NAME: NIHA
USN: 1BM18CS060
21st NOV '2020

papergrid
Date: / /

# LAB-5

⇒ 2-3 tree :- Insertion and deletion:-

```
class TreeNode
{
    int *keys;
    TreeNode **child;
    int n;
    bool leaf;
     // functions 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->key[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->key[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] = key[i];
            i--;
        }
        key[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 key)
{
    TreeNode *z = new TreeNode(y -> leaf)
    z -> n = 1;
    z -> keys[0] = y -> keys[2];
    if (y -> leaf == false)
    {
        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;
    }
```

Date: / /

```
void TreeNode :: remove (int k)
{
  int idx = findKey(k)  // returns index of the
                        // first key greater
                        // than or equal to k
  if ( idx < n && keys[idx] == k)
  {
      if( leaf)
          removeFromLeaf( idx) ;
      else
          removeFromNonLeaf( idx) ;
  }
  else {
      if (leaf)
      {
          cout << "key 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[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);   // merges child[idx] with child[idx+1]
    child[idx]->remove(k);  // child[idx+1] is freed after merging
    }
    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;
}
```