```
            return root.left
        }
        // Both left and right child exists
        else
        {
            TreeNode selected_node = root
            while ( selected_node.left != NULL )
                selected_node = selected_node.left
            root.val = selected_node.val
            root.left = delete_element(root.left, selected_node.val)
        }
    }
    return root
}
```

Applications of Tree Data Structure

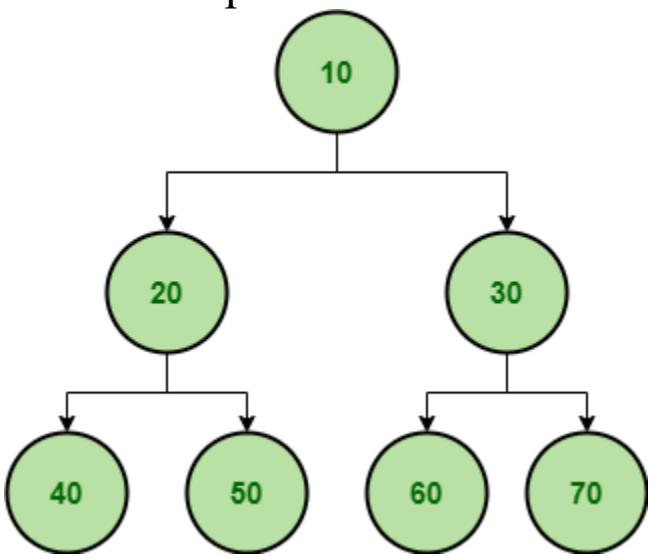
As we've mentioned above, tree data structure stores data in a hierarchical manner. Nodes are arranged at multiple levels.

- Information stored in the computer is in a hierarchical manner. There are drives that contain multiple folders. Each folder can have multiple subfolders. And then there are files like documents, images, etc.

- Searching for a node in a tree data structure (such as BST) is relatively faster, and so are other operations such as insertion and deletion, given its ordered structure.
- Trie is a type of tree data structure that is used to insert, search, and start strings. (Example - Contact list on your phone)
- We use a tree data structure to store data in routing tables in the routers.
- Programmers also use syntax trees in compilers to verify the syntax of the programs they write.

Applications of Binary Trees:

A binary tree is a tree that has at most two children for any of its nodes. There are several types of binary trees. To learn more about them please refer to the article on "**Types of binary tree**"



Sample Binary Tree

Application of Binary Trees:

- Huffman coding tree is an application of binary trees that are used in data compression algorithms.
- In compilers, Expression Trees are used which are applications of binary trees.

- Priority Queue is another application of binary tree that is used to search maximum or minimum in O(log N) time complexity.
- Represent hierarchical data.
- used in editing software like Microsoft Excel and spreadsheets.
- useful for indexing segmented at the database is useful in storing cache in the system,
- syntax trees are used for most famous compilers for programming like GCC, and AOCL to perform arithmetic operations.
- for implementing priority queues.
- used to find elements in less time (binary search tree)
- used to enable fast memory allocation in computers.
- to perform encoding and decoding operations.

Real-time applications of Binary Trees:

- DOM in HTML.
- File explorer.
- Used as the basic data structure in Microsoft Excel and spreadsheets.
- Editor tool: Microsoft Excel and spreadsheets.
- Evaluate an expression
- Routing Algorithms

Advantages of Binary Tree:

- The searching operation in a binary tree is very fast.
- The representation of a binary tree is simple and easy to understand.

- Traversing from a parent node to its child node and vice-versa is efficiently done.
- simple to implement
- easy to understand.
- a hierarchical structure.
- reflect structural relationships that are present in the data set
- easy to insert data than in another data store.
- easy to store data in memory management.
- user can many nodes
- executions are fast.
- store an arbitrary number of data values.

Disadvantages of Binary Tree:

- In binary tree traversals, there are many pointers that are null and hence useless.
- The access operation in a Binary Search Tree (BST) is slower than in an array.
- A basic option is dependent on the height of the tree.
- Deletion node not easy.
- A basic option is based on the height of tree.

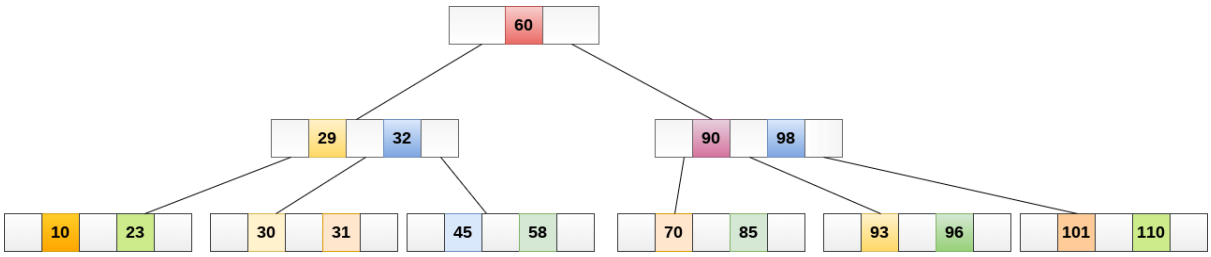
B Tree:

B Tree is a specialized m-way tree that can be widely used for disk access. A B-Tree of order m can have at most m-1 keys and m children. One of the main reason of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small.

A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

1. Every node in a B-Tree contains at most m children.
2. Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
3. The root nodes must have at least 2 nodes.
4. All leaf nodes must be at the same level.

It is not necessary that, all the nodes contain the same number of children but, each node must have m/2 number of nodes.
A B tree of order 4 is shown in the following image.



While performing some operations on B Tree, any property of B Tree may violate such as number of minimum children a node can have. To maintain the properties of B Tree, the tree may split or join.

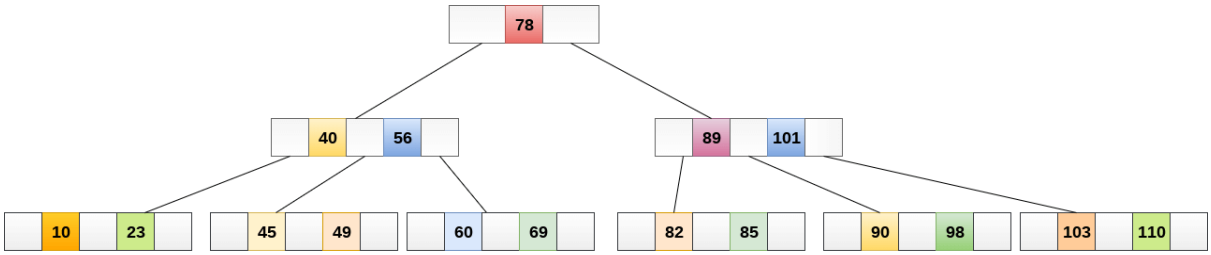
Operations

Searching :

Searching in B Trees is similar to that in Binary search tree. For example, if we search for an item 49 in the following B Tree. The process will something like following :

1. Compare item 49 with root node 78. since 49 < 78 hence, move to its left sub-tree.
2. Since, 40<49<56, traverse right sub-tree of 40.
3. 49>45, move to right. Compare 49.
4. match found, return.

Searching in a B tree depends upon the height of the tree. The search algorithm takes O(log n) time to search any element in a B tree.



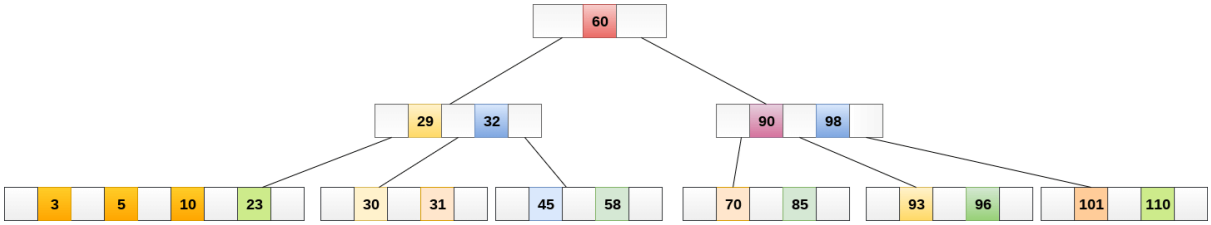
Inserting

Insertions are done at the leaf node level. The following algorithm needs to be followed in order to insert an item into B Tree.

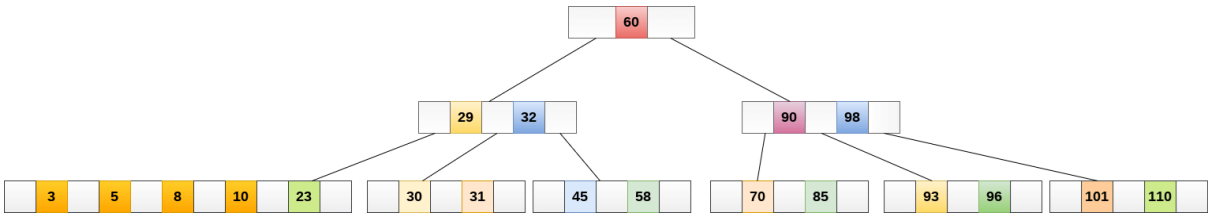
1. Traverse the B Tree in order to find the appropriate leaf node at which the node can be inserted.
2. If the leaf node contain less than m-1 keys then insert the element in the increasing order.
3. Else, if the leaf node contains m-1 keys, then follow the following steps.
    - Insert the new element in the increasing order of elements.
    - Split the node into the two nodes at the median.
    - Push the median element upto its parent node.
    - If the parent node also contain m-1 number of keys, then split it too by following the same steps.

Example:

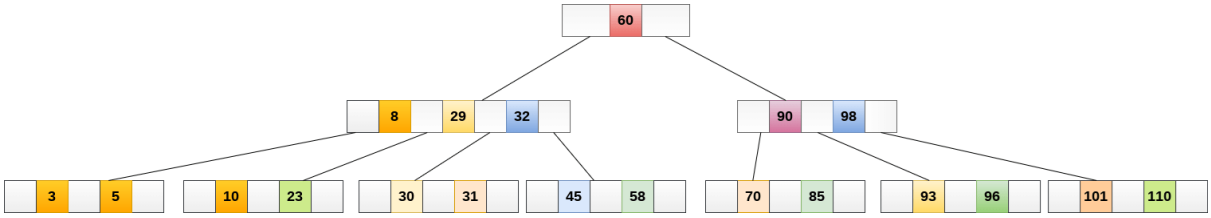
Insert the node 8 into the B Tree of order 5 shown in the following image.



8 will be inserted to the right of 5, therefore insert 8.



The node, now contain 5 keys which is greater than (5 -1 = 4) keys. Therefore split the node from the median i.e. 8 and push it up to its parent node shown as follows.



Deletion

Deletion is also performed at the leaf nodes. The node which is to be deleted can either be a leaf node or an internal node. Following algorithm needs to be followed in order to delete a node from a B tree.

1. Locate the leaf node.
2. If there are more than m/2 keys in the leaf node then delete the desired key from the node.
3. If the leaf node doesn't contain m/2 keys then complete the keys by taking the element from eight or left sibling.
    - If the left sibling contains more than m/2 elements then push its largest element up to its parent and

move the intervening element down to the node where the key is deleted.

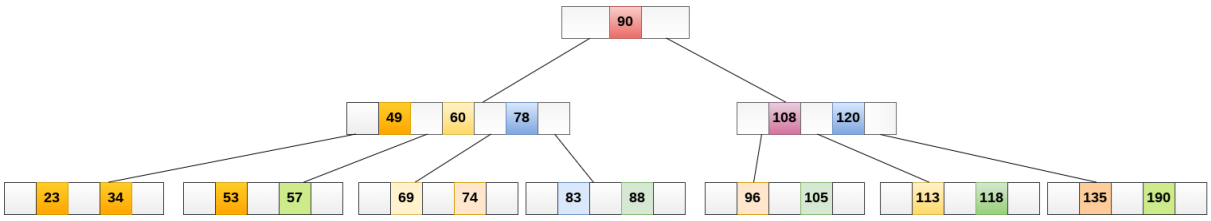
- If the right sibling contains more than m/2 elements then push its smallest element up to the parent and move intervening element down to the node where the key is deleted.

4. If neither of the sibling contain more than m/2 elements then create a new leaf node by joining two leaf nodes and the intervening element of the parent node.
5. If parent is left with less than m/2 nodes then, apply the above process on the parent too.

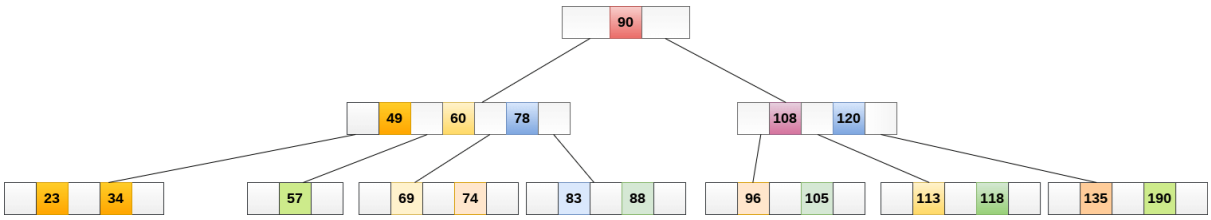
If the the node which is to be deleted is an internal node, then replace the node with its in-order successor or predecessor. Since, successor or predecessor will always be on the leaf node hence, the process will be similar as the node is being deleted from the leaf node.

Example 1

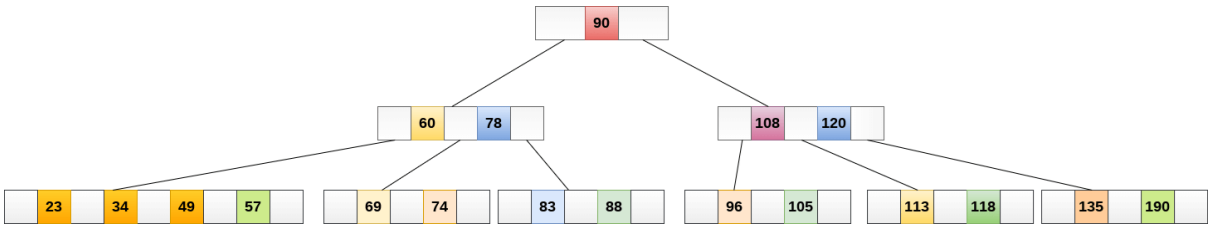
Delete the node 53 from the B Tree of order 5 shown in the following figure.



53 is present in the right child of element 49. Delete it.



Now, 57 is the only element which is left in the node, the minimum number of elements that must be present in a B tree of order 5, is 2. it is less than that, the elements in its left and right sub-tree are also not sufficient therefore, merge it with the left sibling and intervening element of parent i.e. 49.
The final B tree is shown as follows.



Application of B tree

B tree is used to index the data and provides fast access to the actual data stored on the disks since, the access to value stored in a large database that is stored on a disk is a very time consuming process.
Searching an un-indexed and unsorted database containing n key values needs O(n) running time in worst case. However, if we use B Tree to index this database, it will be searched in O(log n) time in worst case.

B+ Tree:

B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.
In B Tree, Keys and records both can be stored in the internal as well as leaf nodes. Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values.
The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.