subtrees, therefore we need to find a replacement for separation. Note that the smallest element in the right subtree is greater than the separator. Likewise, the largest element in the left subtree is smaller than the separator. Both of those elements are in leaf nodes or child nodes, and either one can be the new separator for the two sub trees.

1. Choose a new separator (first try from the right child left most value , if it is not undeflow other wise from the left subtree (right most value) if it is not under flow and replace the element to be deleted with this new separator element.
2. If not possible the replacement of deleted element in the above step, simply merge the parent node and both left and right children's as a single node.

Deletion in a B-Tree

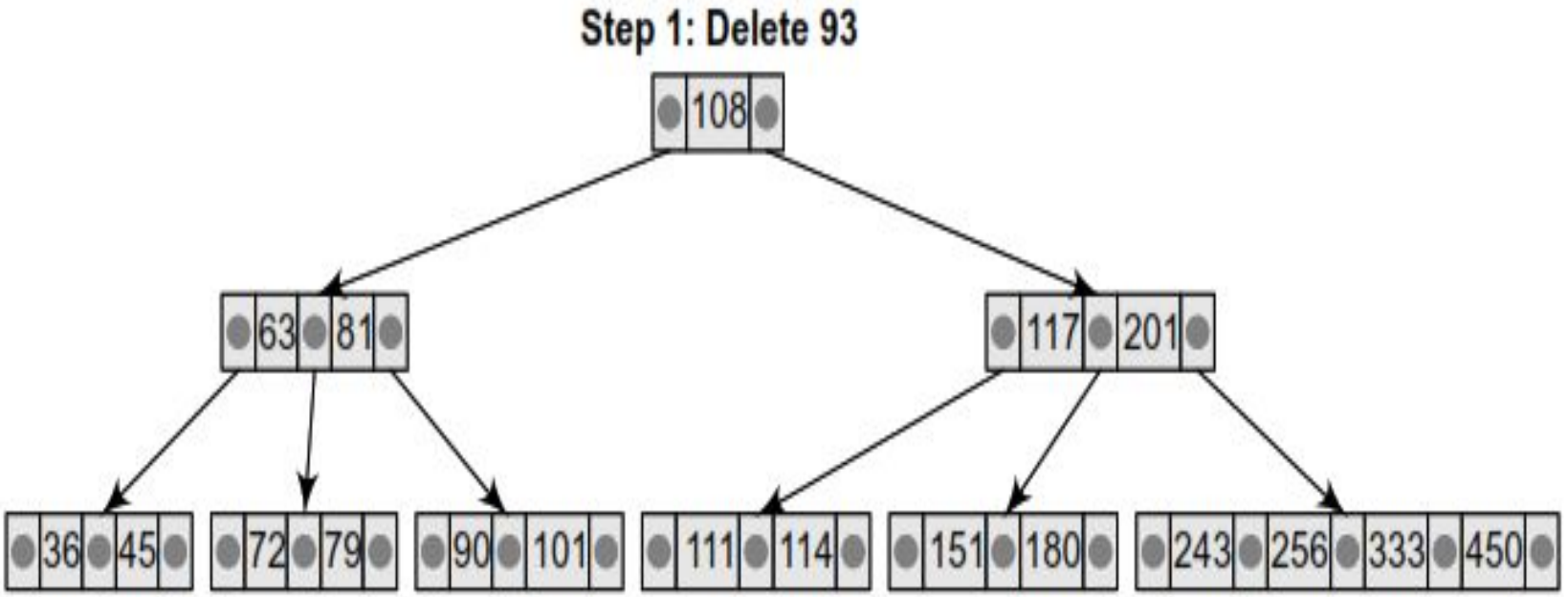
Example :- Consider the following B tree of order 5 and delete values 93, 201, 180, and 72 from it

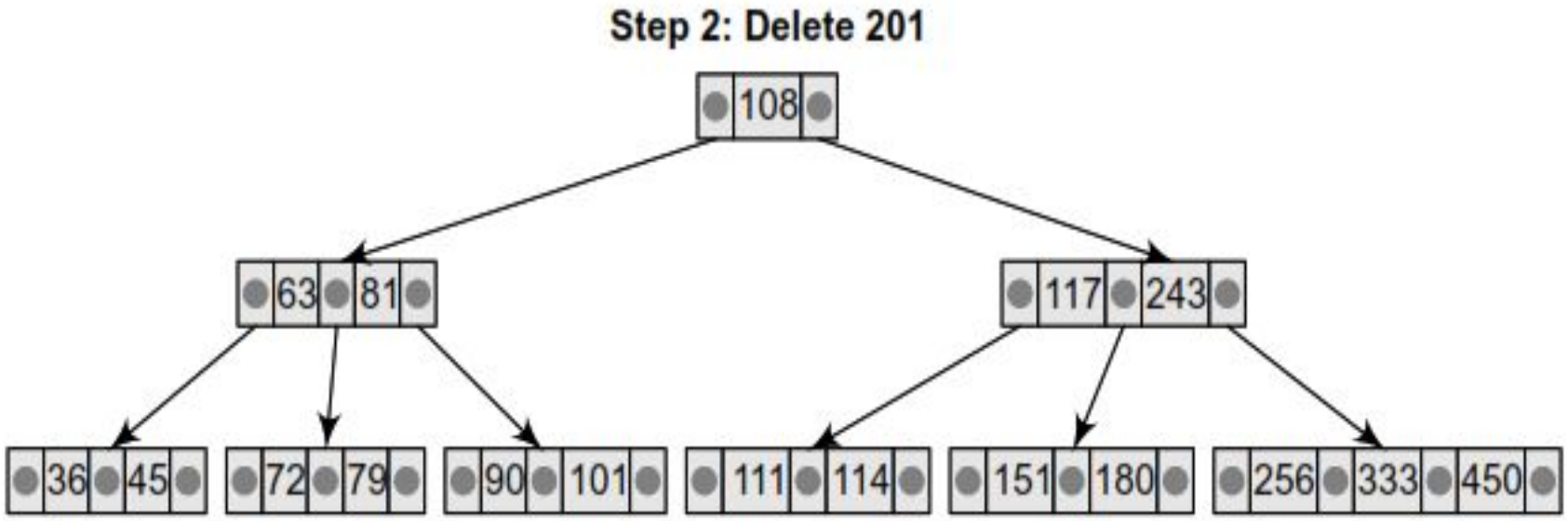


1. Find the minimum elements in a internal and leaf nodes
   m/2 to m number of children or **m/2-1 to m-1** number of data items

   minimum is 2 data item and maximum 4 items

Deletion in a B-Tree





Deletion in a B-Tree