**DATE:-**

**ASIGNEMENT NUMBER:-**

**PROBLEM STATEMENT:-**

**THEORY:-**

Binary Search Tree, is a node-based binary tree data structure which has the following properties:

* The left subtree of a node contains only nodes with keys lesser than the node’s key.
* The right subtree of a node contains only nodes with keys greater than the node’s key.
* The left and right subtree each must also be a binary search tree.  
  There must be no duplicate nodes.



The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no ordering, then we may have to compare every key to search a given key.

**ALGORITHM:-**

**Input:-**

**Output:-**

**Steps:-**

**Algorithm\_Create\_BST()**

1. If(root = NULL)

Then

* 1. set root = GetNode()
  2. set root->left = root->right = NULL
  3. set root ->data = elem
  4. return root

1. Else
   1. If(elem<root->data)

Then

* + 1. set root->left = Create\_BST()
  1. Else
     1. If(elem > root->data)

Then

* + - 1. set root->right = Create\_BST()
    1. Else
       1. Print “Duplicate Element !! Not Allowed !!!”

EndIf

EndIf

EndIf

1. Return root

**Algorithm\_Inorder(root)**

1. If (root != Null)
   1. Inorder(root->left)
   2. Print root->data
   3. Inorder(root->right)

EndIf

**Algorithm\_Preorder(root)**

1. If(root != Null)
   1. Print root->data
   2. Preorder(root->left)
   3. Preorder(root->right)

EndIf

**Algorithm\_Postorder(root)**

1. If(root != Null)
   1. Postorder(root->left)
   2. Postorder(root->right)
   3. Print root->data

**SOURCE CODE:-**

#include<stdio.h>

#include<stdlib.h>

typedef struct tnode

{

int data;

struct tnode \*right,\*left;

}TNODE;

TNODE \*CreateBST(TNODE \*, int);

void Inorder(TNODE \*);

void Preorder(TNODE \*);

void Postorder(TNODE \*);

TNODE \*Delete(TNODE \*,int);

TNODE \*minvaluenode(TNODE \*);

int main()

{

TNODE \*root=NULL;

int ch,data,n,i;

do

{

printf("\n1-Creation of BST");

printf("\n2-Traverse in Inorder");

printf("\n3-Traverse in Preorder");

printf("\n4-Traverse in Postorder");

printf("\n5-Delete an Element");

printf("\n6-Exit\n");

printf("\nEnter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1: root=NULL;

printf("\n\nBST for How Many Nodes ?:");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

printf("\nEnter the Data for Node %d: ",i);

scanf("%d",&data);

root=CreateBST(root,data);

}

printf("\nBST with %d nodes is ready to Use!!\n",n);

break;

case 2:Inorder(root);

break;

case 3:Preorder(root);

break;

case 4:Postorder(root);

break;

case 5:printf("\nEnter the node to be deleted:");

scanf("%d",&data);

Delete(root,data);

break;

case 6: printf("\n\n Terminating \n\n");

break;

default: printf("\n\nInvalid Option !!! Try Again !! \n\n");

break;

}

}while(ch != 6);

}

TNODE \*CreateBST(TNODE \*root, int elem)

{

if(root == NULL)

{

root=(TNODE \*)malloc(sizeof(TNODE));

root->left= root->right = NULL;

root->data=elem;

return root;

}

else

{

if( elem < root->data )

root->left=CreateBST(root->left,elem);

else

if( elem > root->data )

root->right=CreateBST(root->right,elem);

else

printf(" Duplicate Element !! Not Allowed !!!");

return(root);

}

}

void Inorder(TNODE \*root)

{

if( root != NULL)

{

Inorder(root->left);

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

Inorder(root->right);

}

}

void Preorder(TNODE \*root)

{

if( root != NULL)

{

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

Preorder(root->left);

Preorder(root->right);

}

}

void Postorder(TNODE \*root)

{

if( root != NULL)

{

Postorder(root->left);

Postorder(root->right);

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

}

}

TNODE \* minvaluenode(TNODE \* node)

{

TNODE \* current = node;

while(current->left != NULL)

{

current = current->left;

}

return current;

}

TNODE \*Delete(TNODE \*root, int data)

{

if(root == NULL)

{

printf("\nWouldn't find the Element!!");

return root;

}

if(data < root->data)

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

else if(data > root->data)

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

else

{

if(root->left == NULL)

{

TNODE \* temp = root->right;

free(root);

return temp;

}

else if(root->right == NULL)

{

TNODE \* temp = root->left;

free(root);

return temp;

}

TNODE \* temp = minvaluenode(root->right);

root->data = temp->data;

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

root->data = temp->data;

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

}

return root;

}

**INPUT AND OUTPUT:-**

**DISCUSSION:-**

------------------------