```
class TreeNode
{
    int *keys
    TreeNode **child;
    int n
    bool leaf;
    friend class Tree;
};
class Tree
{
    Tree Node *root = NULL;
    public:
        class traverse (){
            if (root! =NULL)
                root->traverse();
        }
        void insert (int k);
        void remove (int k);
};
void Tree : insert (int k)
{
    if (root == NULL)
    {
        root = new Tree Node (true);
        root->key[0] = k;
        root->n = 1;
    }
```

```
else
    if(root -> n == 3)
    {
        TreeNode *s = new TreeNode(false);
        s -> child[0] = root;
        s -> splitchild(0, root);
        int i = 0;
        if (s -> keys[0] < k)
            i++;
        s -> child[i] -> insertNonFull(k)
        root = s;
    }
    else
        root -> insertNonFull(k);
}
void TreeNode :: 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;
    }
```

```
else {
  while (i >= 0 && keys[i] > k)
      i--;
  if (child[i+1] -> n == 3)
  {
      splitchild (i+1, child[i+1]);
      if ( keys[i+1] < k)
          i++;
  }
  child[i+1] -> insertNonFull (k);
  }
}
void TreeNode :: splitchild (int i; TreeNode y)
{
  TreeNode z = new TreeNode (y->leaf);
  z -> n = 1;
  z -> keys[0] = y->key[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 >= i+1 ; j--)
      child[j+1] = child[j];
  child[i+1] = z;
  for (int j =n-1; j >= i; j--)
      key[j+1] = key[j];
  keys[i] = y-> keys[1];
```

Karthik S. Anand
IBM18CS046


```
    n = n + 1;
}
void TreeNode :: remove (int k)
{
    int idx = findKey (k)
    if (idx < n && keys [idx] == k)
    {
        if (leaf)
            removeFromLeaf (idx);
        else
            removeFromNonleaf (idx)
    }
    else {
        if (leaf)
        {   cout << "key doesn't exist \n";
            return;
        }
        bool flag = ((idx == n)? true : false);
        if (child [idx] -> n < 2)
            fill (idx);
        if (flag && idx > n)
            child [idx - 1] -> remove (k);
        else
            child [idx] -> remove (k);
    }
    return;
}
void TreeNode :: RemoveFromLeaf (int idx)
{   for (int i = idx + 1; i < n; ++i)
        keys [i - 1] = keys [i];
    n--;
```

Krishik S. Anand
IBM18CS046

```
    return;
}
void TreeNode :: removeFromLeaf(int idx)
{
    for (int i=idx+1; i<n; ++i) int k=keys[idx];
        keys[i-1]  if(child[idx]->n>=2)
    {   int pred = getPred(idx);
        key[idx]=pred
        child[idx]->remove(pred);
    }
    else if(child[idx+1]->n>=2)
    {   int succ =getsucc(idx);
        keys[idx]=succ;
        child[idx+1]->remove(pred);
    }
    else if(child[idx+]->n>=2)
    {   int succ=getsucc(idx);
        keys[idx]=succ;
        child[idx]->remove(succ)
    }
    else{
        merge(idx);
        child[idx]->remove(x);
    }
    return;
}
```

Kshitish S. Anand
IBM18CS015

```
Void Tree :: remove (int k)
{
    if (!root)
    {
        cout << "Tree is empty \n";
        return;
    }
    root -> remove (k);
    if ( root -> n == 0){
        TreeNode *tmp = root
        if ( root -> leaf ) root = NULL;
        else root = root -> child [0];
        delete tmp;
    }
    return;
}
```