Page No.:
Date:

Experiment no - 8

Aim: Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree -
i. Insert new node
ii. Find no. of nodes in longest path
iii. Minimum data value found in the tree
iv. Change a tree so that the roles of the left & right pointers are swapped at every node
v. Search a value

Theory:
Binary Search Tree is 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.

Algorithms:

Create and Insert:
① Accept a random order of numbers from the user. The first no. would be inserted at the root node.
② Thereafter, every no. is compared with the root node. If less than or equal to the data in the root node, proceed left of the BST, else proceed right.
③ Perform this till you reach a null pointer, the place where the present data is to inserted
④ Allocate a new node and assign the data in this node and allocate pointers appropriately.

Page No.:
Date:

Height of BST:

① This algorithm is based on the idea of storing the nodes of every level of the BST in a dynamic queue (link list). It is also simultaneously useful to point the tree level wise. The total no. of levels accessed would be the height of the tree.

② Initialize the contents of the list with the root of the BST. The counter no_of_nodes_in_current_level = 1 and the level_accessed = 1.

③ Access no_of_nodes_in_current_level from the link list and add all their children to the list at the end and simultaneously keep track of the no of nodes accessed in the next level in a variables which at the end is assigned back to no_of_nodes_in_current_level. Also increment level_accessed, indicating one more level accessed in the BST.

④ Continue step 3 repeatedly till no_of_nodes_in_current_level is 0, which means no more nodes in the next level. The value stored in the variable level_accessed is the height of the BST.

Leaf Nodes of BST:

① There are many algorithms to find the leaf nodes of an BST. The one considered here is based on the idea that one could do a simple inorder traversal of the BST & just before printing the data as one normally does in an inorder traversal, check if both the left and right nodes are NULL. If so, it means the node under consideration is a leaf node and must be printed.

② Inorder: The recursive function will receive the root of the tree (or subtree) from where inorder traversal is to be initiated. Algorithm is to proceed left, which in this cases is to call the same function with the left child of the node, print the data if both left & right pointed are NULL & then proceeds right, which in this case is to call the same function with the right child of the ~~root~~ node.

③ Thus all the leaf nodes of the BST are printed

Page No.:
Date:

Mirror of Tree:

1. Following is a algorithm of a recursive function to find mirror of the tree. The function mirror-tree accepts a pointer to a tree as the parameter. Initially the root node is passed later the roots of the subsequent subtrees are passed as the parameter.
2. The function begins by checking if the pointer passed is not NULL. If not, allocated a new node. Assigns the data of the original node to the copied node. Assigns the left child of the new node by calling the function mirror_Tree with the right child of the original node & assigns the right child of the new node by calling the function mirror_Tree with the left child of the original node. If the pointer passed is NULL, NULL is returned by the function else the new node created in returned.

Level wise printing:

1. This algorithm is based on the idea of storing the nodes of every level of the BST in a dynamic queue (link list).
2. Initialize the contents of the list with the root of the BST. The counter no_of_nodes_in_current_level = 1 and the level_accessed = 1
3. Access no_of_nodes_in_current_level from the link list. Print the level Number and all the data of all the nodes of the current level. Simultaneously add all their children to the list at the end and keep track of the no. of nodes accessed in the next level in a variable which at the end is assigned back to no_of_nodes_in_current_level. Also increment level_accessed, indicating one more level accessed in the BST
4. Continue step 3 repeatedly till no_of_nodes_in_current_level is 0, which means no more nodes in the next level.

Test conditions:

Simple input of random no. Display the height of the tree & the leaf nodes. The BST entered could be drawn on the rough page and one could check if the height calculated and the leaf nodes printed are correct.

Page No.:
Date:

For eg. Enter : 34, 12, 56, 6, 14, 40, 70.
The height of the BST is 3
The leaf nodes are 6, 14, 40 & 70.

For Mirror Image:

Enter : 34, 12, 56, 6, 14, 40, 70

Level wise printed of original tree.

Level1 : 34
Level2 : 12, 56
level3 : 6, 14, 40, 70

Levelwise printed of the mirror tree
level1 : 34
level2 : 56, 12
level 3 : 70, 40, 14, 6

Input :
Enter data (no.) to be stored in the binary search tree. Every node in the BST would contain 3 fields:
data, left child pointer and right child pointer.

Output:
The height of the tree and the list of its leaf nodes.
The original & mirror image printed levelwise

Program –

```cpp
#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 : " << endl;
        cin >> ans;
        count++;
        cout << endl;
    } while (ans == 'y');
    cout << "The Total no.of nodes are:" << count;
}
void bst::insert(node *root, node *temp)
```

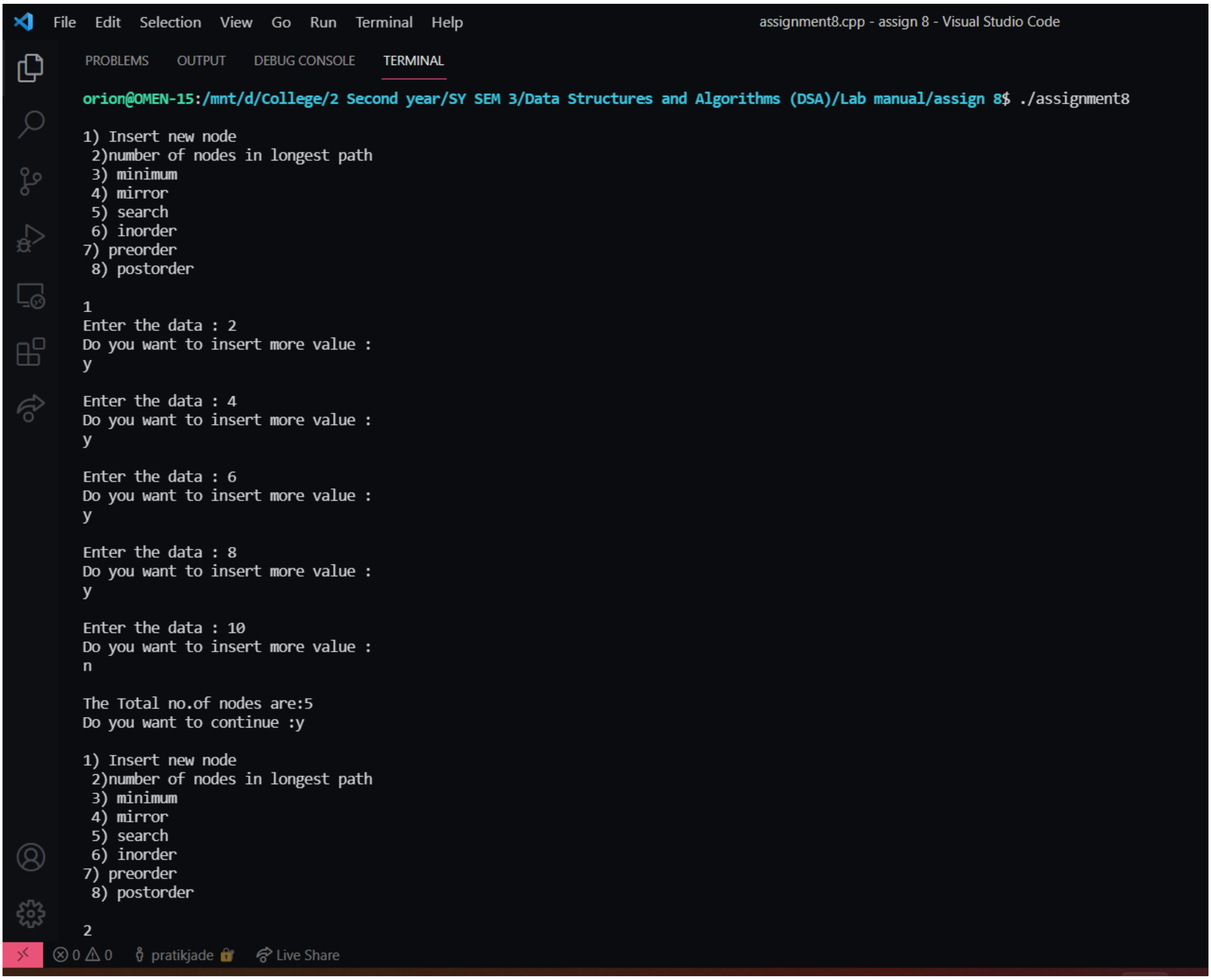
```
{
    if (temp->data > root->data)
    {
        if (root->R == NULL)
        {
            root->R = temp;
        }
        else
            insert(root->R, temp);
    }
    else
    {
        if (root->L == NULL)
        {
            root->L = temp;
        }
        else
            insert(root->L,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

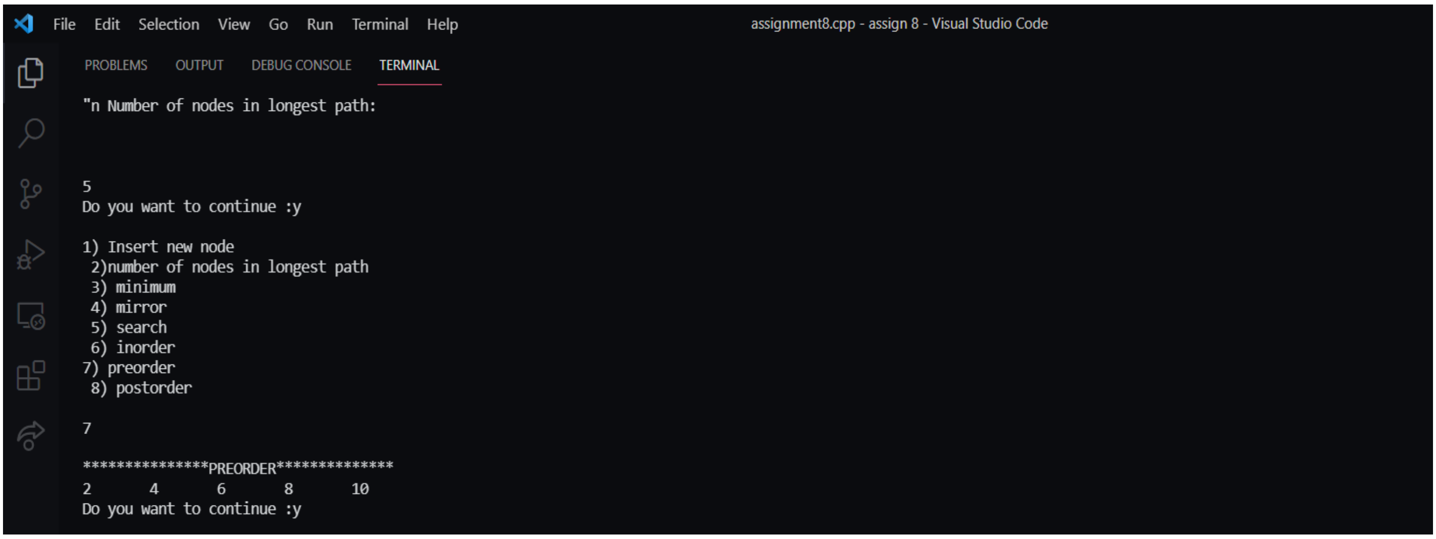
```
    {
        if (key == temp->data)
        {
            cout << "KEY FOUND\n";
            flag = 1;
            break;
        }
        node *parent = temp;
        if (key > parent->data)
        {
            temp = temp->R;
        }
        else
        {
            temp = temp->L;
        }
    }
    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;
}
```

```cpp
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\n 2)number of nodes in longest path\n 3) minimum\n 4)
mirror\n 5) search\n 6) inorder \n7) preorder\n 8) postorder\n" << 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 :";
        cin >> ans;
    } while (ans == 'y');
    return 0;
}
```

Output-





File Edit Selection View Go Run Terminal Help   assignment8.cpp - assign 8 - Visual Studio Code

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Do you want to continue :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**************
10      8       6       4       2
Do you want to continue :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: 10       8       6       4       2
Do you want to continue :y

1) Insert new node
 2)number of nodes in longest path
 3) minimum
 4) mirror
 5) search
 6) inorder
7) preorder
 8) postorder

3

The min element is:
2
Do you want to continue :
```

⊗ 0 ⚠ 0 pratikjade Live Share