B-Tree

1BM18CS002
Abhijnya

Insertion

- traverse()

```
for i 0 to n
    if (leaf == false)
        c[i] → traverse()
    cout << keys[i]
if (leaf == false)
    c[i] → traverse()
```

- Insert(k)

```
if (root == NULL)
{
    root = new Btree(d, true)
    root → keys[0] = k
    root → n = 1
}
else
        // If root is full then tree ↑ in height
    if (root → n == 2*d - 1)
    // Making old root as child of new root
    s → c[0] = root
    // Split old root by calling the function
    s → splitchild(0, root)
```

```
// Deciding which node shld have new key
if (s -> keys[0] < k)
    p++
s -> c[i] -> insertNonfull(k)
root = s
}
else
    // If root is not full then call function not full
        root -> insertNonfull(k)
```

- InsertNonfull(k)

```
{   if (leaf == true)
    {   while (i >= 0 && keys[i] > k)
        {
            keys[i+1] = keys[i];
            i--;
        }
    keys[i+1] = k
    }
    else
    {   // If node is not leaf
        // Find child which is going to have new element
        // check if chosen node is full
        if (c[i+1] -> n == 2*d - 1)
        {
            splitchild(i+1, c[i+1])
        }
        c[i+1] -> insertnonfull(k);
}   }
```

```
childsplit ( i, y )
{       z->n = d - 1
        for j 0 to d - 1
            z->keys[j] = y->keys[j + d]

        if (y->leaf == false)
        {
            for j 0 to d
                z->c[j] = y->c[j + d]
        }
        y->n = d - 1 ; // Reduce no of keys in y
        for j n-1 to j >= i
            keys[j+1] = keys[j]

        keys[i] = y->keys[d - 1]
        n = n + 1
}
```