Design Analysis and Algorithm – Lab Work

Week 4

Question 1: Write a program to implement the creation and rotation in AVL Tree

Code:

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int key;
    struct Node *left;
    struct Node *right;
    int height;
};
int height(struct Node *N){
    if (N==NULL)
        return 0;
    return N->height;
}
int max(int a, int b){
    return (a>b)?a:b;
}
struct Node* newNode(int key){
    struct Node* node=(struct Node*)malloc(sizeof(struct Node));
    node->key=key;
    node->left=NULL;
    node->right=NULL;
    node->height=1;
    return(node);
}
struct Node *rightRotate(struct Node *y){
    struct Node *x=y->left;
    struct Node *T2=x->right;
    x->right=y;
    y->left=T2;
    y->height=max(height(y->left), height(y->right))+1;
    x->height=max(height(x->left), height(x->right))+1;
    return x;
}
struct Node *leftRotate(struct Node *x){
    struct Node *y=x->right;
    struct Node *T2=y->left;
    y->left=x;
    x->right=T2;
    x->height=max(height(x->left), height(x->right))+1;
    y->height=max(height(y->left), height(y->right))+1;
```

```
    return y;
}
int getBalance(struct Node *N){
    if (N==NULL)
        return 0;
    return height(N->left)-height(N->right);
}
struct Node* insertNode(struct Node* node, int key){
    if (node==NULL)
        return(newNode(key));
    if (key<node->key)
        node->left=insertNode(node->left, key);
    else if(key>node->key)
        node->right=insertNode(node->right, key);
    else
        return node;
    node->height=1+max(height(node->left), height(node->right));
    int balance=getBalance(node);
    if(balance>1&&key<node->left->key)
        return rightRotate(node);
    if(balance<-1&&key>node->right->key)
        return leftRotate(node);
    if(balance>1&&key>node->left->key) {
        node->left=leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance<-1&&key<node->right->key) {
        node->right=rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
struct Node * minValueNode(struct Node* node){
    struct Node* current=node;
    while(current->left!=NULL)
        current=current->left;
    return current;
}
struct Node* deleteNode(struct Node* root, int key){
    if(root==NULL)
        return root;
    if(key<root->key)
        root->left=deleteNode(root->left, key);
    else if(key>root->key)
        root->right=deleteNode(root->right, key);
    else{
        if((root->left==NULL)||(root->right==NULL)) {
            struct Node *temp=root->left?root->left:root->right;
```

```
            if(temp==NULL) {
                temp=root;
                root=NULL;
            } else
                *root=*temp;
            free(temp);
        } else{
            struct Node* temp=minValueNode(root->right);
            root->key=temp->key;
            root->right=deleteNode(root->right, temp->key);
        }
    }
    if(root==NULL)
      return root;
    root->height=1+max(height(root->left),height(root->right));
    int balance=getBalance(root);
    if(balance>1&&getBalance(root->left)>=0)
        return rightRotate(root);
    if(balance>1&&getBalance(root->left)<0){
        root->left=leftRotate(root->left);
        return rightRotate(root);
    }
    if(balance<-1&&getBalance(root->right)<=0)
        return leftRotate(root);
    if(balance < -1&&getBalance(root->right)>0){
        root->right=rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}
void preOrder(struct Node *root){
    if(root != NULL){
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}
int main(){
    struct Node *root=NULL;
    root=insertNode(root, 10);
    root=insertNode(root, 20);
    root=insertNode(root, 30);
    root=insertNode(root, 40);
    root=insertNode(root, 50);
    root=insertNode(root, 25);
    printf("AVL tree:\n");
    preOrder(root);
    printf("\n");
```

```
    root=deleteNode(root, 10);
    printf("After deletion of 10:\n");
    preOrder(root);
    printf("\n");
    return 0;
}
```

Output:

```
AVL tree:
30 20 10 25 40 50
After deletion of 10:
30 20 25 40 50
PS C:\Users\kaush\Documents\College\Fourth Sem\DAA>
```

Space Complexity: O(n)

Time Complexity: O(nlogn) – Creation

O(logn) - Insertion

Question 2: Write a program to implement creation and rotation in Red Black Tree.

Code:

```
#include<stdio.h>
#include<stdlib.h>
typedef enum{RED,BLACK}Color;
typedef struct RBNode{
    int data;
    Color color;
    struct RBNode*left,*right,*parent;
}RBNode;
RBNode*root=NULL;
RBNode*createNode(int data){
    RBNode*n=(RBNode*)malloc(sizeof(RBNode));
    n->data=data;
    n->color=RED;
    n->left=n->right=n->parent=NULL;
    return n;
}
void leftRotate(RBNode**root,RBNode*x){
    RBNode*y=x->right;
    x->right=y->left;
    if(y->left){
        y->left->parent=x;
    }
    y->parent=x->parent;
```

```
    if(!x->parent){
        *root=y;
    }
    else if(x==x->parent->left){
        x->parent->left=y;
    }
    else{
        x->parent->right=y;
    }
    y->left=x;
    x->parent=y;
}
void rightRotate(RBNode**root,RBNode*y){
    RBNode*x=y->left;
    y->left=x->right;
    if(x->right){
        x->right->parent=y;
    }
    x->parent=y->parent;
    if(!y->parent){
        *root=x;
    }
    else if(y==y->parent->left){
        y->parent->left=x;
    }
    else{
        y->parent->right=x;
    }
    x->right=y;
    y->parent=x;
}
void fixInsert(RBNode**root,RBNode*z)
{
    while(z!=*root&&z->parent->color==RED){
        RBNode*p=z->parent;
        RBNode*g=p->parent;
        if(p==g->left){
            RBNode*u=g->right;
            if(u&&u->color==RED){
                g->color=RED;
                p->color=BLACK;
                u->color=BLACK;
                z=g;
            }
            else{
                if(z==p->right){
                    z=p;
                    leftRotate(root,z);
```

```
                }
                p->color=BLACK;
                g->color=RED;
                rightRotate(root,g);
            }
        }
        else{
            RBNode*u=g->left;
            if(u&&u->color==RED){
                g->color=RED;
                p->color=BLACK;
                u->color=BLACK;
                z=g;
            }
            else{
                if(z==p->left){
                    z=p;
                    rightRotate(root,z);
                }
                p->color=BLACK;
                g->color=RED;
                leftRotate(root,g);
            }
        }
    }
    (*root)->color=BLACK;
}
void insertRB(int data){
    RBNode*z=createNode(data);
    RBNode*y=NULL;
    RBNode*x=root;
    while(x){
        y=x;
        if(z->data<x->data){
            x=x->left;
        }
        else{
            x=x->right;
        }
    }
    z->parent=y;
    if(!y){
        root=z;
    }
    else if(z->data<y->data){
        y->left=z;
    }
    else{
```

```
        y->right=z;
    }
    fixInsert(&root,z);
}
void inorder(RBNode*r){
    if(r){
        inorder(r->left);
        printf("%d ",r->data);
        inorder(r->right);
    }
}
int main(){
    int a[]={20,15,25,10,5,1,30};
    int n=sizeof(a)/sizeof(a[0]);
    for(int i=0;i<n;i++){
        insertRB(a[i]);
    }
    inorder(root);
    return 0;
}
```

Output:

```
  1 5 10 15 20 25 30
○ PS C:\Users\kaush\Documents\College\Fourth Sem\DAA>
```

Space Complexity: O(n)

Time Complexity: O(logn)