Sairam INSTITUTIONS

Sri SAI RAM ENGINEERING COLLEGE

INSTITUTE OF TECHNOLOGY

West Tambaram, Chennai - 44

YEAR	SEM
II	III

CS8391

DATA STRUCTURES
(Common to CSE & IT)

UNIT No. 3
NON LINEAR DAT A STRUCTURES – TREES

3.6.1 – B – TREE – INSERTION
3.6.1 – B – TREE - DELETIONN

Version: 1.XX

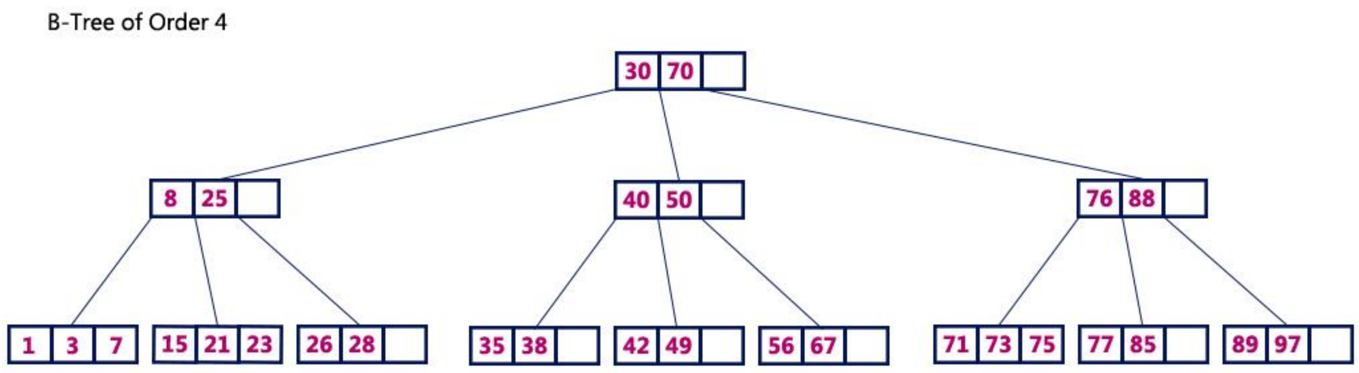
Introduction to B TREE and its operations

A **B tree** is designed to store sorted data and allows search, insertion and deletion operation to be performed in logarithmic time. As In multiway search tree, there are so many nodes which have left subtree but no right subtree. Similarly, they have right subtree but no left subtree. As is known, access time in the tree is totally dependent on the level of the tree. So our aim is to minimize the access time which can be through balance tree only.

For balancing the tree each node should contain n/2 keys. So the B tree of order n can be defined as:

1. All leaf nodes should be at same level.
2. All leaf nodes can contain maximum n-1 keys.
3. The root has at least two children.
4. The maximum number of children should be n and each node can contain k keys. Where, k<=n-1.
5. Each node has at least n/2 and maximum n nonempty children.
6. Keys in the non-leaf node will divide the left and right sub-tree where the value of left subtree keys will be less and value of right subtree keys will be more than that particular key.

Example



Operations on a B-Tree

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

1. **Search**
2. **Insertion**
3. **Deletion**

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. In a B-Tree, the search operation is performed with **O(log n)** time complexity. The search operation is performed as follows...

- **Step 1 -** Read the search element from the user.
- **Step 2 -** Compare the search element with first key value of root node in the tree.
- **Step 3 -** If both are matched, then display "Given node is found!!!" and terminate the function

- **Step 4 -** If both are not matched, then check whether search element is smaller or larger than that key value.
- **Step 5 -** If search element is smaller, then continue the search process in left subtree.
- **Step 6 -** If search element is larger, then compare the search element with next key value in the same node and repeate steps 3, 4, 5 and 6 until we find the exact match or until the search element is compared with last key value in the leaf node.
- **Step 7 -** If the last key value in the leaf node is also not matched then display "Element is not found" and terminate the function.

Insertion Operation in B-Tree

In a B-Tree, a new element must be added only at the leaf node. That means, the new keyValue is always attached to the leaf node only. The insertion operation is performed as follows...

- **Step 1 -** Check whether tree is Empty.
- **Step 2 -** If tree is **Empty**, then create a new node with new key value and insert it into the tree as a root node.
- **Step 3 -** If tree is **Not Empty**, then find the suitable leaf node to which the new key value is added using Binary Search Tree logic.
- **Step 4 -** If that leaf node has empty position, add the new key value to that leaf node in ascending order of key value within the node.
- **Step 5 -** If that leaf node is already full, **split** that leaf node by sending middle value to its parent node. Repeat the same until the sending value is fixed into a node.
- **Step 6 -** If the spilting is performed at root node then the middle value becomes new root node for the tree and the height of the tree is increased by one.

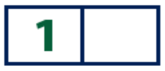
Example

Construct a **B-Tree of Order 3** by inserting numbers from 1 to 10.

Construct a B-Tree of order 3 by inserting numbers from 1 to 10.

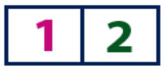
insert(1)

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



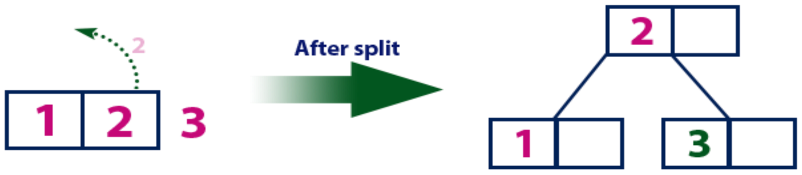
insert(2)

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.



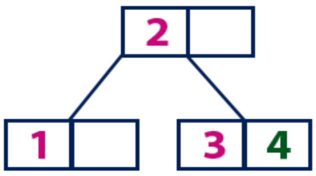
insert(3)

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.



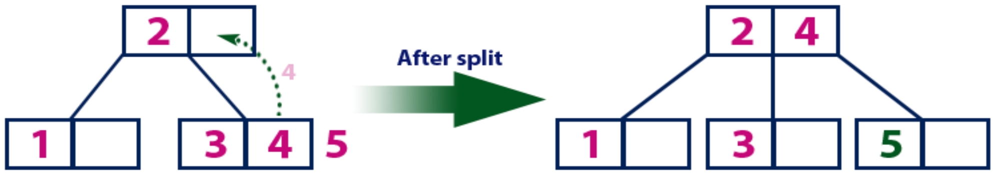
insert(4)

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.



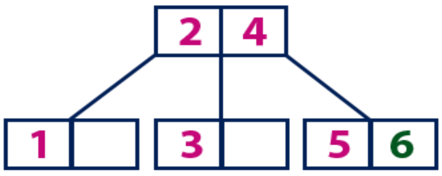
insert(5)

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.



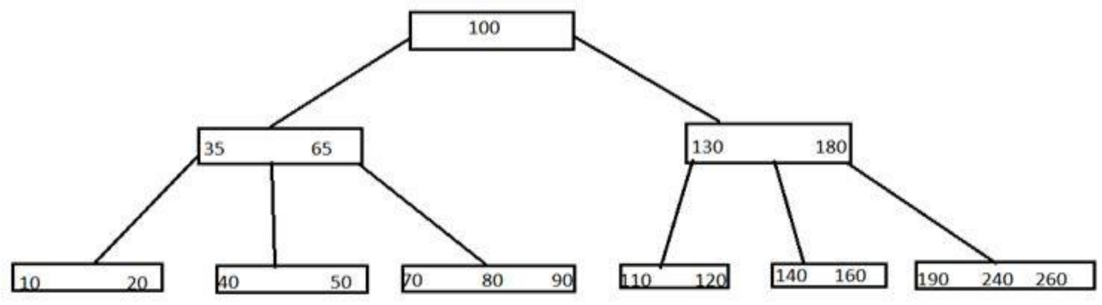
insert(6)

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.



EXAMPLE NO : 02

Let us take a B-tree of order 5,



Here, we can see all leaf nodes are at same level. All non-leaf nodes have no empty sub-tree and they have keys 1 less than the number of their children.

Operations performed on B Tree

1. **Insertion in B-Tree**
2. **Deletion from B-Tree**

1) Insertion in B-Tree

The insertion of a key in a B tree requires the first traversal in B-tree. Through the traversal, it is easy to find that key which needs to be inserted is already existed or not. There are basically two cases for inserting the key that are:

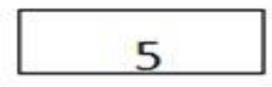
1. Node is not full
2. Node is already full

If the leaf node in which the key is to be inserted is not full, then the insertion is done in the node.

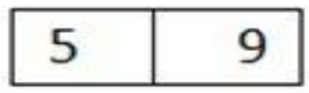
If the node were to be full then insert the key in order into existing set of keys in the node, split the node at its median into two nodes at the same level, pushing the median element up by one level.

Let us take a list of keys and create a B-Tree: 5,9,3,7,1,2,8,6,0,4

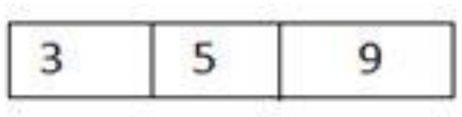
1) Insert 5



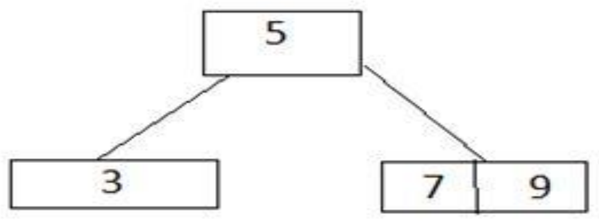
2) Insert 9: B-tree insert simply calls B tree insert non-full, putting 9 to the right of 5.



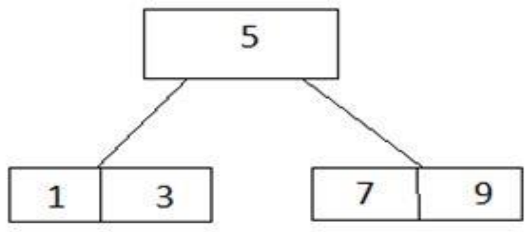
3) Insert 3: Again B-tree insert non-full is called



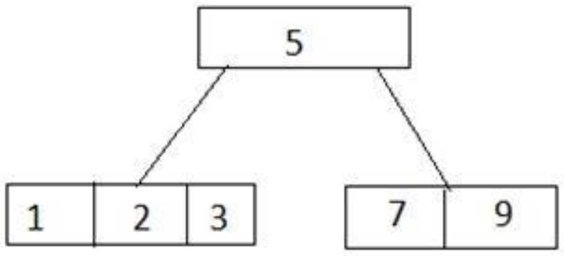
4) Insert 7: Tree is full. We allocate a new empty node, make it the root, split a former root, and then pull 5 into a new root.



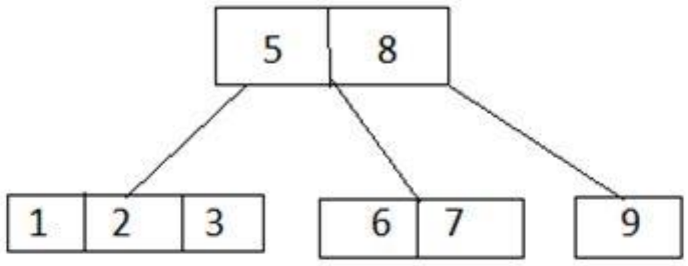
5) Insert 1: It goes with 3



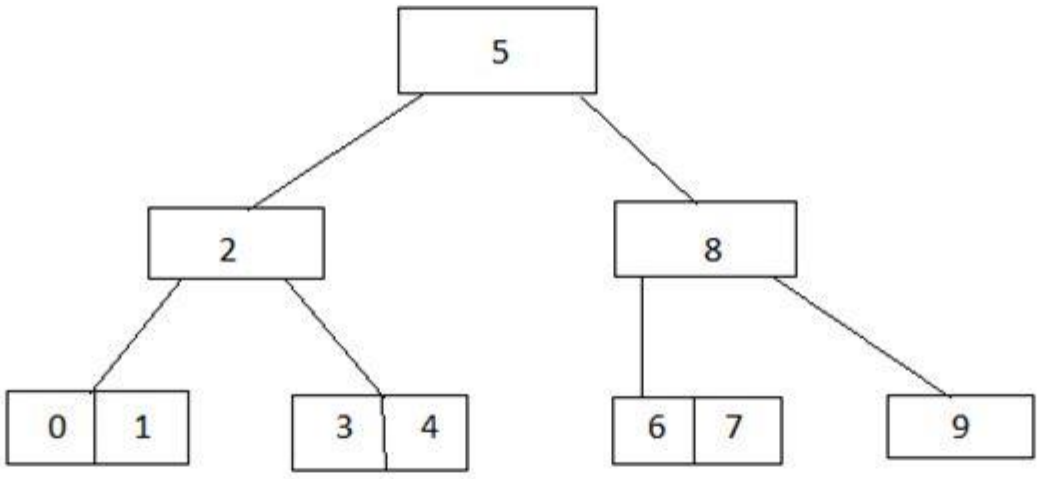
6) Insert 2: It goes with 3



7) Insert 8, 6: As firstly 8 goes with the 9 and then 6 would go with 7, 8, 9 but that node is full. So we split it bring its middle child into the root.



8) Insert 0, 4: 0 would go with the 1, 2, and 3 which are full, so we split it sending the middle child up to the root. Now it would be nice to just stick 4 in with 3, but the B-tree algorithm requires us to split the full root. Now we can insert 4 assured that future insertion will work.

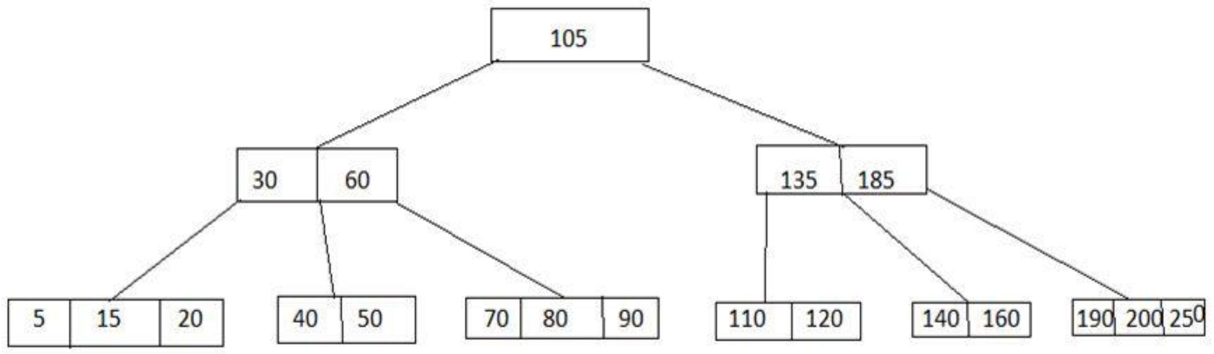


2) Deletion from B Tree

Deletion of the key also requires the first traversal in B tree, after reaching on a particular node two cases may be occurred that are:

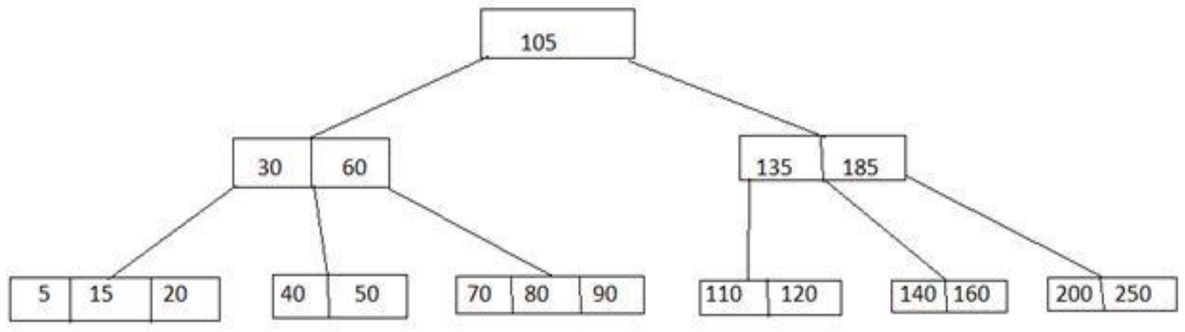
1. Node is leaf node
2. Node is non leaf node

Example: Let us take a B tree of order 5



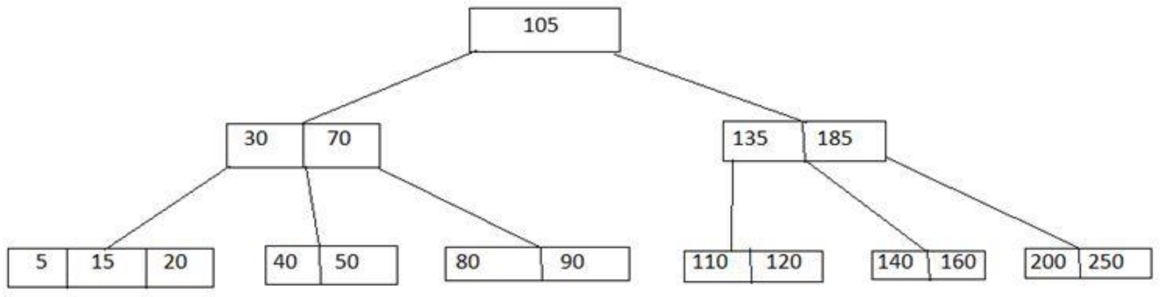
1)

Delete 190: Here 190 is in leaf node, so delete it from only leaf node.



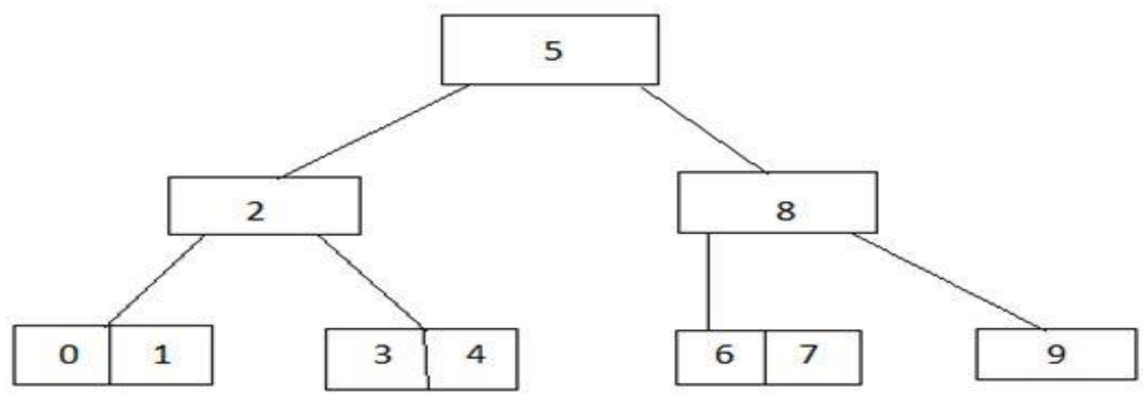
2)

Delete 60: Here 60 is in non leaf node. So first it will be deleted from the node and then the element of the right child will come in that node.



Applications of B Tree

The main application of a B tree is the organization of a huge collection of a data into a file structure.In this insertion, deletion and modification can be carried out perfectly and efficiently.



Full, so we split it sending the middle child up to the root. Now it would be nice to just stick 4 in with 3, but the B-tree algorithm requires us to split the full root. Now we can insert 4 assured that future insertion will work.