Assignment No: 8

Aim:

Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing;

i) Insert a new node
ii) Find no. of nodes in longest path
iii) Min. data value found in tree.
iv) Change a tree so that roles of left & right pointers are swapped at every node.
v) Search a value.

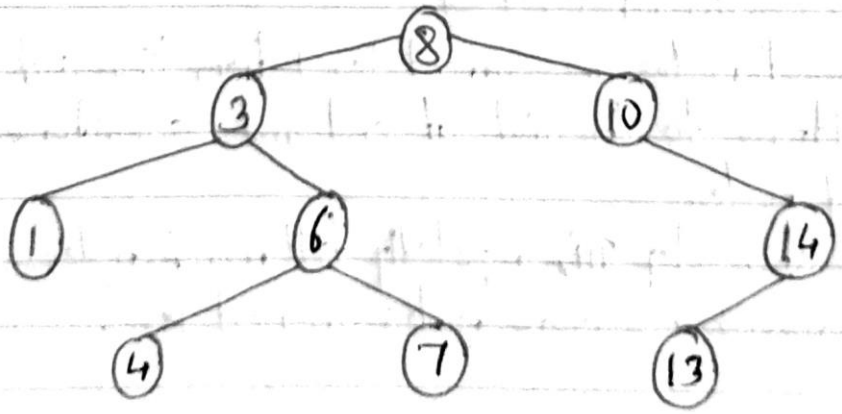
Theory:-

BST is a node base binary tree data which has →

left subtree of node contains only nodes with keys lesser than nodes key.

right subtree of node contains only nodes with keys greater than nodes key

left and right subtree each must also be a binary search tree



Algorithms ⇒

Create and Insert ⇒

(I) Access a random order of numbers from the user.
(II) There after, every number is compared with node. If less than or equal to data in the root node, proceed left of BST, else proceed right.
(III) Perform this till you reach a null pointer, place where present data is to be inserted.
(IV) Allocate a new node and assign data in this node and allocate pointers.

Height of BST ⇒

(I) This algorithm is based on the idea of storing the nodes of every level of BST in a dynamic queue. It is also useful to print tree level.
(II) Initialize the contents of list with root of BST. The counter no-of-node-in-current-level = 1.
(III) Access no-of-nodes-in-current-level from link list and add all their childrens to the list at end and simultaneously keep track of no. of nodes.
(IV) Continue above step (III) repetedly till no-of-nodes-in-current-level is 0, which means no more nodes in next level.

Leaf Node of BST ⇒

(I) There are many algorithms to find leaf nodes of BST. The one considered here is based on idea that one could do simple inorder traversal of BST and just before printing, the data as one normally does an inorder traversal and right nodes to NULL.

(II) Inorder: The recursive function will receive root of tree from where inorder traversal is to initialized algorithm is to proceed left, which in this case is to call same function; print the data if both left and right pointers are NULL.

(III) Thus, all the leaf nodes of BST are printed.

Mirror of Tree ⇒

(I) Following is a algorithm of recursive function to find mirror of tree. The function mirror-Tree accepts a pointer to tree.

(II) The function begains by checking if pointer passed is not NULL If not, allocates a new node. Assign the data of original node to be copied node. Assign left child of the new node by calling function mirror-Tree, with right child of original node and assign the right child of new node by calling function mirror-Tree NULL returned by function.

Level-wise printing ⇒

(I) This algorithm is based on idea of storing the nodes of every level of BST in dynamic queue.

(II) Initialize the contents of list with root of BST. Counter no-of-nodes-in-current-level = 1.

(III) Access no-of-nodes-in-current-level from linked list, print the level no. and all data of all nodes of current level and at end keep track of number of nodes accessed in next level in variable which at end is assigned back to no-of-nodes-in-current-level.

(IV) Continue step (III) repetedly till no-of-nodes-in-current-level is 0, which means no more nodes in next level.

Test Conditions ⇒

For e.g ⇒ Enter: 34, 12, 56, 6, 14, 40, 70
Height of BST is 3
Leaf node are 6, 14, 40, 70

For mirror ⇒ Enter, 34, 12, 56, 6, 14, 40, 70
Level wise printing of original tree

Level 1: 34
Level 2: 12, 56
Level 3: 6, 14, 40, 70

Level wise printing of mirror tree

Level 1: 34
Level 2: 56, 12
Level 3: 30, 40, 14, 6

Input ⇒

Enter data (nos to be stored in binary search tree). Every node in BST would contain 3 fields: data, left child pointer and right child pointer.

Output ⇒

The height of tree and the list of its leaf node. The original and mirror image printed levelwise.

Program Code :-

```
#include <iostream>
using namespace std;
struct node
{
   int data;
   node *L;
   node *R;
};
node *root, *temp;
int count, key;

class bst
{
public:
   void create();
   void insert(node *, node *);
   void disin(node *);
   void dispre(node *);
   void dispost(node *);
   void search(node *, int);
   int height(node *);
   void mirror(node *);
   void min(node *);
   bst()
   {
      root = NULL;
      count = 0;
   }
```

```
};

void bst::create()
{
   char ans;
   do
   {
      temp = new node;
      cout << "Enter the data : ";
      cin >> temp->data;
      temp->L = NULL;
      temp->R = NULL;
      if (root == NULL)
      {
         root = temp;
      }
      else
         insert(root, temp);
      cout << "Do you want to insert more value :(y/n) : " << endl;
      cin >> ans;
      count++;
      cout << endl;
   } while (ans == 'y');
   cout << "The Total no.of nodes are : " << count;
}
void bst::insert(node *root, node *temp)
{
   if (temp->data > root->data)
    {
      if (root->R == NULL)
```

```
        {
            root->R = temp;
        }
        else
            insert(root->R, temp);
    }
    else
    {
        if (root->L == NULL)
        {
            root->L = temp;
        }
        else
            insert(root->L, temp);
    }
}

void bst::disin(node *root)
{
    if (root != NULL)
    {
        disin(root->L);
        cout << root->data << "\t";
        disin(root->R);
        count++;
    }
}
void bst::dispre(node *root)
{
    if (root != NULL)
```

```
    {
        cout << root->data << "\t";
        dispre(root->L);
        dispre(root->R);
    }
}
void bst::dispost(node *root)
{
    if (root != NULL)
    {
        dispost(root->L);
        dispost(root->R);
        cout << root->data << "\t";
    }
}
void bst::search(node *root, int key)
{
    int flag = 0;
    cout << "\nEnter your key : " << endl;
    cin >> key;
    temp = root;
    while (temp != NULL)
    {
        if (key == temp->data)
        {
            cout << "KEY FOUND\n";
            flag = 1;
            break;
        }
        node *parent = temp;
```

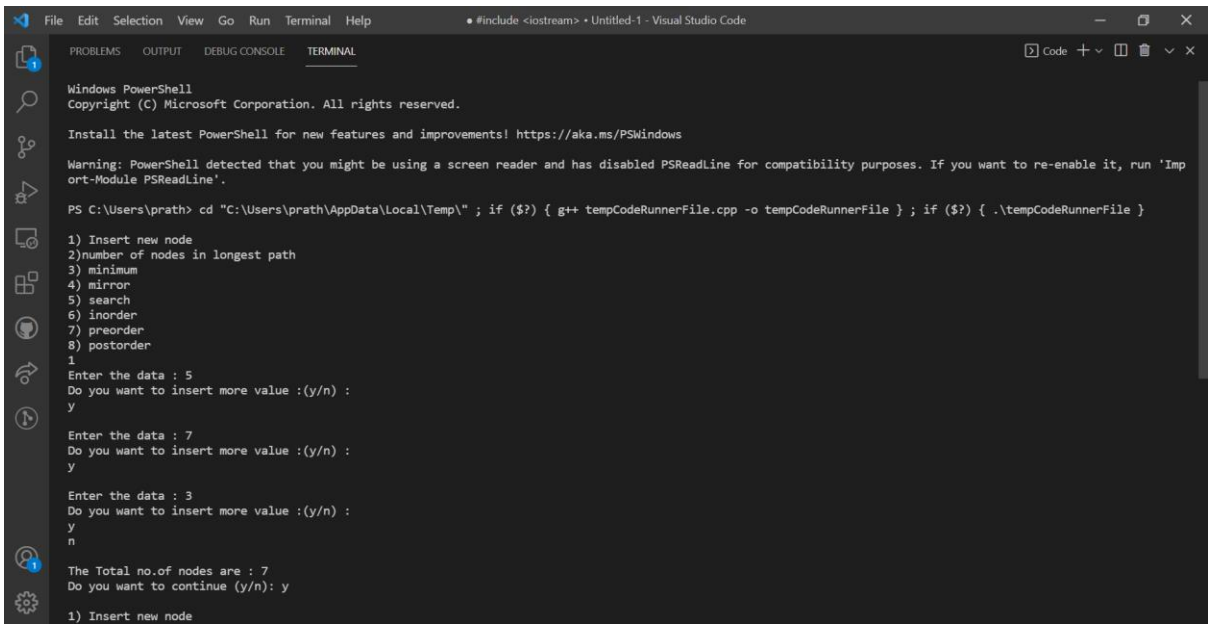
```
        if (key > parent->data)
        {
            temp = temp->R;
        }
        else
        {
            temp = temp->L;
        }
    }
    if (flag == 0)
    {
        cout << "KEY NOT FOUND " << endl;
    }
}
int bst::height(node *root)
{
    int hl, hr;
    if (root == NULL)
    {
        return 0;
    }
    else if (root->L == NULL && root->R == NULL)
    {
        return 0;
    }
    cout << endl;
    hr = height(root->R);
    hl = height(root->L);
    if (hr > hl)
    {
```

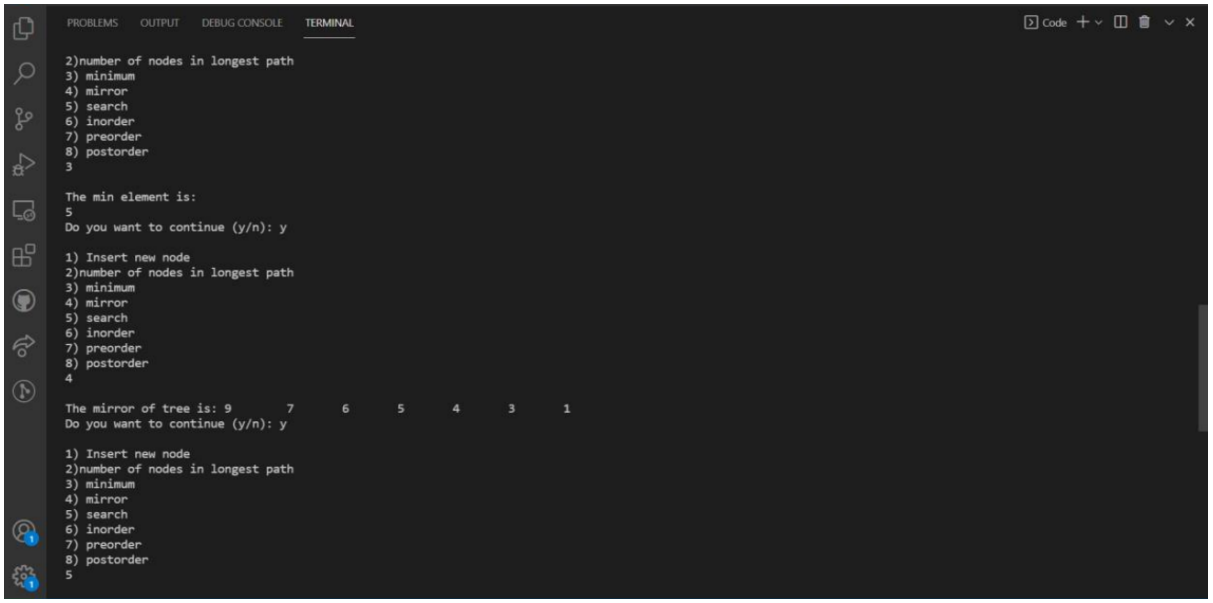
```
        return (1 + hr);
    }
    else
    {
        return (1 + hl);
    }
}

void bst::min(node *root)
{
    temp = root;
    cout << endl;
    while (temp->L != NULL)
    {
        temp = temp->L;
    }
    cout << root->data;
}

void bst::mirror(node *root)
{
    temp = root;
    if (root != NULL)
    {
        mirror(root->L);
        mirror(root->R);
        temp = root->L;
        root->L = root->R;
        root->R = temp;
    }
```

```
}
int main()
{

    bst t;
    int ch;
    char ans;
    do
    {
        cout << "\n1) Insert new node\n2)number of nodes in longest path\n3) minimum\n4) mirror\n5) search\n6) inorder\n7) preorder\n8) postorder" << endl;
        cin >> ch;
        switch (ch)
        {
        case 1:
            t.create();
            break;
        case 2:
            cout << "\n Number of nodes in longest path: " << (1 + (t.height(root)));
            break;
        case 3:
            cout << "\nThe min element is: ";
            t.min(root);
            break;
        case 4:
            t.mirror(root);
            cout << "\nThe mirror of tree is: ";
            t.disin(root);
            break;
        case 5:
            t.search(root, key);
```

```
            break;
        case 6:
            cout << "\n***************INORDER**************" << endl;
            t.disin(root);
            break;
        case 7:
            cout << "\n***************PREORDER**************" << endl;
            t.dispre(root);
            break;
        case 8:
            cout << "\n*******************POSTORDER**************" << endl;
            t.dispost(root);
            break;
        }
        cout << "\nDo you want to continue (y/n): ";
        cin >> ans;
    } while (ans == 'y');
    return 0;
}
```

Program Output: -




PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Code
2)number of nodes in longest path
3) minimum
4) mirror
5) search
6) inorder
7) preorder
8) postorder
3
The min element is:
5
Do you want to continue (y/n): y
1) Insert new node
2)number of nodes in longest path
3) minimum
4) mirror
5) search
6) inorder
7) preorder
8) postorder
4
The mirror of tree is: 9 7 6 5 4 3 1
Do you want to continue (y/n): y
1) Insert new node
2)number of nodes in longest path
3) minimum
4) mirror
5) search
6) inorder
7) preorder
8) postorder
5


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Code
The mirror of tree is: 9 7 6 5 4 3 1
Do you want to continue (y/n): y
1) Insert new node
2)number of nodes in longest path
3) minimum
4) mirror
5) search
6) inorder
7) preorder
8) postorder
5
Enter your key :
6
KEY NOT FOUND
Do you want to continue (y/n): y
1) Insert new node
2)number of nodes in longest path
3) minimum
4) mirror
5) search
6) inorder
7) preorder
8) postorder
6
**************INORDER*************
9 7 6 5 4 3 1
Do you want to continue (y/n): y
1) Insert new node
2)number of nodes in longest path
3) minimum


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Code
7) preorder
8) postorder
6
**************INORDER*************
9 7 6 5 4 3 1
Do you want to continue (y/n): y
1) Insert new node
2)number of nodes in longest path
3) minimum
4) mirror
5) search
6) inorder
7) preorder
8) postorder
7
**************PREORDER*************
5 7 9 6 3 4 1
Do you want to continue (y/n): y
1) Insert new node
2)number of nodes in longest path
3) minimum
4) mirror
5) search
6) inorder
7) preorder
8) postorder
8
*****************POSTORDER*************
9 6 7 4 1 3 5
Do you want to continue (y/n): n
PS C:\Users\prath\vs code data\DSA>