Sumanth.kv.
1BM18CS112
5 C
ADS

2-3 trees

```
struct btreeNode {
        int val[MAX+1], count;
        btreeNode * link[MAX+1];
};

btreeNode  *root = NULL;

btreeNode * createNode (int val, btreeNode *child)
{       btreeNode  *newNode = new btreeNode;
        newNode -> val[1] = val;
        newNode -> count = 1;
        newNode -> link[0] = root;
        newNode -> link[1] = child;
        return newNode;
}
```

```
void AddValToNode (int val, int pos, btreeNode
                   * node, btreeNode *child) {

    int j = node -> cont;
    while (j > pos) {
        node -> val [j+1] = node -> val [j];
        node -> val [j+1] = node -> link [j];
        j--;
    }
    node -> val [j+1] = val;
    node -> link (j+1) = child;
    node -> cont++;
}

Void SplitNode (int val, int *pval, int pos,
btreeNode *node, btreeNode *child, btreeNode
 **newNode) {
    int median, j;
      if (pos > MIN)
          median = MIN + 1;
    else
          median = MIN;
    *newNode = new btreeNode;
    j = median + 1;
    while (j <= MAX) {
    (*newNode) -> val [j-median] = node -> val [j];

    (*newNode) -> link (j-median) = node -> link [j];

      j++;

    }
    node -> cont = median;
    (*newNode) -> cont = MAX - median;
```

```
        if (pos <= MIN) {
            addValToNode (val, pos, node, child);
        }
    else {
        addValToNode(val, pos-median, *newNode,
                                        child);
    }
    *pval = node->val[node->count];
    (*newNode)->link[0] = node->link[node->count];
    node->count--;
}

int SetValueInNode (int val, int *pval,
        btreeNode *node, btreeNode **child) {
    int pos;
    if (!node) {
        *pval = val;
        *child = NULL;
        return 1;
    }
    if (val < node->val[1])
        pos = 0
    else
        for (pos = node->count;
            val < node->val[pos] && pos > 1; pos--)
        if (val == node->val[pos])
            cout << "Duplicate not allowed";
        return 0;

    if (setValueInNode (val, pval, node->link[pos],
        child)) {
        if (node->count < MAX) {
            addValToNode (*pval, pos, node, *child
    } else
```

```
        {
            splitNode (*pval, pval, pos, node, *child,
                       child);
            return 1;
        }
    }
    return 0;
}

void insertion (int val) {
    int flag, i;
    btreeNode *child;
    flag = setValueInNode (val, &i, root, &child);
    if (flag)
        root = createNode (i, child);
}

void copySuccessor (btreeNode *myNode, int pos){
    btreeNode *dummy;
    dummy = myNode->link[pos];

    for( ; dummy->link[0] != NULL;)
        dummy = dummy->links[0];
    myNode->val[pos] = dummy->val[1];
}

void removeVal (btreeNode *myNode, int pos)
{
    int i = pos+1;
    while (i <= myNode->count) {
        myNode->val[i-1] = myNode->val[i];
        myNode->link[i-1] = myNode->links[i];
        i++;
    }
    myNode->count--;
}
```

```
void doRightShift (btreeNode *myNode, int pos)
{
    btreeNode *x = myNode->link[pos];
    int j = x->count;
    while (j>0) {
        x->val[j+1] = x->val[j];
        x->link[j+1] = x->link[j];
    }
    x->val[1] = myNode->val[pos];
    x->link[1] = x->link[0];
    x->count++;

    x = myNode->link[pos-1];
    myNode->val[pos] = x->val[x->count];
    myNode->link[pos] = x->link[x->count];
    x->count--;
    return;
}

void doleftShift (btreeNode *myNode, int pos) {
    int j=1;
    btreeNode *x = myNode->link[pos-1];

    x->count++;
    x->val[x->count] = myNode->val[pos];
    x->link[x->count] = myNode->link[pos]->link[0];

    x = myNode->link[pos];
    myNode->val[pos] = x->val[1];
    x->link[0] = x->link[1];
    x->count--;
    while (j<=x->count) {
        x->val[j] = x->val[j+1];
        x->link[j] = x->link[j+1];
        j++;
    }
}
```

Page No.
Date

```
    return j
}

Void mergeNodes (btreeNode *myNode, int pos) {
    int j=1;
    btreeNode *x1 = myNode->link[pos], *x2 =
    myNode->link[pos-1];
    x2->count++;
    x2->val[x2->count] = myNode->val[pos];
    x2->link[x2->count] = myNode->link[0];

    while (j<=x1->count) {
        x2->count++;
        x2->val[x2->count] = x1->val[j];
        x2->link[x2->count] = x1->link[j];
        j++;
    }
    j = pos;
    while (j<myNode->count) {
        myNode->val[j] = myNode->val[j+1];
        myNode->link[j] = myNode->link[j+1];
        j++;
    }
    myNode->count--;
    free(x1);
}

void adjustNode (btreeNode *myNode, int pos) {
    if (
```

```
int delValFromeNode(int val, btreeNode, *myNode) {
    int pos, flag=0;
    if(myNode) {
        if(val < myNode->val[1]) {
            pos=0;
            flag=0;
        }
        else {
            for(pos=myNode->count; val<myNode->
                val[pos] && pos>1); pos--);
            if (val == myNode->val[pos]) {
                flag=1;
            }
            else {
                flag=0;
            }
        }
        if (flag) {
            if (myNode->link[pos-1]) {
                copySuccessor(myNode, pos);
            flag= delValFromNode(myNode->val[pos],
                myNode->link[pos]);
                if(flag==0) {
                    cout << "Not present";
                }
            }
        else {
            removeVal(myNode, pos);
            }
        }
        else {
            flag= delValFromNode(val, myNode->link[pos]);
        }
```

```
        if (myNode->link[pos]) {
            if (myNode->link[pos]->count < MIN)
                adjustNode (myNode, pos);
        }
    }
    return flag;
}

void deletion (int val, btreeNode *myNode) {
    btreeNode *tmp;
    if (!delValFromNode (val, myNode)) {
        return;
    }
    else {
        if (myNode->count == 0) {
            tmp = myNode;
            myNode = myNode->link[0];
            free (tmp);
        }
    }
    root = myNode;
    return;
}
```

(8) smanth.c