Hemanth Kumar V.C
IBM18CS037

Q-3 tree insertion

```
class Treenode
{
    int *keys;
    Treenode **child;
    int n;
    bool leaf;
};
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;
    }
```

```
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 = n+1;
    }
```

```
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);
  }
}
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 = n+1;
    }
void TreeNode::remove(int k)
{
    int idx = findKey(k);
    if (idx < n && keys[idx] == k)
    {
        if (leaf)
            removeFromLeaf(idx);
        else
            removeFromNonLeaf(idx);
    }
    else
    {
        if (leaf)
        {
            cout << "key does not exist" << endl;
            return;
        }
        bool flag = ((idx == n)? true : false);
        if (child[idx]->n < t)
            fill(idx);
        if (flag && idx > n)
            child[idx-1]->remove(k);
        else child[idx]->remove(k);
    }
    return;
}
```

```
void Treenode :: remove from leaf (int idx)
{
    for (int i = idx+1; i<n; i++)
        keys[i-1] = keys[i];
    n--;
    return;
}

void Tree node :: remove from non leaf (int idx)
{
    int k = keys [idx];
    if (child [idx] ->n >= 2)
    {
        int pred = get pred (idx);
        keys (idx) = pred;
        child [idx] -> remove (pred);
    }
    else if (child[idx+1] ->n >=2)
    {
        int succ = getsucc (idx);
        keys[idx] = succ;
        child [idx+1] -> remove (succ);
    }
    else
    {
        merge (idx);
        child [idx] -> remove (k);
    }
    return;
}
```

```
void Tree::remove(int k)
{
    if(!root)
    {
      cout<<"Tree is empty"<<endl;
          return;
    }
    root->remove(k);
    if(root->n==0)
    {
        TreeNode tmp=root
         if(root->leaf)   root=NULL;

        else
              root=root->child[0];
    }          delete tmp;
    return;
}
```