chandan c Bajan
1BM18CS026

Lab-6

program 5 - 2-3 trees insertion deletion.

```
class TreeNode
{
  int * keys;
  TreeNode ** child;
   int n;
   bool leaf;
   friend class Tree;
};                    // function declaration
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(tree);
      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 [1];
   n = n+1;
}
 void Tree Node :: remove (int k)
{
   int idx = find_key(k)     //returns index of the first key
                              // greater than or equal to k

  if (idx <n && keys [idx] ==k)
  {
        if ( leaf) remove From leaf (idx);
        else   remove From Nonleaf (idx);
  }
  else{ if (leaf)
       {
          cout << " keys doesn't exist <<endl;
          return;
       }
    bool flag = ((idx==n) ? true : false);
    if (child [idx] → n<2)
          fill (idx); // fills child[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+1 ; i<n ; ++i )
        keys [i-1] = keys [i];
   n--;
   return;
}
void TreeNode :: removeFromNonLeaf ( int idx)
{
   int k = keys [idx];
   if (child [idx] → n >=2)
   {  int pred = getpred (idx); // gets predecessor of keys[id
      child [idx] → remove (pred);
      keys [idx] = pred;
   }
   else if (child [idx+1] → n >=2)
   {
     int succ = getsucc (idx); // gets successor of keys[idx]
     keys [idx] = succ;
     child [idx+1] → remove (succ);
   }
else{
     merge(idx);  merge child[idx] with child[idx+1]
   child [idx] → remove(k) // child [idx+1] is freed
 return;                                after merging
}
```

```
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 ;
}
```