Chandan A. M.
1BM18CS025

(5) Two-Three Trees :

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

Chandan A M
025

```
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++;
    }
    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);
    }
}
```

Chandan A.M
025

```
void Treenode :: splitchild (int i, Treenode *y)
{
    Treenode *z = new Treenode (y->leaf);
    z->n = 1
    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 [i];
    keys [i] = y->keys [i];
    n++;
}
void Treenode :: remove (int k) {
    int x = findkey (k);
    if (x < n && keys[x] == k) {
        if (leaf)
            removeFromleaf (x);
        else
            removeFromNonleaf (x);
    }
```

```
else {
    if (leaf)
        cout << "The key doesnot exist";
    bool flag = (x == n) ? true : false;
    if child [x] -> n < 2)
        fill (x);
    if (flag && x > n)
        child [x-1] -> remove (k);
    else
        child [x] -> remove (k);
}
return;
}

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

```
        else {
            merge(x);
            child[x] -> remove(k);
        }
        return;
    }
void Tree::remove(int k){
    if(root) {
        cout<<"Empty Tree";
        return
    }
    root->remove(k);
    if(root->n==0) {
        Treenode *temp = root;
        if(root->leaf)
            root = NULL;
        else {
            root = root->child(0);
            delete tmp;
        }}
        return;
}
```