08/10/2020
Wednesday.

ADS

Aryan N.S
1BM18CS019

Lab-5 Writeup

2-3 trees.

```
class node
{       int *data;
        node **child;
        int n;
        bool leaf;
public: node (bool leaf)
        void traversal();
        int search (int item);
        void Insertiononfull (int item);
        void split (int , node *);
        Void deletion (int item);
        void deletionleaf (int itemx);
        void deletiononleaf (int itemx);
        int predecessor (int itemx);
        int successor (int itemx);
        Void fill (int itemx);
        Void borrownext (int itemx);
        Void borrowprev (int itemx);
        Void merge (int itemx);
};
class twothree
{       node *root = NULL;
        public: void traversal()
                {
                        if (root != NULL)
                                root -> traversal();
                }
                Void insertion (int item);
                Void deletion (int item);
};
```


Arya. N.S

Arjun A.S
1BM18CS019

```
void twothree :: insertion (int item)
{
    if (root == NULL)
    {
        root = new Node (true);
        root -> data[0] = item;
        root -> n = 1;
    }
    else
    {
        if (root -> n == 3)
        {
            node *s = new node (false);
            s -> child [0] = root;
            s -> splitc (0, root);
            int i=0;
            if (s -> data[0] < item)
                i++;
            s -> child [i] -> insertiononfull (item);
            root = s;
        }
        else
            root -> insertiononfull (item);
    }
}
void node :: insertiononfull (int item)
{
    int i = n-1;
    if (leaf == true)
    {
        while (i>=0 && data [i] > item)
        {
            data [i+1] = data[i];
            i--;
        }
        data[i+1] = item;
        n = n+1;
    }
    else
    {
        while (i>=0 && data[i] > item)
        {   i--;
            if (child[i+1] -> n == 3)
            {   splitc (i+1, child[i+1]);
                if (data [i+1] < item)
                    i++;
            }
        } child [i+1] -> insertiononfull (item);
    }
}
```

```
void node::deletion(int item)
{
    int itemx = search(item);
    if (itemx<n && data[itemx]==item)
    {   if (leaf)
        {   deletionleaf(itemx);
        else
            deletiononleaf(itemx);
    }
    else
    {
        if (leaf)
        {   cout << "Item does not exists.";
            return;
        }
        bool flag = (itemx==n)? true : false;
        if (child[itemx]->n < 2)
            fill(itemx)
        if (flag && itemx>n)
            child[itemx-1]->deletion(item);
        else
            child[itemx]->deletion(item);
    }
    return;
}
void node::deletionleaf(int itemx)
{   for(int i = itemx+1; i<n; i++)
        data[i-1] = data[i];
    n--;
    return;
}
void node::deletiononleaf(int itemx)
{   int item = data[itemx];
    if(child[itemx]->n >=2)
    {   int pred= predecessor(itemx);
        data[itemx] = pred;
        child[itemx]->deletion(pred);
    }
```

Arjun A.S
1BM18CS019

```
else if (child[itemx+1]->n >= 2)
{
        int succ = successor (itemx);
        data[itemx] = succ;
        child[itemx+1] -> deletion (succ);
}
else
{
        merge (itemx)
        child[itemx] -> deletion (item);
}
return;
}
void twothree :: deletion (int item)
{
        if (root == NULL)
                cout << "The tree is empty.";
                return;
        root -> deletion (item);
        if (root -> n == 0)
        {
                node *temp = root;
                if (root -> leaf)
                        root == NULL;
                else
                        root = root -> child[0];
                delete temp;
        }
}
void node :: split (int i, node *y)
{
        node *z = new node (y -> leaf);
        z -> n = i;
        z -> data[0] = y -> data[2];
        if (y -> leaf == false)
        {
                for (int j=0; j<2; j++)
                        z -> child[j] = y -> child[j+2];
        }
        y -> n = 1;
```

Anju A.S
IBM18CS019

```
    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--)
        data[j+1] = data[j];
    data[i] = y->data[1];
    n = n+1;
}
int node::search(int item)
{
    int itemx = 0;
    while (itemx < n && data[itemx] < item)
        ++itemx;
    return itemx;
}
```