Snehita . I
1BM18CS109

```
class TreeNode
{  int *keys;
   TreeNode **child;
   int n,
   bool leaf;
   friend class Tree;
};
class Tree
{
   TreeNode * root = NULL;
   public:
         void traverse()
         { if (!root) root -> traverse();
         }
         void insert (int k);
         void remove (int k);
};
void Tree :: insert (int k)
{
      if (!root)
      {  root -> new TreeNode(true)
         root -> keys[0] = k;
         root -> n = 1;
      }
      else {   if ( root -> n == ))
               {   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 NonFull(k);
                   root = s;
               }
            else      root 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 = 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 (j=0; j<2; j++)
                      z->child[j] = y->child[j+2];
         }
        y->n = 1;
       keys[i] = y->keys[1];
       n=n+1;
}
void TreeNode :: removeFromLeaf(int in)
{    for(int i=in+1; i<n; i++)
              keys[i-1] = keys[i];
     n--;
     return;
 }
```

```
void TreeNode:: removeFromNonLeaf (int in)
{
        int  k = keys[in];
        if ( child[in] → n >= z)
        {
                int pred = getPred (in);
                keys [in] = pred .
                child [in] → remove(pred);
        }.
        else if ( child[in+1] → n >= z)
        {
                int  succ = getSucc(in);
                keys[in] = succ;
                child [in+1] → remove (succ);
        }
        else {   merge (in);
                 child [in] → remove(k);
        }
        return;
}.
```