Page No.
Date

Sumanth.KV
1BM18CS112
Batch-6

```
struct ~~node~~ avl {
        int data;
        struct avl *l;
        struct avl *r;
};
typedef struct avl * ~~av~~ AVL;
   AVL r= NULL;

~~int~~
 class   avl_tree {
        public:                AVL
           int height (~~avl *~~);
           int difinheight(AVL);
           AVL rr_rot(AVL);
           AVL ll_rot(AVL);
           AVL lr_rot(AVL);
           AVL rl_rot(AVL);
           AVL balance (AVL)
           AVL insert (AVL, int);
           ~~void~~void inorder(AVL);

};

int avl_tree :: height (AVL node) {
        int hei=0
       if (node!=NULL)
          int L_h= height (node->l);
          ind r_h= height (node->r);
       return max(l_h, r_h)+1;

}
```

```
int avl_tree :: difinheight (AVL node) {

    int l_h = height(t->l);
    int r_h = heigh (t->r);
    int balancefactor = l_h - r_h;
    return balance_factor;
}

AVL avl_tree:: rr_rot (AVL par) {
    AVL child;
    child = parent par->r;
    par->r = child->l;
    child-> l = par;
    return child;
}

AVL avltree :: ll_rot (AVL par) {

    AVL child;
    child = par->l;
    par->l = child ->r;
    child->r = par;
    return child;
}

AVL avl_tree :: lr_rot (AVL par) {
    AVL child;
    child = par->l;
    par->l = rr_rot (child);
    return ll_rot (par);
}
```

```
AVL avl_tree::rl_rot(AVL par){
    AVL child;
    child = par->r;
    par->r = ll_rot(child);
    return rr_rot(par);
}

AVL avl_tree::balance(AVL node){
    int bal_fac = difinheight(node);
    if(bal_fac>1){
        if(difinheight(node->l)>0)
            node = ll_rot(node);
        else
            node = lr_rot(node);
    }
    else if(bal_fac<-1){
        if(difinheight(node->r)>0)
            node = rl_rot(node)
        else
            node = rr_rot(node)
    }
    return node;
}

AVL avl_tree::insert(AVL node, int val){
    if(node==NULL)
        node = (AVL)malloc(sizeof(avl));
        node->data = val;
        node->l = NULL;
        node->r = NULL;
        return node;
```

```
        if (val < node->data)
node->l = return insert (node->l, val);
            node = balance (node);

        if (val > node->data)
node->r = return insert (node->r, val);
            node = balance(node);
}

void avl_tree :: inorder (AVL root) {
        if (root != NULL)
            inorder( root->L);
            cout << root->data;
            inorder (root->r);
        }
}
```

```
AVL avl_tree :: remove(int x, AVL node) {
    AVL temp;
    if (node == NULL)
        return NULL;
    else if (x < node->data)
        node->l = remove(x, node->l);
    else if (x > node->data)
        node->r = remove(x, node->r);
    else if (node->l && node->r)
        temp = node->r;
        while(temp->l != NULL) {
            temp = temp->l;
        }
        node->data = temp->data
        node->r = remove(node->data,
                         node->r);
    }
    else {
        temp = node;
        if (node->l == NULL)
            node = node->r;
        else if (node->r == NULL)
            node = node->l;
        delete temp;
    }
    if (node == NULL)
        return NULL;
```

Page No.
Date

```
if(height(node->l) - height(node->r) == 2)
    if (height(node->l->l) -
        height(node->l->r) == 1)
        return LL_rot(node)
    else
        return lr_rot(node)
}
else if (height(node->r) - height(node->l) == 2)
    if (height(node->r->r) - height(node->r->l)
        == 1)
        return rr_rot(node)
    else
        return rl_rot(node)
}
return node;
}
```