**LAB SESSION 6: BINARY TREES**

**AIM:** To implement Binary trees and perform the listed operations on such trees.

**PROBLEM DEFINITION:**

Develop a C program to create a binary tree given its Inorder and Postorder traversal.

Provide options to the user to perform the following operations on the binary tree:

1. Display the height of the tree

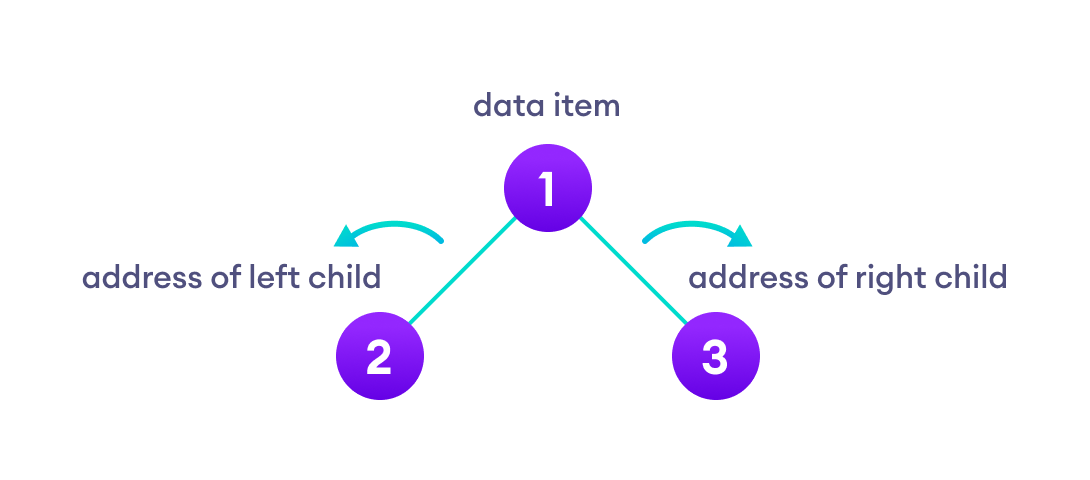
2. Return the depth of a given node in a tree

3. Perform Level order traversal

**THEORY:**

Binary Tree is a tree data structure where each node has at most 2 children. Each node of a binary tree consists of three items:

* data item



* address of left child
* address of right child

**Types of Binary Trees**

**1. Full Binary Tree**



A full Binary tree is a special type of binary tree

in which every parent node/internal node

has either two or no children.

**2. Perfect Binary Tree**



A perfect binary tree is a type of binary tree

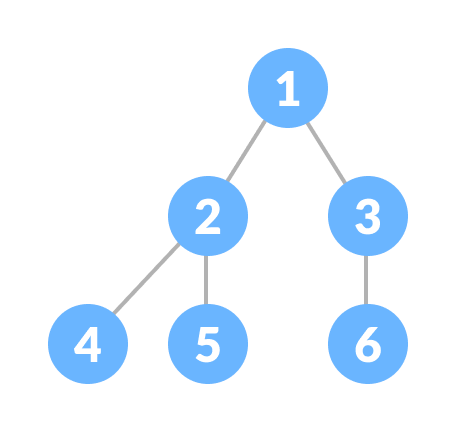
in which every internal node has exactly

two child nodes and all the leaf nodes

are at the same level.

**3. Complete Binary Tree**

A complete binary tree is just like a full binary tree,



but with two major differences:

* Every level must be completely filled
* All the leaf elements must lean towards the left.
* The last leaf element might not have a right

sibling i.e. a complete binary tree doesn't

have to be a full binary tree.

**Tree Traversals**

* 1. Inorder Traversal
* Traverse the left subtree
* Visit the root.
* Traverse the right subtree
  1. Preorder Traversal
* Visit the root.
* Traverse the left subtree
* Traverse the right subtree
  1. Postorder Traversal
* Traverse the left subtree
* Traverse the right subtree
* Visit the root

d. Level Order Traversal

All nodes present in the same level are traversed before traversing the next level.

**ALGORITHMS**

1. Level Order Traversal

* Create an empty queue.
* Enqueue the root node.
* While the queue is not empty, do the following:
  + Dequeue a node from the queue and print its value.
  + Enqueue the left child of the dequeued node if it exists.
  + Enqueue the right child of the dequeued node if it exists.

1. Finding the depth of a node

* Initialize two variables dpt and found to -1 and 0, respectively.
* If the tree is empty, return dpt.
* Increment dpt by 1.
* If the current node is the target node, set found to 1 and return dpt.
* Recursively find the target node in the left subtree.
* If the target node is found in the left subtree, return dpt.
* Recursively find the target node in the right subtree.
* If the target node is found in the right subtree, return dpt.
* If the target node is not found in either subtree, decrement dpt by 1 and return dpt.

**PROGRAM AND OUTPUT:**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

struct treenode{

    struct treenode \* lchild;

    int info;

    struct treenode \* rchild;

};

struct listnode{

    struct listnode\* next;

    int info;

};

struct treenode \*queue[MAX];

int front = - 1, rear = - 1;

struct listnode\* creatlist(int num){

    if(num == 0)

        return NULL;

    struct listnode\* tmp = (struct listnode\*)malloc(sizeof(struct listnode));

    int info;

    printf("Enter element: ");

    scanf("%d",&info);

    tmp->info = info;

    tmp->next = creatlist(--num);

    return tmp;

}

void display\_list(struct listnode\* p){

    if(p == NULL)

        return;

    printf("%d\n",p->info);

    display\_list(p->next);

}

struct treenode\* construct\_in\_post(struct listnode\* inptr, struct listnode\* postptr, int num) {

    struct treenode\* temp;

    struct listnode\* q, \* ptr;

    int i,j;

    if (num <= 0)

        return NULL;

    ptr = postptr;

    for (i = 1; i < num; i++)

        ptr = ptr->next;

    temp = (struct treenode\*)malloc(sizeof(struct treenode));

    temp->info = ptr->info;

    temp->lchild = NULL;

    temp->rchild = NULL;

    if (num == 1)

        return temp;

    q = inptr;

    for (i = 0; q->info != ptr->info; i++)

        q = q->next;

    temp->lchild = construct\_in\_post(inptr, postptr, i);

    for(j=1; j<=i;j++)

        postptr=postptr->next;

    temp->rchild = construct\_in\_post(q->next, postptr, num - i - 1);

    return temp;

}

void post\_traversal(struct treenode \*p){

    if(p == NULL)

        return;

    post\_traversal(p->lchild);

    post\_traversal(p->rchild);

    printf("%d ",p->info);

}

int height(struct treenode\* p){

    int lh, rh;

    if(p == NULL)

        return 0;

    lh = 1 + height(p->lchild);

    rh = 1 + height(p->rchild);

    if(lh > rh)

        return lh;

    else

        return rh;

}

int depth(struct treenode \*p, int key){

    static int dpt = -1;

    static int found = 0;

    if(p == NULL)

        return dpt;

    dpt ++;

    if(p->info == key){

        found = 1;

        return dpt;

    }

    depth(p->lchild,key);

    if(found)

        return dpt;

    depth(p->rchild,key);

    if(found)

        return dpt;

    else

        dpt --;

    return dpt;

}

void insert\_queue(struct treenode \*item)

{

    if (rear == MAX - 1)

    {

        printf ("Queue Overflow.......\n");

        return;

    }

    if (front == - 1)

        front = 0;

    rear = rear + 1;

    queue[rear] = item;

}

struct node \* del\_queue()

{

    struct treenode \*item;

    if (front == - 1|| front == rear + 1)

    {

        printf ("Queue underflow.......\n");

        exit(1);

    }

    item = queue[front];

    front = front + 1;

    return item;

}

int queue\_empty ()

{

    if (front == - 1|| front == rear + 1)

        return 1;

    else

        return 0;

}

void level\_trav(struct node \*root)

{

    struct treenode \*ptr = root ;

    if (ptr == NULL)

    {

        printf ("Tree is empty\n");

        return;

    }

    insert\_queue(ptr);

    while(! queue\_empty())

    {

        ptr = del\_queue ();

        printf ("%d ",ptr->info);

        if (ptr->lchild != NULL)

            insert\_queue(ptr->lchild );

        if (ptr->rchild != NULL)

            insert\_queue(ptr->rchild );

    }

    printf ("\n");

}

int main(){

    int p,i;

    struct listnode \* inorder, \*postorder;

    struct treenode \*binary\_tree;

    printf("Enter number of elements in inorder list: ");

    scanf("%d",&i);

    inorder = creatlist(i);

    printf("list created sucessfuly !\n");

    printf("Enter number of elements in postorder list: ");

    scanf("%d",&p);

    postorder = creatlist(p);

    printf("list created sucessfuly !\n");

    binary\_tree = construct\_in\_post(inorder, postorder, p);

    printf("Postorder traversal: ");

    post\_traversal(binary\_tree);

    int op, key,d;

    while(1){

        printf("\n1. Tree Height\n2. Depth of given node\n3. Level order traversal\n4. Exit\n");

        printf("Enter : ");scanf("%d",&op);

        switch(op){

        case 1:

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

            break;

        case 2:

            printf("Enter Node of which depth is to found out : ");

            scanf("%d",&key);

            d = depth(binary\_tree, key);

            if(d == -1)

                printf("Node with key %d not found !\n",key);

            else

                printf("Depth of node %d = %d",key,d);

            break;

        case 3:

            level\_trav(binary\_tree);

            break;

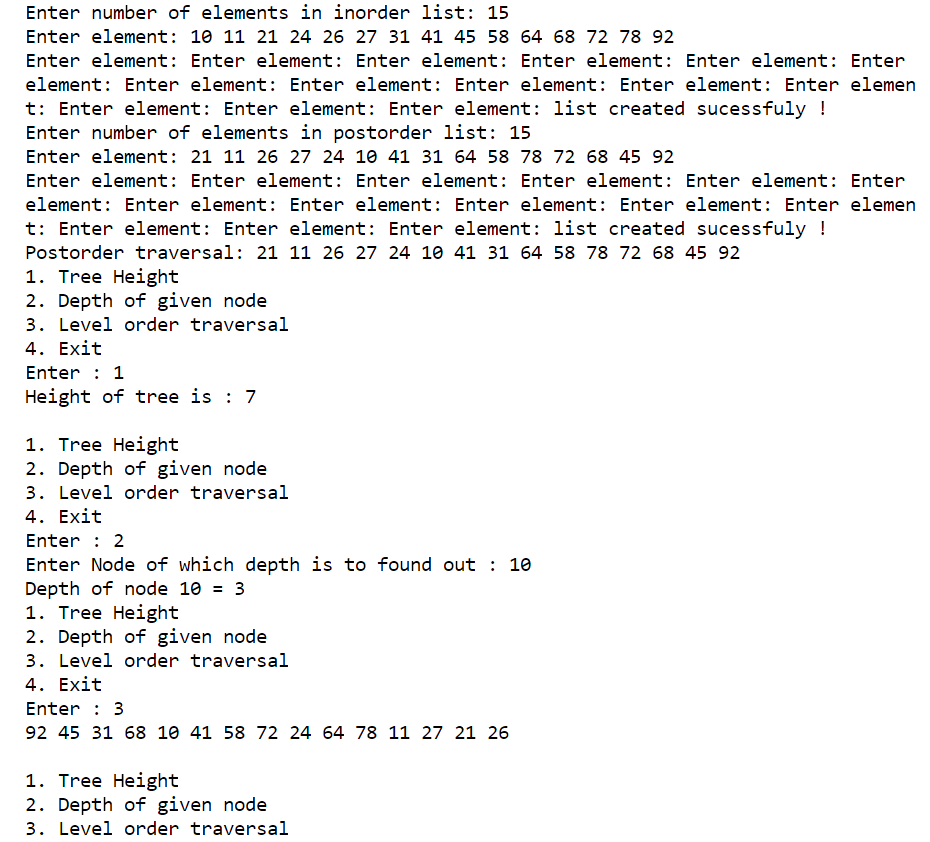
        case 4: return 0;

        default : printf("Invalid input !\n");

        }

    }

}

****

**CONCLUSION: all the basic concepts from binary tree were understood and implemented..**