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.**

Recursive Solution: (Iterative is after this)

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;
}
```

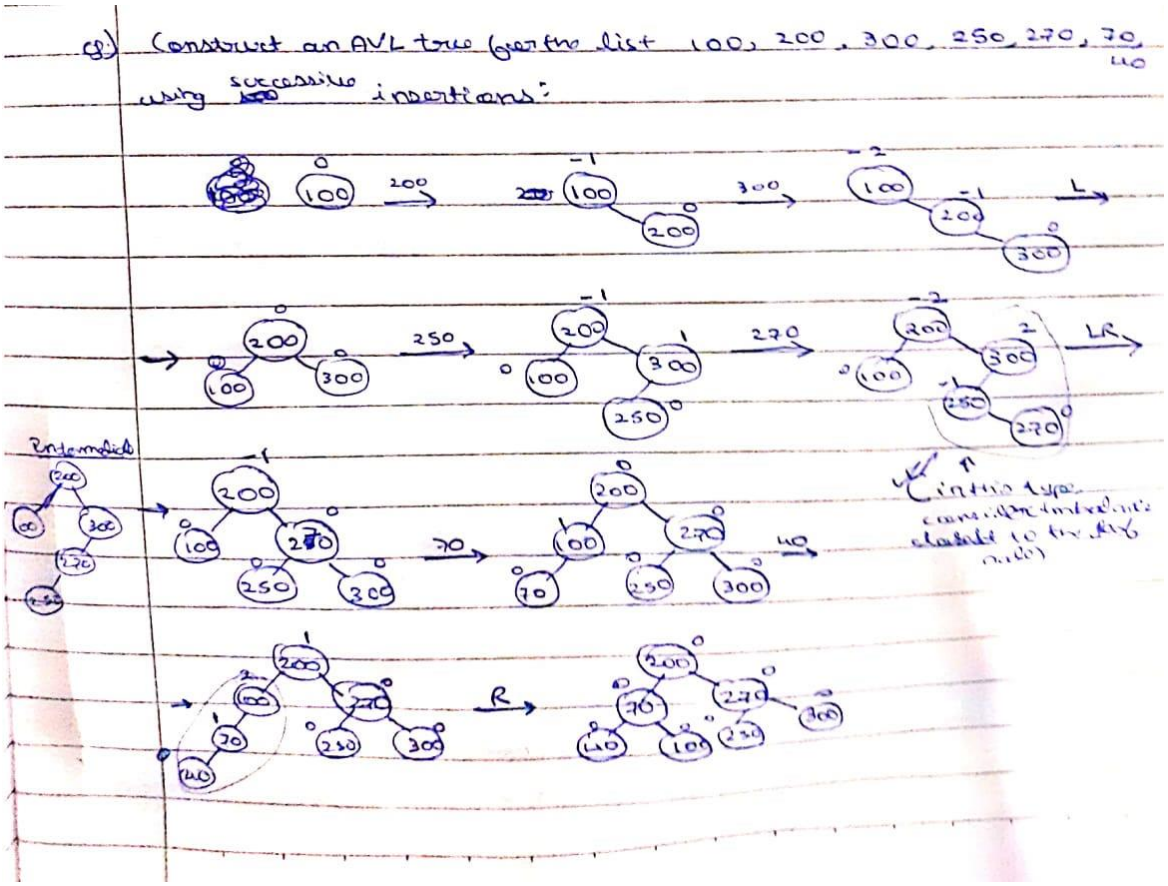
OUTPUT:

```
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:



<u>Iterative Solution:</u>

The same code using iterations instead of recursions will be:

CODE:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int info;
    struct node *left, *right;
```

```c
} NODE;

struct Stack
{
    int top;
    unsigned capacity;
    NODE **array;
};

struct Stack *createStack(unsigned capacity)
{
    struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (NODE **)malloc(stack->capacity * sizeof(NODE *));
    return stack;
}

int isFull(struct Stack *stack)
{
    return stack->top == stack->capacity - 1;
}

int isEmpty(struct Stack *stack)
{
    return stack->top == -1;
}

void push(struct Stack *stack, NODE *item)
{
    if (isFull(stack))
        return;
    stack->array[++stack->top] = item;
    // printf("%d pushed to stack\n", item);
}

NODE *pop(struct Stack *stack)
{
    if (isEmpty(stack))
        return NULL;
    return stack->array[stack->top--];
}

NODE *peek(struct Stack *stack)
{
    if (isEmpty(stack))
        return NULL;
    return stack->array[stack->top];
```

```
}

int max(int x, int y)
{
    return x > y ? x : y;
}

int height(NODE *root)
{
    if (root == NULL)
        return 0;
    return 1 + max(height(root->left), height(root->right));
}

int getBalFactor(NODE *root)
{
    return height(root->left) - height(root->right);
}

NODE *rightRotate(NODE *y)
{
    NODE *x = y->left;
    NODE *T2 = x->right;
    x->right = y;
    y->left = T2;
    return x;
}

NODE *leftRotate(NODE *x)
{
    NODE *y = x->right;
    NODE *T2 = y->left;
    y->left = x;
    x->right = T2;
    return y;
}

NODE *create(NODE *root, int x)
{
    struct Stack *stack = createStack(100);
    NODE *newnode = (NODE *)malloc(sizeof(NODE));
    newnode->info = x;
    newnode->right = NULL;
    newnode->left = NULL;
    NODE *curr = root;
    NODE *trail = NULL;
    while (curr != NULL)
    {
```

```
        trail = curr;
        push(stack, trail);
        if (x < curr->info)
            curr = curr->left;
        else if (x > curr->info)
            curr = curr->right;
        else
        {
            printf("Duplicate element\n");
            exit(0);
        }
    }
    if (trail == NULL)
    {
        trail = newnode;
        return trail;
    }
    else if (x < trail->info)
        trail->left = newnode;
    else
        trail->right = newnode;
    NODE *newRoot = root;
    while (!isEmpty(stack))
    {
        NODE *toBalance = pop(stack);
        NODE *prev = peek(stack);
        int balance = getBalFactor(toBalance);
        if (balance > 1 && x < toBalance->left->info)
        {
            toBalance = rightRotate(toBalance);
        }
        else if (balance < -1 && x > toBalance->right->info)
        {
            toBalance = leftRotate(toBalance);
        }
        else if (balance > 1 && x > toBalance->left->info)
        {
            toBalance->left = leftRotate(toBalance->left);
            toBalance = rightRotate(toBalance);
        }
        else if (balance < -1 && x < toBalance->right->info)
        {
            toBalance->right = rightRotate(toBalance->right);
            toBalance = leftRotate(toBalance);
        }
        if (prev != NULL && prev->info > toBalance->info)
        {
            prev->left = toBalance;
```

```
        }
        else if (prev != NULL)
        {
            prev->right = toBalance;
        }
        newRoot = toBalance;
    }
    return newRoot;
}

void inorder(NODE *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%5d", root->info);
        inorder(root->right);
    }
}

void postorder(NODE *root)
{
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%5d", root->info);
    }
}

void preorder(NODE *root)
{
    if (root != NULL)
    {
        printf("%5d", root->info);
        preorder(root->left);
        preorder(root->right);
    }
}

int printBalanceFactor(NODE *root)
{
    if (root != NULL)
    {
        printf("\nBalance factor of node with value %d : %d", root-
>info, getBalFactor(root));
        printBalanceFactor(root->left);
        printBalanceFactor(root->right);
```

```
    }
}

void main()
{
    int n, x, ch, i;
    NODE *root;
    root = NULL;
    printf(" 1. Insert\n 2. All traversals\n 3. Get Balance Factor\n 4. Exit\n
");
    while (1)
    {
        printf("Enter your choice : ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter node (do not enter duplicate nodes) : ");
            scanf("%d", &x);
            root = create(root, x);
            break;
        case 2:
            printf("\nInorder traversal   : ");
            inorder(root);
            printf("\nPreorder traversal  : ");
            preorder(root);
            printf("\nPostorder traversal : ");
            postorder(root);
            printf("\n");
            break;
        case 3:
            printBalanceFactor(root);
            printf("\n");
            break;
        case 4:
            exit(0);
        default:
            printf("Invalid Choice\n");
        }
    }
}
```

OUTPUT:

```
D:\CSE\CSE Labs\DAA Lab\Week 7 - Transform and Conquer - 1>gcc avliterative.c -o avliterative

D:\CSE\CSE Labs\DAA Lab\Week 7 - Transform and Conquer - 1>avliterative
 1. Insert
 2. All traversals
 3. Get Balance Factor
 4. Exit
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 100
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 200
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 300
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 250
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 270
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 70
Enter your choice : 1
Enter node (do not enter duplicate nodes) : 40
Enter your choice : 2

Inorder traversal   :    40   70  100  200  250  270  300
Preorder traversal  :   200   70   40  100  270  250  300
Postorder traversal :    40  100   70  250  300  270  200
Enter your choice :
```

THE END