Lab 5

Prathibha. R
1BM18CS074

```
class Tree.
{
  TreeNode *root = NULL;
public:
  void Traverse()
    if (root != NULL)
      root -> traverse.
    void insert(int k);
    void remove(int k);
  }
class TreeNode
{ int keys;
  TreeNode * *child;
  int n;
  bool leaf;
};
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 :: insert (Non full (int n))
{ int i=n-1;
  if (leaf == true)
  { while (i>=0 && keys[i]>k)
    { key [i+1] = key [i];
      i--;
    }
    key[i+1] = k;
    n = n+1;
  }
  else
  while (i>=0 && keys[i]>k)
    i--;
  if (child [i++] -> n == 3)
  {
    splitchild (i+1, child [i+1]);
    if (keys[i+1] < k)
      i++;
  }
  child [i+1] -> insertNonfull (n); }}
void TreeNode :: splitchild (int i, Tree Node * 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+j] = z;
  for (int j=n-1; j>=i; j--)
    keys [j+1] = keys [j]
  key [i] = y -> keys [i]
  n = n+1;
```

```
void TreeNode :: remove(int k)
{ int idx = findKey(k)
  if (idx < n && key[idx] == k)
  { if (leaf)
      removeFromLeaf(idx);
    else
      removeFromNonLeaf(idx);
  }
  else
  {
  if (leaf)
  {
    cout << "by doesn't exist" << endl;
    return;
  }
  bool flag = ((idx == n) ? true : false);
  if (child[idx] -> n < 2)
      fill(idx);
  if (flag && idx > n)
    child[idx-1] -> remove(k);
  else
      child[idx] -> remove(k);
  }
} return;

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

void TreeNode :: removeFromNonLeaf (int idx)
{ int k = keys[idx];
```

```
if (child[idx]->n>=2)
{ int pred = getpred(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 *temp = root;
    if (root->leaf)
      root = NULL;
    else
      root = root->child[0];
    delete temp;
  }
  return;
}
```