**BINARY SEARCH TREES**

**EXPT NO: 5** **DATE: 26/11/21**

**AIM**

**1)** Write a program to perform the following operations in binary search tree.

**2)** Write a program to implement following traversals in binary tree

1. Pre-order
2. In-order
3. Post-order
4. Level order

**3)** Write a program to display the width of binary search tree and also the level.

**4)** Write a program to input a level and display nodes of that level from left to

Right.

**THEORY**

A is a non-linear data structure in which data is organized in an hierarchal manner. It consists of a finite collection of elements called nodes or vertices. (Refer Table A, Page No: )

A formal definition of trees would be

**Tree can be defined recursively as: -**

A tree is a finite set of nodes such that:

1) There is a distinguished node called the root

2) The remaining nodes are partitioned into n>=0 disjoint sets T1, T2, …. Tn where each of these sets is a tree. The sets T1, T2……Tn are the subtrees of the root.

**Binary Trees: -**

In a binary tree no node can have more than two children i.e. node can have 0, 1 or 2 children. Each child is designated as left child or right child.

**Binary Tree can be defined recursively as: -**

A binary tree is a finite set of nodes that is:

1) Either empty or

2) Consists of a distinguished node called root and remaining nodes are portioned into two disjoint sets T1 and T2 and both of them are binary trees. T1 is called left subtree and T2 is called the right subtree.

**Linked representation of binary trees**

In linked-list representation, we take three members in a node of the tree. First member is a pointer that stores the address of the left child, second member is for data and the third member is assigned a pointer that stores the address of the right child.

**The structure of the node can be declared as-**

struct node

{

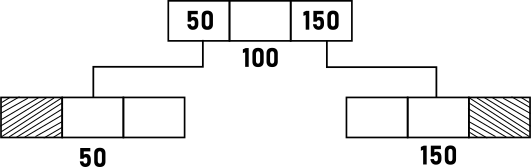
struct node \*Left;

char data;

struct node \*right;

};

If node has no left child, then left pointer should be NULL and if it has no right child then the right pointer should be NULL.



**Binary Search Trees**

One of the most important uses of binary tree is in searching. Binary search trees that are specially organized for the purpose of searching. N element can be searched in average O(logN)time where N is the number of nodes. In binary search tree, a key is associated with each node.

**Binary Search Tree can be defined recursively as: -**

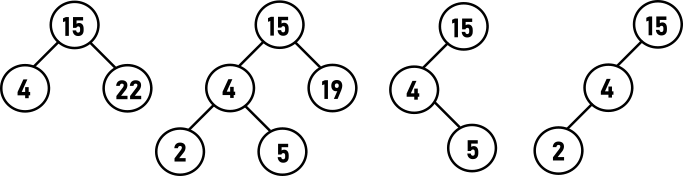
A binary search tree is a binary tree that may be empty and if it is not empty then it satisfies the following properties: -

1) All the keys in the left subtree of the root are less than the key in the root.

2) All the keys in the right subtrees of the root are greater than the key in the root.

3) Left and right subtrees of the root are also binary search trees

(We have assumed that all the keys of the binary search tree are distinct)



**Traversing in Binary trees**

Traversing a binary tree means visiting each node of the tree exactly one. Traversal of tree gives a linear order of the nodes, i.e. all node can be put in one line.

**Preorder Traversal (NLR)**

1. Visit the root (N)
2. Traverse the left subtree of root in Preorder (L)
3. Traverse the right subtree of root in Preorder (R)

**Inorder Traversal (LNR)**

1. Traverse the left subtree of root in Inorder (L)
2. Visit the root (N)
3. Traverse the right subtree of root in Inorder (R)

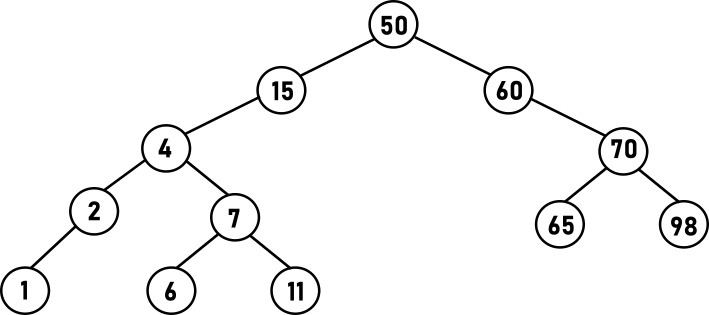
**Postorder Traversal (LRN)**

1. Traverse the left subtree of root in Postorder (L)
2. Traverse the right subtree of root in Postorder (R)
3. Visit the root (N)

**Level order Traversal (LRN)**

1. Insert root node in the queue
2. Delete the node from the front of the queue and visit it
3. Insert the left child of visited node in the queue at the end
4. Insert the right child of the visited node in the queue at the end
5. Repeat steps 2, 3, 4 till the queue is not empty

**Example:**



|  |  |
| --- | --- |
| **TRAVERSAL NAME** | **OUTPUT** |
| Preorder | 50 15 4 2 1 7 6 11 60 70 65 98 |
| Inorder | 1 2 4 6 7 11 15 50 60 65 70 98 |
| Postorder | 1 2 6 11 7 4 15 65 98 70 60 50 |
| Level order | 50 15 60 4 70 2 7 65 98 1 6 11 |

**Advantages of Binary Search Trees**

1. Searching become very efficient in a binary search tree since, we get a hint at each step, about which subtree contains the desired element.

2. The binary search tree is considered as efficient data structure in compare to arrays and linked lists. In searching process, it removes half sub-tree at every step.

3. It also speed up the insertion and deletion operation as compare to that in array and linked list.

4. Searching for an element in binary search tree takes O(log2n) time. In worst case, the time it takes to search an element is O(n).

**Table A**

|  |  |
| --- | --- |
| **TERMINOLOGY** | **EXPLANATION** |
| Node | Each element in the tree is know as a node |
| Edges | The lines connecting the nodes are called edges or branches |
| Parent Node | The immediate predecessor of a node |
| Child Node | The immediate successors of a node |
| Root Node | Specially designated node that does not have any parent |
| Leaf Node | A node that does not have any child |
| Level | Level of any node is defined as the distance of that node from the root |
| Height | The total number of levels in a tree is the height of the tree |
| Sibling | Two or more nodes which have the same parent are called siblings |
| Subtree | A tree may be divided into subtree which can further be divided into subtrees. The first node of that subtree is called the root of that subtree |
| Degree | The number of subtrees or children of a node is called its degree |
| Forest | A forest is a set of n disjoint trees where n>=0. If the root is removed, we get a forest consisting of its subtrees. |

**ALGORITHMS**

**1)**

**void search(struct node\*p,int data)**

1. if(p==NULL)

1. Output data not found

2. else if(p->data>data)

1. search(p->left,data)

3. else if(p->data<data)

1. search(p->right,data)

4. else

1. Output data found

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

1. if(p==NULL)

1. p=(struct node\*)malloc(sizeof(struct node))

2. p->data=data

3. p->left=NULL

4. p->right=NULL

2. else if(p->data>data)

1. p->left=insert(p->left,data)

3. else if(p->data<data)

1. p->right=insert(p->right,data)

4. else

1. Output duplicate data

5. return p

**struct node \*del(struct node\*p,int data)**

1. struct node\*t

2. if(p==NULL)

1. Output data not found

3. else if(p->data>data)

1. p->left=del(p->left,data)

4. else if(p->data<data)

1. p->right=del(p->right,data)

5.else

1. if(p->left!=NULL&&p->right!=NULL)

1. t=p->right

2. while(t-left!=NULL)

1. t=t->left

2. p->data=t->data

3. p->right=del(p->right, t->data)

2. else

1. t=p

2. if(p->left!=NULL)

1. p=p->left

3. else if(p->right!=NULL)

1. p=p->right

4. else

1. p=NULL

5. free(t)

6. return p

**void display(struct node\*root,int space)**

1. Declare int i

2. if(root==NULL)

1. return

3. space+=CNT

4. display(root->right,space)

5. for(i=CNT;i<space;i++)

1. Output root->data

6. Display(root->left,space)

**2)**

**void preoerder(struct node\*root)**

1. if(root==NULL)

1. return

2. Output root->data

3. preoerder(root->left)

4. preoerder(root->right)

**void levelorder(struct node\*root, int lv)**

1. if(root==NULL)

1. return

2. else if(lv==0)

1. Output root->data

3. else

1. levelorder(root->left,lv-1)

2. levelorder(root->right,lv-1)

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

1. if(p==NULL)

1. p=(struct node\*)malloc(sizeof(struct node))

2. p->data=data

3. p->left=NULL

4. p->right=NULL

2.else if(p->data>dta)

1. p->left=insert(p->left,data)

3. else if(p->data<data)

1. p->right=insert(p->right,data)

4. else

1. Output duplicate data

5. return p

**int height(struct node\*root)**

1. Declare int lheight,rheight

2. if(root==NULL)

1. return 0

3. lheight=height(root->left)

4. rheight=height(root->right)

5. if(lheight>rheight)

1. return lheight+1

6. else

1. return rheight+1

Void postorder(struct node\*root)

1. if(root==NULL)

1. return

2. postorder(root->left)

3. postorder(root->right)

4. Output root->data

**void inorder(struct node\*root)**

1. if (root==NULL)

1. return

2. inorder(root->left)

3. Output root->data

4. inorder root->right

**CODES**

**1)**

#include<stdio.h>

#include<stdlib.h>

#define CNT 10

int key,x=0;

struct node{

struct node \*left;

int data;

struct node \*right;

};

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

{

if(p==NULL)

{

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

p->data=data;

p->left=NULL;

p->right=NULL;

}

else if(data < p->data)

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

else if(data > p->data)

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

else

printf("DUPLICATE VALUE\n");

return p;

}

void search(struct node\* p, int data)

{

if(p==NULL)

printf("DATA NOT FOUND\n");

else if(p->data > data)

search(p->left, data);

else if(p->data < data)

search(p->right, data);

else

printf("DATA FOUND\n");

}

struct node \*del(struct node \*p,int key)

{

struct node \*temp,\*succ;

if(p==NULL)

{

printf("%d IS NOT PRESENT IN THE TREE\n",key);

return p;

}

if(key < p->data)

p->left=del(p->left,key);

else if(key > p->data)

p->right=del(p->right,key);

else{

if(p->left!=NULL&&p->right!=NULL)

{

succ=p->right;

while(succ->left!=NULL)

succ=succ->left;

p->data=succ->data;

p->right=del(p->right,succ->data);

}

else

{

temp=p;

if(p->left!=NULL)

p=p->left;

else if(p->right!=NULL)

p=p->right;

else

p=NULL;

free(temp);

}

}

return p;

}

void display(struct node \*root,int space)

{

int i;

space+=CNT;

if(root==NULL)

return;

display(root->right,space);

printf("\n");

for(i=CNT;i<space;i++)

printf(" ");

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

display(root->left,space);

}

int main()

{

struct node \*root=NULL;

int choice,i,data,key;

while(1)

{

printf("\n\nBINARY SEARCH TREE\n");

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

printf("1: INSERT A NODE\n");

printf("2: DELETE A NODE\n");

printf("3: SEARCH A NODE\n");

printf("4: DISPLAY THE TREE\n");

printf("\n\nENTER YOUR CHOICE: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("ENTER THE NODE DATA: ");

scanf("%d",&data);

root=insert(root,data);

break;

case 2:

printf("ENTER THE ELEMENT TO BE DELETED\n");

scanf("%d",&key);

root=del(root,key);

break;

case 3:

printf("ENTER A SEARCH KEY: ");

scanf("%d",&key);

search(root,key);

break;

case 4:

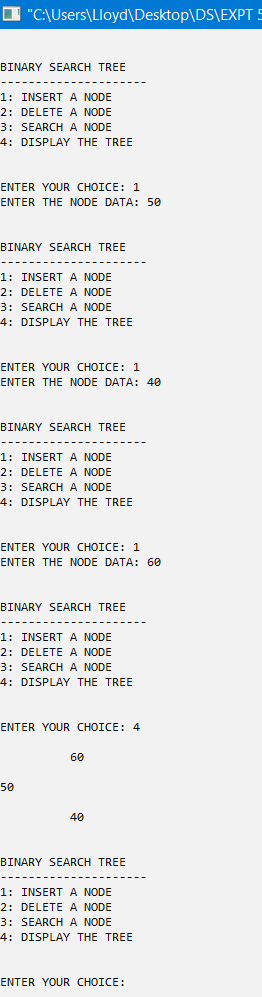
display(root,0);

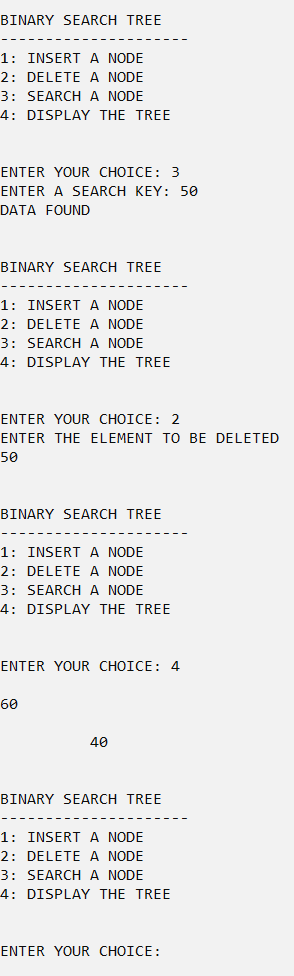
break;

}

}

}

**OUTPUT**

****

**2)**

#include<stdio.h>

#include<stdlib.h>

struct node{

struct node \*left;

int data;

struct node \*right;

};

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

{

if(p==NULL)

{

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

p->data=data;

p->left=NULL;

p->right=NULL;

}

else if(data < p->data)

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

else if(data > p->data)

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

else

printf("DUPLICATE VALUE\n");

return p;

}

void preorder(struct node \*p)

{

if(p==NULL)

return;

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

preorder(p->left);

preorder(p->right);

}

void inorder(struct node \*p)

{

if(p==NULL)

return;

inorder(p->left);

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

inorder(p->right);

}

void postorder(struct node \*p)

{

if(p==NULL)

return;

postorder(p->left);

postorder(p->right);

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

}

int height(struct node \*ptr)

{

int Hleft,Hright;

if(ptr==NULL)

return 0;

Hleft=height(ptr->left);

Hright=height(ptr->right);

if(Hleft > Hright)

return Hleft+1;

else

return Hright+1;

}

void displaygivenlevel(struct node \*ptr, int level)

{

if(ptr==NULL)

return;

if(level==1)

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

else if(level>1)

{

displaygivenlevel(ptr->left,level-1);

displaygivenlevel(ptr->right,level-1);

}

}

void levelorder(struct node \*ptr)

{

int h=height(ptr);

int i;

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

displaygivenlevel(ptr,i);

}

int main()

{

struct node \*root=NULL;

int choice,i,data;

while(1)

{

printf("\n\n1: ENTER A NODE\n");

printf("2: PREORDER TRAVERSAL\n");

printf("3: INORDER TRAVERSAL\n");

printf("4: POSTORDER TRAVERSAL\n");

printf("5: LEVELORDER TRAVERSAL\n");

printf("\n\nENTER YOUR CHOICE: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("ENTER THE NODE DATA: ");

scanf("%d",&data);

root=insert(root,data);

break;

case 2:

printf("PREORDER: ");

preorder(root);

break;

case 3:

printf("INORDER: ");

inorder(root);

break;

case 4:

printf("POSTORDER: ");

postorder(root);

break;

case 5:

printf("LEVEL ORDER: ");

levelorder(root);

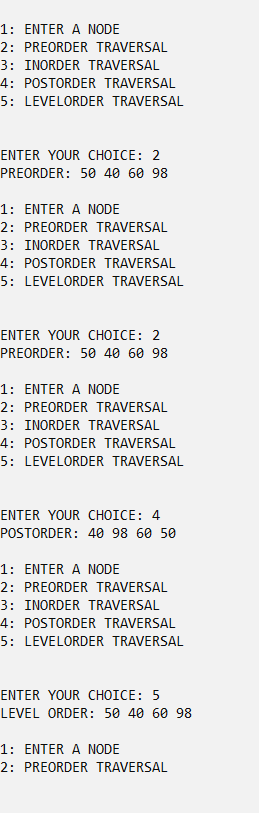
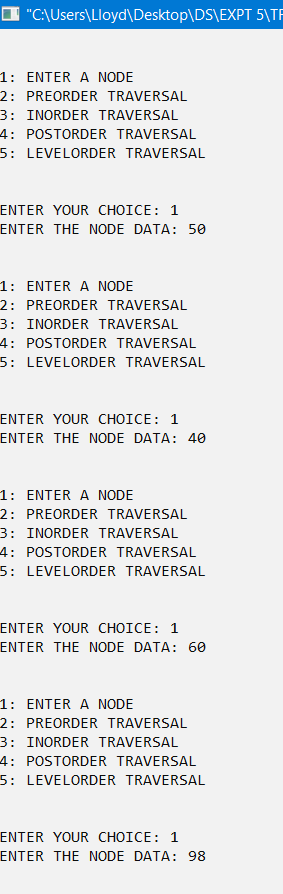
break;

}

}

}

**OUTPUT**

****

**3)**

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

struct node

{

struct node\* left;

int data;

struct node\* right;

};

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

int width(struct node\* root, int level);

int height(struct node\* root);

int main()

{

int n, i, h, data;

struct node\* root=NULL;

printf("ENTER NUMBER OF NODES: ");

scanf("%d",&n);

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

{

printf("ENTER THE DATA: ");

scanf("%d",&data);

root=insert(root, data);

}

h=height(root);

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

printf("\nLEVEL %d NUMBER OF NODES %d\n",i, width(root, i));

return 0;

}

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

{

if(p==NULL)

{

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

p->data=data;

p->left=NULL;

p->right=NULL;

}

else if(p->data > data)

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

else if(p->data < data)

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

else

printf("DUPLICATE\n");

return p;

}

int width(struct node\* root, int level)

{

if(root==NULL)

return 0;

else if(level==0)

return 1;

else

return width(root->left, level-1) + width(root->right, level-1);

}

int height(struct node\* root)

{

int lheight, rheight;

if(root==NULL)

return 0;

lheight=height(root->left);

rheight=height(root->right);

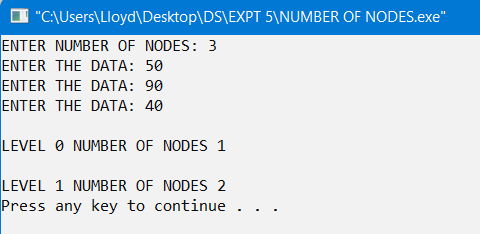
if(lheight > rheight)

return lheight+1;

else

return rheight+1;}

**OUTPUT**

****

**4)**

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

struct node

{

struct node\* left;

int data;

struct node\* right;

};

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

void levelnode(struct node\* root, int lv);

int height(struct node\* root);

int main()

{

int n, i, data, lv;

struct node\* root=NULL;

printf("ENTER NUMBER OF NODES: ");

scanf("%d",&n);

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

{

printf("ENTER THE DATA: ");

scanf("%d",&data);

root=insert(root, data);

}

while(1)

{

printf("\nENTER THE LEVEL (ENTER NEGATIVE NUMBER TO EXIT): ");

scanf("%d",&lv);

if(lv<0)

break;

levelnode(root, lv);

}

return 0;

}

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

{

if(p==NULL)

{

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

p->data=data;

p->left=NULL;

p->right=NULL;

}

else if(p->data > data)

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

else if(p->data < data)

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

else

printf("Duplicate data\n");

return p;

}

void levelnode(struct node\* root, int lv)

{

if(root==NULL)

return;

else if(lv==0)

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

else{

levelnode(root->left, lv-1);

levelnode(root->right, lv-1);

}

}

int height(struct node\* root)

{

int lheight, rheight;

if(root==NULL)

return 0;

lheight=height(root->left);

rheight=height(root->right);

if(lheight > rheight)

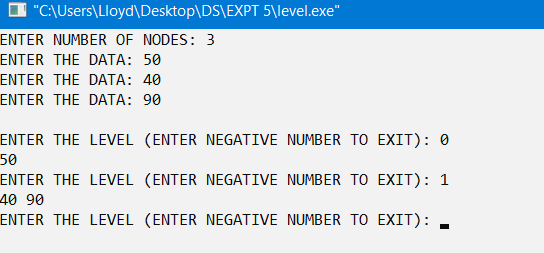
return lheight+1;

else

return rheight+1;

}

**OUTPUT**

****

**CONCLUSION**

The given problem statement was successfully compiled and executed.

**LEARNINGS AND FINDINGS**

This Experiment demonstrates:

1. Concept of Trees, Binary trees, Binary Search trees
2. Various Binary Tree traversals
3. Insertion and Deletion of element in BST
4. Advantages of BSTs.

Binary search trees offer lucrative advantages in terms of searching and organizing data. However, it faces issues such as improper balancing which may result into unnecessary increment of searching time.

|  |  |
| --- | --- |
| **SR. NO.** | **COMPILATION TIME** |
| 1 | 0.61 s |
| 2 | 0.31 s |
| 3 | 0.58 s |
| 4 | 0.63 s |