Lab5 write up

```
#include <bits/stdc++.h>
using namespace std;

class TwoThreeNode
{
    int *keys;
    int t;
    TwoThreeNode **C;
    int n;
    bool leaf;

public:
    TwoThreeNode(bool leaf);
    void traverse();
    TwoThreeNode *search(int k);
    int findKey(int k);
    void insertNonFull(int k);
    void splitChild(int i, TwoThreeNode *y);
    void remove(int k);
    void removeFromLeaf(int idx);
    void removeFromNonLeaf(int idx);
    int getPred(int idx);
    int getSucc(int idx);
    void fill(int idx);
    void borrowFromPrev(int idx);
    void borrowFromNext(int idx);
    void merge(int idx);
    friend class TwoThreeTree;
};
```

```
TwoThreeNode *root;
int t;
public:
    TwoThreeTree()
    {
        root = NULL;
        t = 2;
    }
    void traverse()
    {
        if (root != NULL)
            root->traverse();
    }
    TwoThreeNode *search(int k)
    {
        return (root == NULL)? NULL:
            root->search(k);
    }
    void insert(int k);
    void remove(int k);
};

void TwoThreeNode::remove(int k)
{
    int idx = findKey(k);
    if (idx < n && keys[idx] == k)
    {
        if (leaf)
            removeFromLeaf(idx);
        else
            removeFromNonLeaf(idx);
    }
    else
    {
        if (leaf)
        {
            cout << "Not exists" <<
                endl;
            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 TwoThreeNode::removeFromLeaf(int idx)
{
    for (int i=idx+1; i<n; ++i)
        keys[i-1] = keys[i];
    n--;
    return;
}

void TwoThreeNode::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;
}
```

```
void TwoThreeNode:: insert(int k)
{   if(root==NULL)
    {
        root = new TwoThreeNode(true);
        root->keys[0]=k;
        root->n=1;
    }
else
{   if(root->n==2*t-1)
    {  TwoThreeNode *s =
       new TwoThreeNode(false);
        s->C[0]=root;
        s->splitChild(0,root);
        int i=0;
        if(s->keys[0]<k)
            i++;
        s->C[i]->insertNotFull(k);
        root=s;
    }
    else
        root->insertNonfull(k);
}
}
void TwoThreeNode::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(C[i+1]->n==2*t-1)
    {  splitChild(i+1,C[i+1]);
       if(keys[i+1]<k)i++;
    }
    C[i+1]->insertNonFull(k);
}
}
void TwoThreeNode::splitChild
   (int i,TwoThreeNode* y)
{
TwoThreeNode* z= new
   TwoThreeNode(y->leaf);
z->n=t-1;
for(int j=0;j<t-1;j++)
    z->keys[j]=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--)
    key[j+1]=keys[j];
keys[i]=y->keys[t-1];
n++;
}
```