**Programs**

1. Write a program to construct an AVL tree for the given set of keys. Also write function for deleting a key from the given AVL tree.

Ans:

#include<stdio.h>

#include<conio.h>

struct node

{

int data;

struct node \*left;

struct node \*right;

int ht;

};

struct node \*root = NULL;

struct node \*create(int);

struct node \*insert(struct node\*, int);

struct node \*delete(struct node\*, int);

struct node \*search(struct node\*, int);

struct node \*rotate\_left(struct node\*);

struct node \*rotate\_right(struct node\*);

int balance\_factor(struct node\*);

int height(struct node\*);

void inorder(struct node\*);

void preorder(struct node\*);

void postorder(struct node\*);

void main()

{

int ch,data,i,n;

struct node\* result = NULL;

clrscr();

while (ch!=7)

{

printf("\n---------------------------MENU-----------------------------");

printf("\n1.Insert \t2.Delete \t3.Search \t4.Inorder");

printf("\n5.Preorder \t6.Postorder \t7.Exit");

printf("\n------------------------------------------------------------");

printf("\nEnter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:printf("Enter how many elements you want to insert:");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("Enter value:");

scanf("%d",&data);

root=insert(root,data);

}

break;

case 2:printf("Enter data:");

scanf("%d",&data);

root=delete(root,data);

printf("Key deleted..");

break;

case 3:printf("Enter data:");

scanf("%d",&data);

result=search(root,data);

if(result==NULL)

printf("Node not found!");

else

printf("Node found");

break;

case 4:printf("Inorder traversal:");

inorder(root);

break;

case 5:printf("Preorder tracersal:");

preorder(root);

break;

case 6:printf("Postorder traversal:");

postorder(root);

break;

case 7:exit(0);

default:printf("Invalid Choice..");

}

}

getch();

}

struct node \*create(int data)

{

struct node \*new\_node=(struct node \*)malloc(sizeof(struct node));

if (new\_node==NULL)

{

printf("Memory can't be allocated..");

return NULL;

}

new\_node->data=data;

new\_node->left=NULL;

new\_node->right=NULL;

return new\_node;

}

struct node \*rotate\_left(struct node \*root)

{

struct node \*right\_child=root->right;

root->right=right\_child->left;

right\_child->left=root;

root->ht=height(root);

right\_child->ht=height(right\_child);

return right\_child;

}

struct node \*rotate\_right(struct node \*root)

{

struct node \*left\_child=root->left;

root->left=left\_child->right;

left\_child->right=root;

root->ht=height(root);

left\_child->ht=height(left\_child);

return left\_child;

}

int balance\_factor(struct node \*root)

{

int lh,rh;

if(root==NULL)

return 0;

if(root->left==NULL)

lh=0;

else

lh=1+root->left->ht;

if(root->right==NULL)

rh=0;

else

rh=1+root->right->ht;

return lh-rh;

}

int height(struct node \*root)

{

int lh,rh;

if(root==NULL)

return 0;

if(root->left==NULL)

lh=0;

else

lh=1+root->left->ht;

if(root->right==NULL)

rh=0;

else

rh=1+root->right->ht;

if(lh>rh)

return (lh);

return (rh);

}

struct node \*insert(struct node \*root,int data)

{

if(root==NULL)

{

struct node \*new\_node=create(data);

if(new\_node==NULL)

return NULL;

root=new\_node;

}

else if(data>root->data)

{

root->right=insert(root->right,data);

if(balance\_factor(root)==-2)

{

if(data>root->right->data)

root=rotate\_left(root);

else

{

root->right=rotate\_right(root->right);

root=rotate\_left(root);

}

}

}

else

{

root->left=insert(root->left, data);

if(balance\_factor(root)==2)

{

if(data<root->left->data)

root=rotate\_right(root);

else

{

root->left=rotate\_left(root->left);

root=rotate\_right(root);

}

}

}

root->ht=height(root);

return root;

}

struct node \*delete(struct node \*root,int x)

{

struct node \*temp=NULL;

if(root==NULL)

return NULL;

if(x>root->data)

{

root->right=delete(root->right,x);

if(balance\_factor(root)==2)

{

if(balance\_factor(root->left)>=0)

root=rotate\_right(root);

else

{

root->left=rotate\_left(root->left);

root=rotate\_right(root);

}

}

}

else if(x<root->data)

{

root->left=delete(root->left,x);

if(balance\_factor(root)==-2)

{

if(balance\_factor(root->right)<=0)

root=rotate\_left(root);

else

{

root->right=rotate\_right(root->right);

root=rotate\_left(root);

}

}

}

else

{

if(root->right!=NULL)

{

temp=root->right;

while(temp->left!=NULL)

temp=temp->left;

root->data=temp->data;

root->right=delete(root->right,temp->data);

if(balance\_factor(root)==2)

{

if(balance\_factor(root->left)>=0)

root = rotate\_right(root);

else

{

root->left=rotate\_left(root->left);

root=rotate\_right(root);

}

}

}

else

return (root->left);

}

root->ht=height(root);

return (root);

}

struct node \*search(struct node \*root,int key)

{

if(root==NULL)

return NULL;

if(root->data==key)

return root;

if(key>root->data)

search(root->right,key);

else

search(root->left,key);

}

void inorder(struct node \*root)

{

if(root==NULL)

return;

inorder(root->left);

printf("%d ",root->data);

inorder(root->right);

}

void preorder(struct node \*root)

{

if(root==NULL)

return;

printf("%d ",root->data);

preorder(root->left);

preorder(root->right);

}

void postorder(struct node \*root)

{

if(root==NULL)

return;

postorder(root->left);

postorder(root->right);

printf("%d ",root->data);

}

Output:





