**Experiment No.4**

**Aim:** Implementation of operations (insertion, deletion, counting of nodes, counting of leaf nodes etc.) in a binary search tree.

**Code:**

// C program to demonstrate delete operation in binary search tree

#include<stdio.h>

#include<stdlib.h>

struct node

{

int key;

struct node \*left, \*right;

};

// A utility function to create a new BST node

struct node \*newNode(int item)

{

struct node \*temp = (struct node \*)malloc(sizeof(struct node));

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

static int count = 0,leaf=0;

// A utility function to count the number of nodes in the tree

int countnodes(struct node \*root)

{

if(root != NULL)

{

countnodes(root->left);

count++;

countnodes(root->right);

}

return count;

}

// A utility function to count the number of leaf nodes in the tree

int leafnodes(struct node\* root)

{

if (root != NULL)

{

leafnodes(root->left);

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

{

leaf++;

}

leafnodes(root->right);

}

return leaf;

}

// A utility function to do inorder traversal of BST

void inorder(struct node \*root)

{

if (root != NULL)

{

inorder(root->left);

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

inorder(root->right);

}

}

/\* A utility function to insert a new node with given key in BST \*/

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

{

/\* If the tree is empty, return a new node \*/

if (node == NULL) return newNode(key);

/\* Otherwise, recur down the tree \*/

if (key < node->key)

node->left = insert(node->left, key);

else

node->right = insert(node->right, key);

/\* return the (unchanged) node pointer \*/

return node;

}

/\* Given a non-empty binary search tree, return the node with minimum

key value found in that tree. Note that the entire tree does not

need to be searched. \*/

struct node \* minValueNode(struct node\* node)

{

struct node\* current = node;

/\* loop down to find the leftmost leaf \*/

while (current && current->left != NULL)

current = current->left;

return current;

}

/\* Given a binary search tree and a key, this function deletes the key

and returns the new root \*/

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

{

// base case

if (root == NULL) return root;

// If the key to be deleted is smaller than the root's key,

// then it lies in left subtree

if (key < root->key)

root->left = deleteNode(root->left, key);

// If the key to be deleted is greater than the root's key,

// then it lies in right subtree

else if (key > root->key)

root->right = deleteNode(root->right, key);

// if key is same as root's key, then This is the node

// to be deleted

else

{

// node with only one child or no child

if (root->left == NULL)

{

struct node \*temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL)

{

struct node \*temp = root->left;

free(root);

return temp;

}

// node with two children: Get the inorder successor (smallest

// in the right subtree)

struct node\* temp = minValueNode(root->right);

// Copy the inorder successor's content to this node

root->key = temp->key;

// Delete the inorder successor

root->right = deleteNode(root->right, temp->key);

}

return root;

}

// Driver Program to test above functions

int main()

{

/\* Let us create following BST

50

/ \

30 70

/ \ / \

20 40 60 80 \*/

struct node \*root = NULL;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 70);

root = insert(root, 60);

root = insert(root, 80);

printf("Inorder traversal of the given tree \n");

inorder(root);

printf("\nNumber of nodes in the given tree: %d \n",countnodes(root));

printf("\nNumber of leaf nodes in the given tree: %d \n",leafnodes(root));

count=0;

leaf=0;

printf("\nDelete 20\n");

root = deleteNode(root, 20);

printf("Inorder traversal of the modified tree \n");

inorder(root);

printf("\nNumber of nodes in the given tree: %d \n",countnodes(root));

printf("\nNumber of leaf nodes in the given tree: %d \n",leafnodes(root));

count=0;

leaf=0;

printf("\nDelete 30\n");

root = deleteNode(root, 30);

printf("Inorder traversal of the modified tree \n");

inorder(root);

printf("\nNumber of nodes in the given tree: %d \n",countnodes(root));

printf("\nNumber of leaf nodes in the given tree: %d \n",leafnodes(root));

count=0;

leaf=0;

printf("\nDelete 50\n");

root = deleteNode(root, 50);

printf("Inorder traversal of the modified tree \n");

inorder(root);

printf("\nNumber of nodes in the given tree: %d \n",countnodes(root));

printf("\nNumber of leaf nodes in the given tree: %d \n",leafnodes(root));

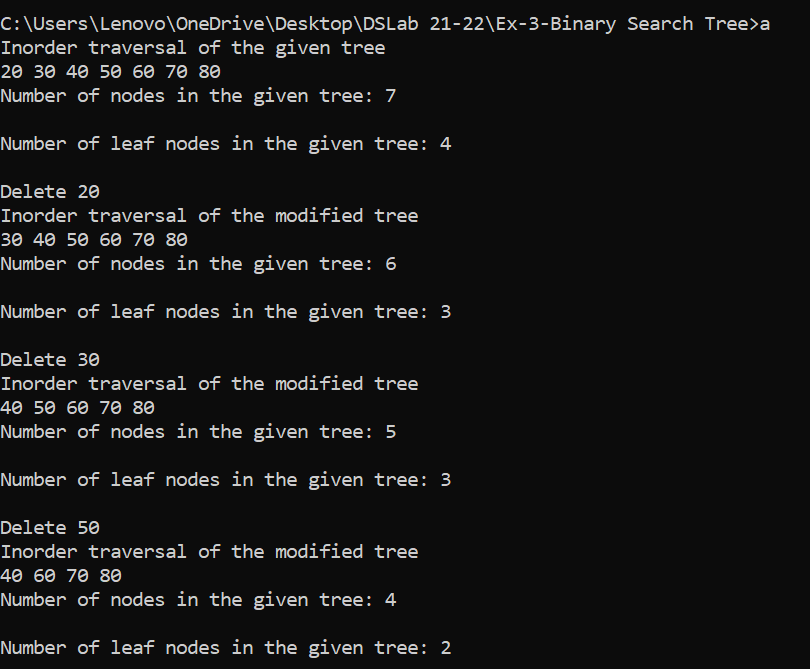
count=0;

leaf=0;

return 0;

}

OUTPUT:



Conclusion: - Thus in this experiment we implemented of operations insertion, deletion, counting of nodes, counting of leaf nodes in a binary search tree