2-3 Tree

Ajay Mittur
1BM18CS006

Insert

```
class TwoThreeTree {
    TwoThreeNode* root;
    int t; // degree
    TwoThreeTree() {
        root=NULL;
        t=2;
    }
};
```

```
void insert(int k) {
    if (!root) {
        root = new TwoThreeNode(k);
        root->keys[0] = k;
        root->n = 1;
    } else if (root->n == 2*t-1) {
        TwoThreeNode* s = new TwoThreeNode();
        s->C[0] = root;
        s->split(0, root);
        int i=0; if(s->keys[0]<k) i++;
        s->C[i]->insertintoNode(k);
        root=s;
    } else
        root->insertintoNode(k);
}
```

Delete

```
void remove(int k) {
    if (!root) return;
    root->remove(k);
    if (root->n == 0) {
        TwoThreeNode *tmp=root
        if (root->leaf) root=NULL;
        else root = root->C[0];
        delete tmp;
    }
    return;
}
```

Date ....

Expt. No. .... 2-3 Trees Page No. ....

Ajay Mittur
1BM18CS006

```
void insertintoNode (int k) {
 int i = n-1;
 if (leaf) {
     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 (C[i+1]->n == 2*t-1)
             split (i+1, C[i+1]);
      if (keys[i+1]<k) i++;
     }
     C[i+1] -> insertintoNode(k);
  }
}
void  split ( int i, TwoThreeNode* y) {
 TwoThreeNode *z = new TwoThreeNode(y->leaf);
   y->n = t-1;      ⟶  for (int j=0; j<t-1; j++ )
                           z->keys[j] = y->keys
                                          [j+t]
   for( int j=n-1; j>=i+1; j--)
        C[j+1] = C[j];
   C[i+1]=z;
   for( int j=n-1; j>=i; j--)
    keys[j+1] = keys[j];
    keys[i] = y->keys[t-1];
     n= n+1;
 }
```

Teacher's Signature : ______________

2-3 Trees

Ajay Mittur
~~1BM18CS006~~ [illegible]

```
void TwoThreeNode::remove(int k) {
    int idx = findKey(k);
    if (idx < n && keys[idx] == k) {
        if (leaf) removeLeaf(idx);
        else removeNonLeaf(idx);
    } else {
        if (leaf) ~~&~~ return;
        bool flag = ((idx == n)? true : false);
        if ((C[idx]->n < t) fill(idx);
        if (flag && idx > n) (C[idx-1]->remove(k);
        else (C[idx]->remove(k);
    }
    return;
}
void removeLeaf(int idx) {
    for (int i = idx+1; i<n; i++) keys[i-1] = keys[i];
    n--;
    return;
}
void removeNonLeaf(int idx) {
    int k = keys[idx]s, n;
    if ((C[idx]->n >= t) {
        ~~int pred~~ n = get Pred(idx)
    else n = get Successor(idx)
    keys[idx] = n;
    C[idx] ~~o~~ -> remove(pred);
    else {
        merge(idx);
        C[idx]->remove(k);
    }
}
```