# 2-3 Tree

```
class twothree
{   node *root = NULL;
    public: void traversal()
            { if(root != NULL)
                    root -> traversal();
            }
            void insertion(int item);
            void deletion(int item);
};
void twothree::insertion(int item)
{   if(root == NULL)
    {   root = new node(true);
        root -> data[0] = item;
        root -> n = 1;
    }
    else
    {   if(root -> n == 3)
        {   node *s = new node(false)
            s -> child[0] = root;
            s -> split(0, root);
            int i = 0;
            if(s -> data[0] < item)
                i++;
            s -> child[i] -> insertion non full(item);
            root = s;
        }
        else
            root -> insertion non full(item);
    }
}
```

```
void node :: insertionnonfull (int item)
{
    int i = n-1;
    if (leaf == true)
    {
        while (i>=0 && data[i] > item)
        {
            data[i+1] = data[i];
            i--;
        }
        data[i+1] = item;
        n = n+1;
    }
    else
    {
        while (i>=0 && data[i] > item)
        {
            i--;
            if (child[i+1] -> n == 3)
            {
                split (i+1, child[i+1]);
                if (data[i+1] < item)
                    i++;
            }
            child[i+1] -> insertionnonfull (item);
        }
    }
}
void node :: deletion (int item)
{
    if (itemx < n && data[itemx] == item)
    {
        if (leaf)
            deletionleaf (itemx);
        else
            deletionnonleaf (itemx);
    }
    else {
        if (leaf)
        {
            cout << "Item does not exist in";
            return;
        }
```

```
    bool flag = (item x == n) ? true : false;
    if (child[item x] -> n < 2)
            fill (item x);
    if (flag && item x > n)
            child[item x - 1] -> deletion (item);
    else
            child[item x] -> deletion (item);
  }
  return;
}

void node:: deletionleaf (int item x)
{   for (int i = item x + 1; i < n; i++)
            data[i-1] = data[i];
    n--;
    return;
  }

void node:: deletionnonleaf (int item x)
{   int item = data[item x];
    if (child[item x] -> n >= 2)
    { int pred = predecessor (item x);
      data[item x] = pred;
      child[item x] -> deletion(pred);
    }
    else if (child[item x + 1] -> n >= 2)
    {   int succ = successor (item x);
        data[item x] = succ;
        child[item x + 1] -> deletion (succ);
    }
    else
    {   merge (item x);
        child[item x] -> deletion (item);
    }
    return;
}
```

```
void btwothree::deletion(int item)
{   if(!root)
    {  cout<<"The tree is empty\n";
       return;
    }
    root->deletion(item);
    if(root->n==0)
    {      node *temp=root;
           if(root->leaf)
                root=NULL;
           else
                root=root->child[0];
           delete temp;
    }
    return;
}

void node::split(int i, node *y)
{  node *z=new node(y->leaf);
   z->n=1;
   z->data[0]=y->data[2];
   if(y->leaf==false)
   {     for(int j=0;j<2;j++)
              z->child[j]=y->child[j+2];
   }
   y->n=1;
   for(int j=n;j>=j+1;j--)
       child[j+1]=child[j];
   child[i+1]=z;
   for(int j=n-1;j>=i;j--)
        data[j+1]=data[j];
   data[i]=y->data[1];
   n=n+1;
}
```

```
int node::search(int item)
{   int itemx=0;
    while(itemx<n && data[itemx]<k)
        ++itemx;
    return itemx;
}
```