Ayush Goyal

190905522 CSE D 62

DAA Lab 7 (Week 7) – Transform and Conquer – I

1. **Modify the solved exercise to find the balance factor for every node in the binary search tree.**

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX(a,b) ((a)>(b) ? a : b)


typedef struct node{
    int val;
    struct node *left;
    struct node *right;
}*NODE;

NODE insert(NODE root,int x){
    if(root==NULL){
        root=(NODE)malloc(sizeof(struct node));
        root->val=x;
        root->left=root->right=NULL;
    }
    else if(x>root->val)
        root->right=insert(root->right,x);
    else if(x<root->val)
        root->left=insert(root->left,x);
    else{
        printf("Duplicate node\n");
        exit(0);
    }
    return(root);
}

void postorder(NODE cur){
    if(cur){
        postorder(cur->left);
        postorder(cur->right);
        printf("%4d",cur->val);
    }
}

void preorder(NODE cur){
    if(cur){
```

```
        printf("%4d",cur->val);
        preorder(cur->left);
        preorder(cur->right);
    }
}

void inorder(NODE cur){
    if(cur){
        inorder(cur->left);
        printf("%4d",cur->val);
        inorder(cur->right);
    }
}

int height(NODE cur){
    if (cur == NULL)
        return -1;
    else
        return MAX(height(cur->left),height(cur->right))+1;
}

void balancefactor(NODE cur){
    static int x;
    if(cur){
        balancefactor(cur->left);
        x = height(cur->left)-height(cur->right);
        printf("\nNode with value %d has a balance factor of %d",cur->val,x);
        balancefactor(cur->right);
    }
}

int main(){
    NODE root = NULL;
    int ch,x;
    do{
        printf("\n1.Enter element(no duplicates)  2. Print elements  3. Show b
alance factor  4.Exit    Enter choice : ");
        scanf("%d",&ch);
        switch (ch){
            case 1 : printf("Enter element : ");
                     scanf("%d",&x);
                     root = insert(root,x);
                     break;
            case 2 : printf("\nInorder traversal is : ");
                     inorder(root);
                     printf("\nPreorder traversal is : ");
                     preorder(root);
                     printf("\nPostorder traversal is : ");
```

```
                    postorder(root);
                    break;
            case 3 : balancefactor(root);
                    break;

            case 4 : break;

            default:
                break;
        }
    }while(ch != 4);
    return 0;
}
```

OUTPUT:

```
D:\CSE\CSE Labs\DAA Lab\Week 7>gcc balancefactorbst.c -o bfbst

D:\CSE\CSE Labs\DAA Lab\Week 7>bfbst

1.Enter element(no duplicates)  2. Print elements  3. Show balance factor  4.Exit    Enter choice : 1
Enter element : 200

1.Enter element(no duplicates)  2. Print elements  3. Show balance factor  4.Exit    Enter choice : 1
Enter element : 100

1.Enter element(no duplicates)  2. Print elements  3. Show balance factor  4.Exit    Enter choice : 1
Enter element : 300

1.Enter element(no duplicates)  2. Print elements  3. Show balance factor  4.Exit    Enter choice : 1
Enter element : 270

1.Enter element(no duplicates)  2. Print elements  3. Show balance factor  4.Exit    Enter choice : 1
Enter element : 250

1.Enter element(no duplicates)  2. Print elements  3. Show balance factor  4.Exit    Enter choice : 2

Inorder traversal is :  100 200 250 270 300
Preorder traversal is :  200 100 300 270 250
Postorder traversal is :  100 250 270 300 200
1.Enter element(no duplicates)  2. Print elements  3. Show balance factor  4.Exit    Enter choice : 3

Node with value 100 has a balance factor of 0
Node with value 200 has a balance factor of -2
Node with value 250 has a balance factor of 0
Node with value 270 has a balance factor of 1
Node with value 300 has a balance factor of 2
1.Enter element(no duplicates)  2. Print elements  3. Show balance factor  4.Exit    Enter choice : 4

D:\CSE\CSE Labs\DAA Lab\Week 7>
```

Time Complexity Analysis:

The number of additions made is A(n) = n and the number of comparisons made to check whether tree is empty is C(n) = 2n+1 for finding the height of the subtree, which is called upon to get the balance factor, where n is the total number of nodes from that node to the bottom most leaf node. Therefore, the order of growth is belonging to **Θ(n)** for that subtree.

2. **Write a program to create the AVL tree by iterative insertion.**

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX(a,b) ((a)>(b) ? a : b)

typedef struct node{
    int val;
    struct node *left;
    struct node *right;
    int height;
}*NODE;

int height(NODE cur){
    if(cur == NULL)
        return 0;
    return cur->height;
}

NODE newNode(int key){
    NODE new = (NODE)malloc(sizeof(struct node));
    new->val=key;
    new->height=1;
    new->left=NULL;
    new->right=NULL;
    return(new);
}

NODE rRotate(NODE y){
    NODE x = y->left;
    NODE T2 = x->right;
    //Rotate here
    x->right = y;
    y->left = T2;
    //Update height here
    y->height = MAX(height(y->left),height(y->right))+1;
    x->height = MAX(height(x->left),height(x->right))+1;
    return x; //new root
}

NODE lRotate(NODE y){
    NODE x = y->right;
    NODE T2 = x->left;
    x->left = y;
    y->right = T2;
    y->height = MAX(height(y->left),height(y->right))+1;
    x->height = MAX(height(x->left),height(x->right))+1;
```

```
    return x;
}

int balFactor(NODE cur){
    if(cur == NULL)
        return 0;
    return height(cur->left) - height(cur->right);
}

NODE insert(NODE new, int k){

    if(new == NULL)
        return(newNode(k));

    if(k < new->val)
        new->left = insert(new->left, k);
    else if(k > new->val)
        new->right = insert(new->right, k);
    else
        return new;
    new->height = MAX(height(new->left),height(new->right))+1;

    int bal = balFactor(new);
    //Now to check all four cases of imbalance we have:

    if(bal>1 && k < new->left->val) //LeftLeftCase
        return rRotate(new);

    if(bal<-1 && k > new->right->val)//RightRightCase
        return lRotate(new);

    if(bal>1 && k > new->left->val){//LeftRightCase
        new->left = lRotate(new->left);
        return rRotate(new);
    }
    if (bal<-1 && k < new->right->val){//RightLeftCase
        new->right = rRotate(new->right);
        return lRotate(new);
    }
    //return unchanged node
    return new;
}

void inorder(NODE cur){
    if(cur != NULL){
        inorder(cur->left);
        printf("%d ",cur->val);
        inorder(cur->right);
```

```
    }
}

int main(){
    NODE root = NULL;
    int ch,x;
    do{
        printf("\n1.Enter element(no duplicates)  2. Print inorder of AVL Tree
  3.Exit    Enter choice : ");
        scanf("%d",&ch);
        switch (ch){
            case 1 : printf("Enter element : ");
                     scanf("%d",&x);
                     root = insert(root,x);
                     break;
            case 2 : printf("\nInorder traversal is : ");
                     inorder(root);
                     break;

            case 3 : break;
            default: break;
        }
    }while(ch != 3);
    return 0;
}
```

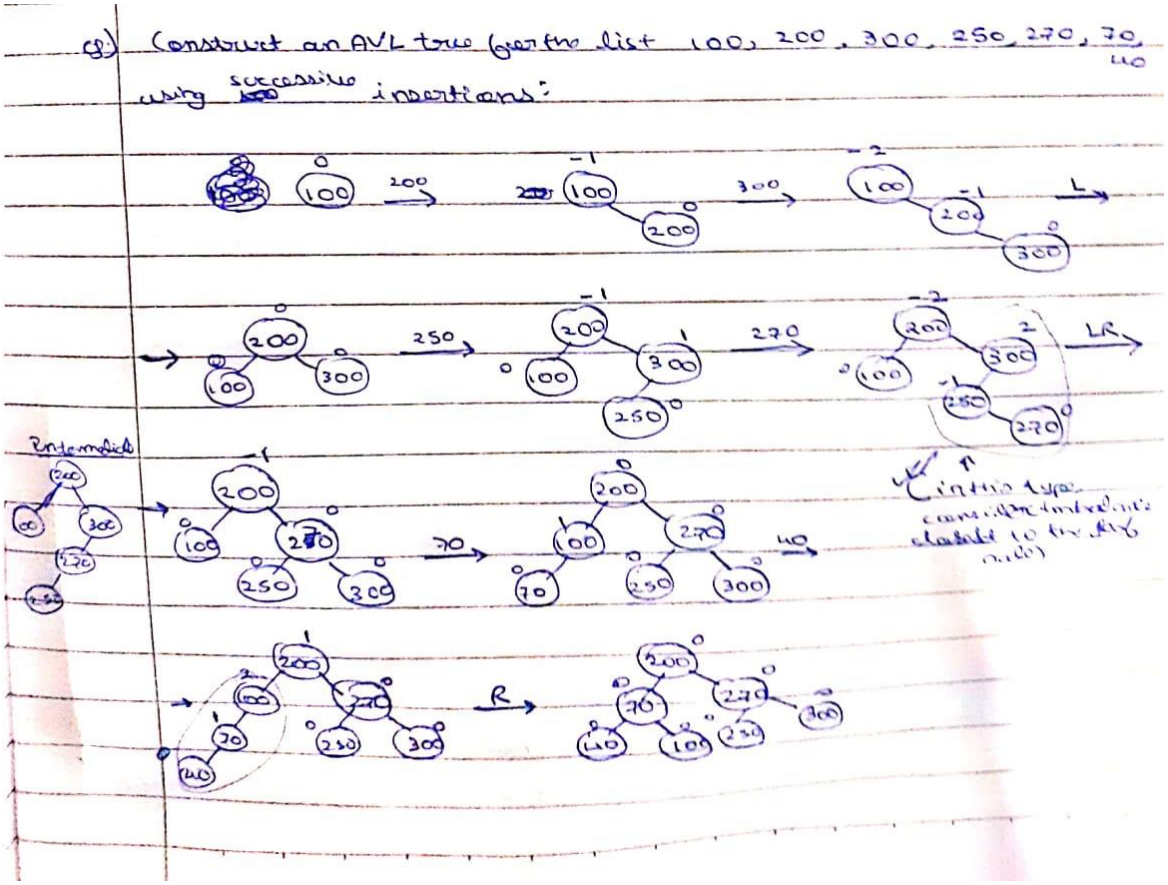
OUTPUT:

```
D:\CSE\CSE Labs\DAA Lab\Week 7>gcc avltree.c -o avltree

D:\CSE\CSE Labs\DAA Lab\Week 7>avltree

1.Enter element(no duplicates)  2. Print inorder of AVL Tree  3.Exit    Enter choice : 1
Enter element : 100

1.Enter element(no duplicates)  2. Print inorder of AVL Tree  3.Exit    Enter choice : 1
Enter element : 200

1.Enter element(no duplicates)  2. Print inorder of AVL Tree  3.Exit    Enter choice : 1
Enter element : 300

1.Enter element(no duplicates)  2. Print inorder of AVL Tree  3.Exit    Enter choice : 1
Enter element : 250

1.Enter element(no duplicates)  2. Print inorder of AVL Tree  3.Exit    Enter choice : 1
Enter element : 270

1.Enter element(no duplicates)  2. Print inorder of AVL Tree  3.Exit    Enter choice : 1
Enter element : 70

1.Enter element(no duplicates)  2. Print inorder of AVL Tree  3.Exit    Enter choice : 1
Enter element : 40

1.Enter element(no duplicates)  2. Print inorder of AVL Tree  3.Exit    Enter choice : 2

Inorder traversal is : 40 70 100 200 250 270 300
1.Enter element(no duplicates)  2. Print inorder of AVL Tree  3.Exit    Enter choice : 3

D:\CSE\CSE Labs\DAA Lab\Week 7>
```

Time Complexity Analysis

The number of nodes on some level 'k' of the binary tree can have value 2^k.
The complexity of the operations on such trees belongs to **$\Theta(\log_2 n)$.**

The steps of conversion for the same tree as used in the example:



THE END