2-3 tree

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

Void insert(int k) {
    if (!root) {
        root = new TwoThreeNode();
        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 -> insert
        root = insertIntoNode(k);
    }
}
```

```
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])
        else if (keys[i+1] < k)
            i++;
    }
    c[i+1] -> insertIntoNode(k);
}
}

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

```
    y->n = t-1;
    for (int j = n ; 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;
}

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

```
void removeFromLeaf (int idx){
  for (int i= idx+1 ; i<n; ++i)
    keys[i-1]= keys[i];
  n--;
  return;
}

void removeFromNonLeaf (int idx){
  int k= keys[idx];
  if (C[idx]->n >=t) {
    int pred= getPred (idx);
    keys[idx] = pred;
    C[idx] =>remove (pred);
  } else if (C[idx+1]->n >=t) {
    int succ= getSucc (idx);
    keys[idx]= succ;
    c[idx+1]-> remove (succ);
  } else {
    merge[idx];
    C[idx] =>remove(k);
  }
  return;
}
```