**V Venkateswarlu Gupta**

**1602-19-733-119**

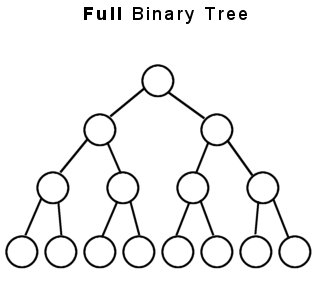
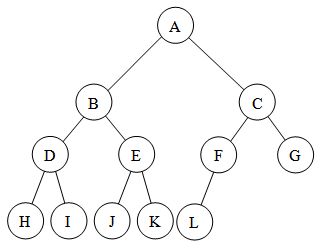
**CSE-B**

**Data Structures Lab**

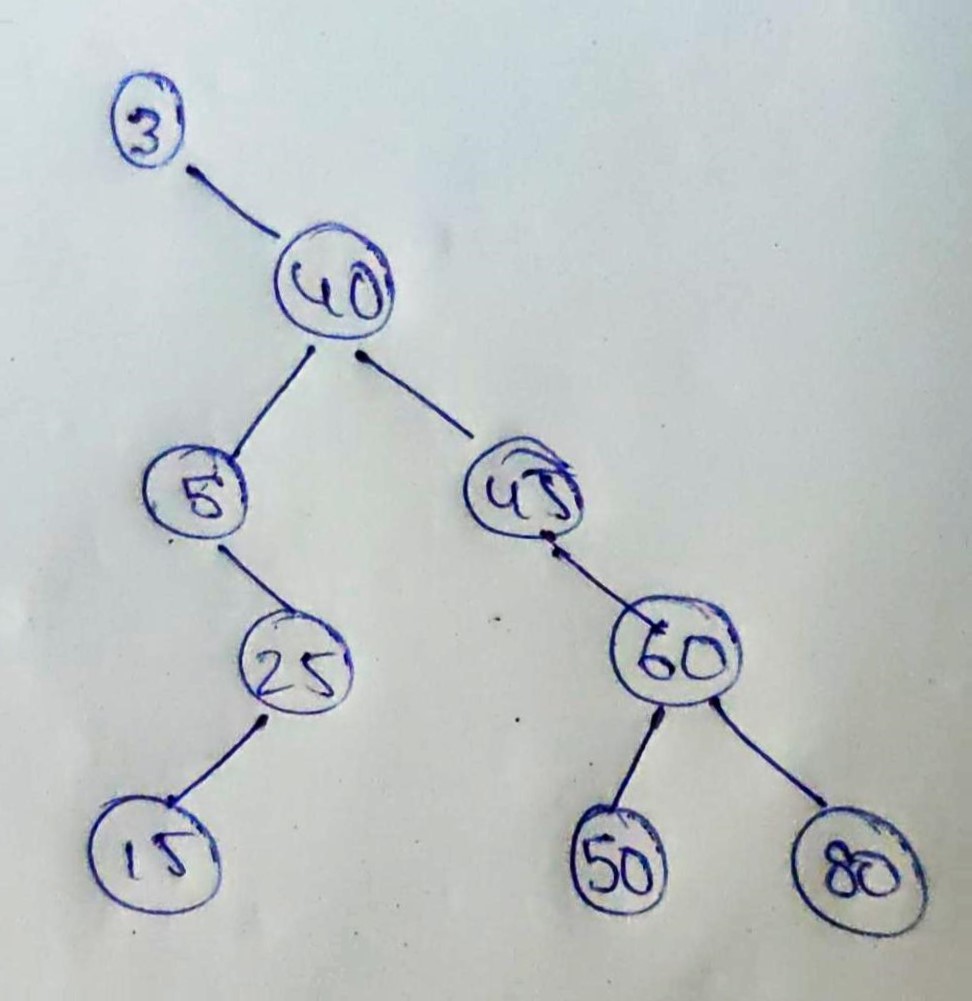
Week – 8

***Prelab Questions***

1. **Differentiate between Complete BT and Full BT. Give Examples.**

* A full binary tree (sometimes proper binary tree or 2-tree) is a tree in which every node other than the leaves has two children
  + [](https://web.cecs.pdx.edu/~sheard/course/Cs163/Doc/FullvsComplete.html)
* A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible
  + 

**2. Construct a BST with the following data 3, 40, 5, 25, 45, 60, 80, 15, 50**



**3. What are the different cases to delete a node in a binary search tree?**

* If the node to be deleted is a **leaf node** then, parent of that node will point to null
* If the node to be deleted **has one child node**, then child node will become a child node of the parent node
* If the node to be deleted **has two children** then, we find the node(minNode) with minimum value from the right subtree of that current node. The current node will be replaced by its successor(minNode).

**4. Write the Preorder, Inorder and Postorder traversal for the binary search tree constructed in question 2.**

Preorder : 3 40 5 25 15 45 60 50 80

InOrder: 3 5 15 25 40 45 50 60 80

PostOrder : 15 25 5 50 80 60 45 40 3

**5. Write the pseudocode to find the maximum element in the left subtree of a given BST.**

int maxValue(BST \*root)

{

     BST \*current=node;

     while(current->rightChild!=NULL)

         current=current->rightChild;

     return current->data;

}

maxValue(root->leftChild);

**6. Write the pseudo code to perform inorder traversal of a BST using non recursive procedure.**

1. Create an empty stack S.
2. Initialize current node as root
3. Push the current node to S and set current = current->left until current is NULL
4. If current is NULL and stack is not empty then
   1. Pop the top item from stack.
   2. Print the popped item, set current = popped\_item->right
   3. Go to step 3.
5. If current is NULL and stack is empty then we are done.

***Prelab Programs***

***TreeOperations.h***

typedef *struct* node{

*int* data;

*struct* node \*leftChild,\*rightChild;

}Tree;

Tree \*createNode(*int* *value*){

    Tree \*newNode;

    newNode=(Tree \*)malloc(sizeof(Tree));

    newNode->leftChild=NULL;

    newNode->data=value;

    newNode->rightChild=NULL;

    return newNode;

}

*void* preOrderTraversal(Tree \**root*){

    if(root!=NULL){

        printf(" %d ",root->data);

        preOrderTraversal(root->leftChild);

        preOrderTraversal(root->rightChild);

    }

}

*void* postOrderTraversal(Tree \**root*){

    if(root!=NULL){

        postOrderTraversal(root->leftChild);

        postOrderTraversal(root->rightChild);

        printf(" %d ",root->data);

    }

}

*void* inOrderTraversal(Tree \**root*){

    if(root!=NULL){

        inOrderTraversal(root->leftChild);

        printf(" %d ",root->data);

        inOrderTraversal(root->rightChild);

    }

}

*int* front=-1;

*int* rear=-1;

#define sz 50

Tree \*Q[sz];

*void* push(Tree \**x*){

    if(rear==sz-1){

        //Full

    }else if(front==-1 && rear==-1){

        front++;

    }

    Q[++rear]=x;

}

Tree \*pop(){

    Tree \*x;

    if(front==-1 && rear==-1){

        //Empty

    }else if(front==rear){

        front=-1;

        rear=-1;

    }else{

        x=Q[front];

        front++;

        return x;

    }

}

*void* levelOrder(Tree \**root*)

{

    if (root == NULL) return;

    Tree \*current;

    push(root);

    push(NULL); //first iteration : root->data null

    //     1

    // 2      3

    while(rear-front>0) {

        current=pop();

        if (current == NULL) {

           push(NULL);

           printf(" \n");

        } else {

            if(current->leftChild)

            push(current->leftChild);

            if(current->rightChild)

            push(current->rightChild);

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

        }

    }

}

*int* isLeaf(Tree \**node*){

    if(node->leftChild==NULL&&node->rightChild==NULL){

        return 1;

    }

    return 0;

}

*int* max(*int* *a*, *int* *b*){

    return (a>b) ? a:b;

}

*int* height(Tree \**node*){

    if(node==NULL||isLeaf(node)){

        return 0;

    }

    return max(height(node->leftChild),height(node->rightChild))+1;

}

Tree \*CommonAncestor(Tree \**root*, Tree \**p*, Tree \**q*){

    if(root==NULL)          return NULL;

    if(root==p||root==q)    return root;

    Tree \*l=CommonAncestor(root->leftChild,p,q);

    Tree \*r=CommonAncestor(root->rightChild,p,q);

    if(l&&r) return root;

    return l?l:r;

}

*void* deletion(Tree \**root*, *int* *key*){

    if(root==NULL){

        return;

    }else if(root->data==key){

            Tree \*t=root;

        if(root->leftChild==NULL&&root->rightChild==NULL){

            root=NULL;

        }else if(root->leftChild==NULL){

            root=root->rightChild;

        }else{

            root=root->leftChild;

        }

        free(t);

        return;

    }else if(root->leftChild->data==key){

        Tree \*t=root->leftChild;

        if(root->leftChild->leftChild==NULL&&root->leftChild->rightChild==NULL){

            root->leftChild=NULL;

        }else if(root->leftChild->leftChild==NULL){

            root->rightChild=root->leftChild->rightChild;

        }else if(root->leftChild->rightChild==NULL){

            root->leftChild=root->rightChild->leftChild;

        }

        free(t);

        return;

    }

    else if(root->rightChild->data==key){

        Tree \*t=root->rightChild;

        if(root->rightChild->leftChild==NULL&&root->rightChild->rightChild==NULL){

            root->rightChild=NULL;

        }else if(root->rightChild->leftChild==NULL){

            root->rightChild=root->rightChild->rightChild;

        }else{

            root->rightChild=root->rightChild->leftChild;

        }

        free(t);

        return;

    }

    if(root->data>key){

        deletion(root->leftChild,key);

    }else{

        deletion(root->rightChild,key);

    }

}

**Write a C program to implement a Binary Tree.**

#include<stdio.h>

*char* BT[100];

*void* root(*char* *data*){

    if(BT[0]!='\0'){

        printf("Root exists\n");

        printf("Do you want to modify?\n# - Yes\t\t0 -  No\n");//# any non zero number

*int* doModify;

        scanf("%d",&doModify);

        if(doModify){

            BT[0]=*data*;

        }

    }else{

        BT[0]=*data*;

    }

}

*void* leftChild(*char* *data*, *int* *parent*){

    if(BT[*parent*]=='\0'){

        printf("Parent doesn\'t exists\n");

    }else{

        BT[*parent*\*2+1]=*data*;

    }

}

*void* rightChild(*char* *data*, *int* *parent*){

    if(BT[*parent*]=='\0'){

        printf("Parent doesn\'t exists\n");

    }else{

        BT[*parent*\*2+2]=*data*;

    }

}

*void* displayBT(){

    for(*int* i=0;i<32;i++){

        if(BT[i]!='\0'){

            printf(" %c ",BT[i]);

        }else{

            printf(" - ");

        }

    }

}

*void* isPresent(*int* *index*){

    if(BT[*index*]){

        printf("Present at %d\tElement is : %c\n",*index*,BT[*index*]);

    }else{

        printf("Not Present at %d\n",*index*);

    }

}

*void* main(){

    //Present at 1    Element is : B

    //Not Present at 7

    // A  B  C  D  E  F  G  -  -  -  -  H  I  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -

    root('A');

    leftChild('B',0);

    rightChild('C',0);

    leftChild('D',1);

    rightChild('E',1);

    leftChild('F',2);

    rightChild('G',2);

    leftChild('H',5);

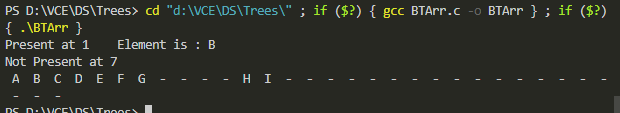
    rightChild('I',5);

    isPresent(1);

    isPresent(7);

    displayBT();

}



**Write a C program to implement TreeTraversals (Level order, Inorder, Preorder, Postorder)**

#include<stdio.h>

#include<stdlib.h>

#include"TreeOperations.h"

#define BT Tree

*void* main(){

    BT \*root;

    root=createNode(1);

    root->leftChild=createNode(2);

    root->rightChild=createNode(3);

    root->rightChild->leftChild=createNode(4);

    root->rightChild->rightChild=createNode(5);

    preOrderTraversal(root);

    printf("\n");

    inOrderTraversal(root);

    printf("\n");

    postOrderTraversal(root);

    printf("\n");

    if(isLeaf(root->leftChild)){

        printf("Yes\n");

    }else{

        printf("No\n");

    }

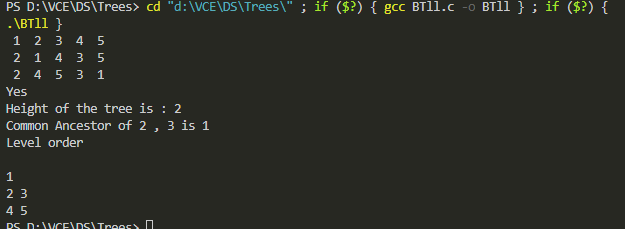
    printf("Height of the tree is : %d\n",height(root));

    printf("Common Ancestor of 2 , 3 is %d\n",CommonAncestor(root, root->rightChild->leftChild, root->leftChild)->data);

printf("Level order\n\n");

    levelOrder(root);

}



**Write a C program to implement insert operation on Binary search tree.**

#include<stdio.h>

#include<stdlib.h>

#include"TreeOperations.h"

#define BST Tree

BST \*insertNode(BST \**root*, *int* *data*){

    BST \*parent;

    BST \*current;

    if(*root*==NULL){

*root*=createNode(*data*);

    }else{

        current=*root*;

        while(1){

            parent=current;

            if(*data*<current->data){

                current=current->leftChild;

                if(current==NULL){

                    parent->leftChild=createNode(*data*);

                    break;

                }

            }else{

                current=current->rightChild;

                if(current==NULL){

                    parent->rightChild=createNode(*data*);

                    break;

                }

            }

        }

    }

    return *root*;

}

BST \*createBST(BST \**root*){

*int* i,n,data;

    printf("No of values in BST : ");

    scanf("%d",&n);

    for(i=0;i<n;i++){

        scanf("%d",&data);

*root*=insertNode(*root*, data);

    }

    return *root*;

}

*void* main(){

    // 3 40 5 25 45 60 80 15 50

    BST \*root=NULL;

    root=createBST(root);

    preOrderTraversal(root);

    printf("\n");

    inOrderTraversal(root);

    printf("\n");

    postOrderTraversal(root);

    printf("\n\n\n\n");

    levelOrder(root);

}

