Name :- Vishal Pandey

Roll no. :- 68

Branch:- AIML(CSE)

Batch :- E4

DSA LAB

LAB ASSIGNMENT 7

Lab Assignment 7:

Implement a Binary search treewith operations – create,

search, insert, delete, inorder, preorder and postorder. Write a menu driven

program that performs the above operations also counts the total nodes and total

leaf nodes in the tree. int count(T) – returns the total number of nodes from BST

int countLeaf(T) – returns the total number of leaf nodes from BST

Program:

PROGRAM :-

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int data;

struct Node \* left;

struct Node \* right;

};

struct Node\* create(int data);

struct Node\* insert(struct Node \*root,int data);

struct Node\* delete(struct Node \*root,int data);

struct Node\* find\_minright(struct Node \*root);

struct Node\* search(struct Node \*root, int data);

void inorder(struct Node\* root);

void preorder(struct Node\* root);

void postorder(struct Node\* root);

void display(struct Node\* root);

int countall(struct Node\* root);

int countleaf(struct Node\* root);

int main(){

struct Node \* root,\* key;

int com;

while(com!=-1){

printf("Enter Commands :\n1)To create binary tree\n2)To insert node\n3)To

display BST\n");

printf("4)To delete node\n5) To search a node\n6)To count total number of

nodes\n7)To count leaf nodes\n-1)To exit\n");

scanf("%d",&com);

if(com==1){

printf("\nEnter the value of node to create\n");

int data;

scanf("%d",&data);

root=create(data);

printf("\nThe nodes of BST in inorder\n");

display(root);

}

else if(com==2){

printf("\nEnter the value of node to insert\n");

int data;

scanf("%d",&data);

root=insert(root,data);

printf("\nAfter insert the BST is in inorder\n");

display(root);

}

else if(com==3){

display(root);

}

else if(com==4){

printf("\nEnter node you want to delete\n");

int data;

scanf("%d",&data);

root=delete(root,data);

display(root);

}

else if(com==5){

printf("\nEnter node you want to search\n");

int data;

scanf("%d",&data);

root=search(root,data);

if(root==NULL){

printf("Value %d is not present in root\n",data);

}

else{

printf("Value %d is present in BST\n",data);

}

}

else if(com==6){

printf("The number of total nodes %d\n",countall(root));

}

else if(com==7){

printf("The number of leaf nodes %d\n",countleaf(root));

}

else if(com==-1) break;

else{

printf("Enter valid command\n");

}

}

}

int countall(struct Node \* root){

if(root==NULL){

return 0;

}

return 1+countall(root->left)+countall(root->right);

}

int countleaf(struct Node \* root){

if(root==NULL){

return 0;

}

else if(root->left==NULL && root->right==NULL){

return 1;

}

else{

return countleaf(root->left)+countleaf(root->right);

}

}

void display(struct Node \* root){

int com;

printf("Enter Commands :\n1)Inorder\n2)Preorder\n3)Postorder\n");

scanf("%d",&com);

if(com==1){

inorder(root);

}

else if(com==2){

preorder(root);

}

else if(com==3){

postorder(root);

}

else{

printf("Enter valid command\n");

}

printf("\n");

}

void postorder(struct Node\* root){

if(root != NULL){

postorder(root->left);

postorder(root->right);

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

}

}

void preorder(struct Node\* root){

if(root!=NULL){

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

preorder(root->left);

preorder(root->right);

}

}

void inorder(struct Node\* root){

if(root != NULL){

inorder(root->left);

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

inorder(root->right);

}

}

struct Node\* create(int data){

struct Node \* tree;

tree=(struct Node \*)malloc(sizeof(struct Node));

tree->data=data;

tree->left=NULL;

tree->right=NULL;

return tree;

}

struct Node\* insert(struct Node \*root,int data){

if(root==NULL){

return create(data);

}

else if(data>root->data){

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

}

else{

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

}

return root;

}

struct Node\* delete(struct Node \*root,int data){

if(root==NULL){

return NULL;

}

if(data>root->data){

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

}

else if(data<root->data){

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

}

else{

if(root->left==NULL && root->right==NULL){

return NULL;

}

else if(root->left==NULL || root->right==NULL){

struct Node \* temp;

if(root->left){

temp=root->right;

}

else{

temp=root->left;

}

free(root);

return temp;

}

else{

struct Node \* temp=find\_minright(root->right);

root->data=temp->data;

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

}

}

return root;

}

struct Node\* find\_minright(struct Node\* root){

if(root==NULL){

return NULL;

}

else if(root->left != NULL){

return find\_minright(root->left);

}

return root;

}

struct Node\* search(struct Node \*root, int data){

if(root==NULL || root->data==data){

return root;

}

else if(data>root->data){

return search(root->right,data);

}

else{

return search(root->left,data);

}

}

OUTPUT:-