# B Trees Insertion, Search and Deletion

## B Tree Structure

A single node of the b tree is a structure having an array of keys, a structure of values associated with each key, and it contains the metadata of the node such as number of keys, degree of the node, Boolean indicating if the node is a leaf, array of children of the node storing the indices of its children.

The nodes are stored in a single array; each node referenced using its index.



## B Tree Insertion

* Similar to insertion in binary search trees, we insert new keys in leaf nodes.
* However, we cannot create a new leaf node for the key; instead, we insert the key into an existing leaf node.
* If the leaf node is full (has maximum (2\*t – 1) keys), we split the full node into two nodes having (t-1) keys each and the median key (t) moves into the parent node which has the two child pointers on each side.
* This can lead to a series of splits from a leaf node all the way up to the root, if the leaf node’s parents are also full.
* To avoid this, we do not wait to see if the leaf node needs to be split. As we traverse down the tree, we split each node that is full. Thus, if we need to split a leaf node, we can be sure that its parent is not full.

To implement these steps, we have implemented three functions.

* Insert Function

1. Insert function takes in the key and the root index as the argument.
2. If the node pointed to by root index is NULL, we create a new node with the given key and set it as the new root.  
   Return from the function.
3. If the node pointed to by the root index is full, a new node is created to act as the new root and the current root is set as a child of the new root.  
   Split child function is called with the indices of the two nodes as arguments. This splits the old root into two nodes. The keys of the new root are compared with the key to be inserted to find the appropriate child for insertion and the index of this node is given as an argument for insert\_non\_full function.
4. Else, root index is passed to the insert\_non\_full function.

* Split Child Function

1. This function takes the indices of the parent and the full node as arguments as well as the index of the full child in the children array of the parent node.
2. We create a new node that takes the last (t-1) keys of the full node.
3. If the full node is not a leaf, the new node takes the last t children of the full node.
4. The median key with the index t is moved to the keys array of parent node at the index position with the right child pointer of this key pointing to the new node.

* Insert Non Full Function

1. If the node pointed to by the node index is a leaf, the key is inserted at an appropriate position in the keys array.
2. Else, we find the appropriate child for insertion of the key by comparing the keys in the array.
3. If the child node is full, split child function is called with the node index and the child node’s index as arguments.  
   Since the child node has been split into two, the keys of the node pointed to by the node index are compared again to find the appropriate child node for insertion.
4. Insert non full function is called with the index of this child node.

## B Tree Deletion

* Delete from Leaf Function

1. When the key k to be deleted is present in a leaf node, we delete this key from the keys array and shift all the keys greater than this key by one position to left in the array.

* Delete from Non Leaf Function

1. If the child that precedes k has at least t keys, we use get Predecessor function to get the predecessor of k in the child and replace k with the predecessor calling delete function to remove the predecessor from the child.
2. If the child that precedes k does not have t keys, we see if the child that follows k has at least t keys. If it does, we find the successor of k in this child node, replacing k with its successor. We call the delete function to delete the successor from the child node.
3. If both the children have fewer than t keys, we merge k and the two children together into the preceding child freeing the successor child using merge function.  
   We call the delete function to delete k from this child node.
4. If the key k is not present in this node, we find the index of the child of this node that must contain k.
   * If this child has fewer than t keys, but has an immediate sibling that has at least t keys, we move a key from the current node into the child and move a key from the child’s sibling to the current node (using Borrow from next or Borrow from previous function).
   * If both the siblings of the child node have (t-1) keys, we merge the child node with one of its siblings by moving a key from the current node to the new merged node as the median key.