Prog-5: 2-3 trees: Insertion and deletion

Shikha N
1BM18CS149

```
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(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;
    }
```


Shikha

```
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] -> insert NoaFull (k);
        root = S;

  }

  else
    { root -> insert NonFull (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 && key[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);
}
}
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[j];
    keys[i] = y -> keys[i];
    n++;
}
```


Shikha

```
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 doesn't exist\n";
         return;
      }
      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 Treenode :: RemoveFromLeaf (int x)
{ for (int i = x+1 ; i < n ; ++i)
        keys(i-1) = keys(i)

    n--;
    return;

}
void Treenode :: 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 << "The tree is empty\n";
        return;
    }
    root->remove(k);
    if(root->n==0)
    {  Treenode *tmp=root;
       if(root->leaf)
           root=NULL;
       else
       {root=root->child[0];
        delete tmp;
       }
    return;

}.
```


Sluhs