Prg 5- 2-3 tree insertion & deletion

```
class TreeNode
{
    int keys;
    TreeNode **child;
    int n;
    bool leaf;
};

class Tree
{
    TreeNode *root = NULL;
public:
    void traverse() {
        if (root != NULL) root->traverse();
    }
    void insert(int k);
    void remove(int k);
};

void Tree::insert(int k)
{
    if (root == NULL)
    {
        root = new TreeNode(true);
        root->keys[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;
```

Scanned by TapScanner

```
if (s->keys[b] < 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);
}
```

Scanned by TapScanner

```
void TreeNode::splitchild(int i, TreeNode *y)
{
    TreeNode *z = new TreeNode(y->leaf);
    z->n = i;
    z->keys[0] = y->keys[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--)
        keys[j+1] = keys[j];
    key[i] = y->keys[i];
    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 {
```

classmate
Date
Page

Scanned by TapScanner

```
if (leaf)
{
    cout<<"key doesnt exist"<<endl;
    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; i<n; ++i)
        keys[i-1] = keys[i];
    n--;
    return;
}
void TreeNode::removeFromNonLeaf (int idx)
{
int k = keys[idx];
if (child[idx]->n>=2)
{
    int pred = getPred(idx);
    keys[idx] = pred;
child[idx]->remove(pred); }
```

Scanned by TapScanner

```
else if (child[idx+1]->n>=2)
{
    int succ = getsucc(idx);
    keys[idx] = succ;
    child[idx+1]->remove(succ);
}
else
{
    merge(idx);
    child[idx]->remove(k);
}
return;
}
void tree::remove(int k)
{
if (!root)
{
cout<<"Tree is empty"<<endl;
    return;
}
root->remove(k);
if (root->n==0)
{
    TreeNode *tmp = root;
if (root->leaf)
    root=NULL;
else
    root=root->child[0];
delete tmp;
}
return;
```

Scanned by TapScanner