# B-tree

Revanth.NM
1BM18CS081

```
insert (int key){
    if (root == NULL){                    // If root of tree is
        root = new BtreeNode();           NULL
        root -> keys[0] = key;
        root -> n = 1;
    }
    else{                                 // If root is not NULL check the
        if (root -> n == 2*t -1) {        current node is full Not
            BtreeNode *s = new BtreeNode ( );
            s -> C[0] = root;
            s -> splitchild (0, root);
            int i = 0;
            if (s -> keys[0] < k)
                i++;
            s -> c[i] -> insertNonFull(k);
            root = s;
        }
        else
            root -> insertNonFull(k);     } If node is not full just
    }                                       insert at appropriate position.
}
```

If Node is full
Split the child node and insert in appropiate position.

```
insertNonFull (int k){
    int i = n-1;
    if (leaf == true){
        while (i >= 0 && keys[i] > k){
            keys[i+1] = keys[i];
            i--;
        }
        keys [i+1] = k;
        n = n+1;
    }
```

If current is a leaf node then find exact position for key to insert and insert the key at found position.

Revanth.NM

Revanth.NM
1BM18CS081

```
else
{
    while (i>=0 && Keys[i] >k)
        i--;
    if (C[i+1]->n == 2*t-1)
        SplitChild (i+1, C[i+1]);
        if (Keys[i+1] < k)
            i++;
    }
    C[i+1]-> insertNonFull(k);
}
}
```

If current node is not leaf node then check the child node that is going to have this new key and also checking if its going to a full node or no.

```
SplitChild(int i, BTreeNode *y){
    BTreeNode *z = new BTreeNode(); } create a new node
    z->n = t-1;
    for (int j=0 ; j<t-1 ; j++)
        z->keys[j] = y->Keys[j+t];
```

} ~~insert~~ copy the t-1 node from y to new node z

```
    if (y->leaf == false){
        for (int j=0 ; j<t ; j++)
            z->C[j] = y->C[j+t];
    }
```

} If y is not a leaf node copy t node from y to z.

```
    y->n = t-1;
    for(int j=n ; j>=i+1 ; j--)
        C[j+1] = C[j];
```

} creating a space for new child.

```
    C[i+1] = z;
    for (int j = n-1 ; j >= i ; j--)
        Keys[j+1] = Keys[j];
    Keys[i] = y->Keys[t-1];
    n = n+1;
}
```

} copy the middle key of y and increament count by 1.