CET214 – Data Structures & Algorithms

Experiment # 12

Experiment Title

Binary Trees

Assessment of CLO(s): IV

Performed on 20-12-2024

Student Name			
Roll No.		**Group**	
Semester		**Session**	

Total (Max)	**Criteria 1 (2.5)**	**Criteria 2 (2.5)**	**Criteria 3 (2.5)**	**Criteria 4 (2.5)**	**Total (10)**
Marks Obtained					
Remarks (if any)					

Experiment evaluated by

Instructor's Name	**Engr. Muhammad Asad Husain**		
Date		**Signature**	

Department of Electrical Engineering
(Usman Institute of Technology)

Course Code: CET214 Course Title: Data Structures & Algorithms Course Credits: 3+1 Session: Fall 2021

Rubric for assessment criteria to perform experiment number 13.

Level / Criteria	UNSATISFACTORY 1	COMPETENT 2	PROFICIENT 3	DISTINGUISHED 4
Capability of writing algorithm/ Procedure	None of the steps are implemented of an algorithm.	Few steps are implemented correctly of an algorithm.	Most of the steps are implemented correctly of an algorithm.	All the steps are implemented correctly of an algorithm.
Capability of writing Program	Programs not completed.	Completeness of code, consistent variable naming and unformatted.	Completeness of code, inconsistent variable naming and well formatted.	Completeness of code, consistent variable naming and well formatted.
Completion of target in Lab	25% target has been completed	50% target has been completed	75% target has been completed	100% target has been completed
Output	None of the outputs are correct.	Few outputs have been found correctly.	Some of the outputs are correct and well formatted.	Most of the outputs are correct and well formatted.

Practical Objective(s):

i. Getting familiar with Binary Trees

Theory

We have seen how objects can be linked into lists. When an object contains two pointers to objects of the same type, structures can be created that are much more complicated than linked lists. In this practical, we'll look at one of the most basic and useful structures of this type: Binary Trees. Each of the objects in a binary tree contains two pointers, typically called *left* and in addition to these pointers, of course, the nodes can contain other types of data. For example, a binary tree of integers could be made up of objects of the following type:

```
struct TreeNode
{
   int item;          // The data in this node.
   TreeNode *left;    // Pointer to the left subtree.
   TreeNode *right;   // Pointer to the right subtree.
}
```

The **left** and **right** pointers in a **TreeNode** can be **NULL** or can point to other objects of type **TreeNode**. A node that points to another node is said to be the parent of that node, and the node it points to is called a child. In figure 1, for example, node 3 is the parent of node 6, and nodes 4 and 5 are children of node 2.

Not every linked structure made up of tree nodes is a binary tree. A binary tree must have the following properties:

1. There is exactly one node in the tree which has no parent. This node is called the *root* of the tree.
2. Every other node in the tree has exactly one parent.
3. There can be no loops in a binary tree. That is, it is not possible to follow a chain of pointers starting at some node and arriving back at the same node.

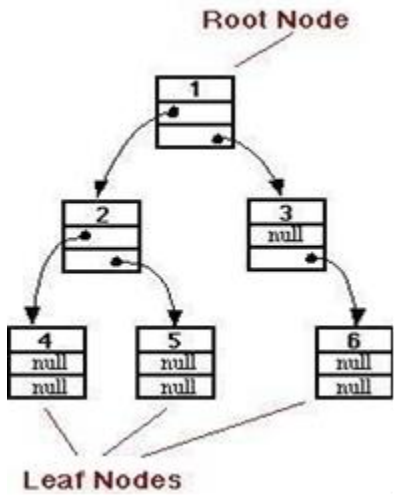

Figure 1. Binary Tree Representation

A node that has no child is called a leaf. A leaf node can be recognized by the fact that both the left and right pointers in the node are NULL. In the standard picture of a binary tree, the root node is shown at the top and the leaf nodes at the bottom.

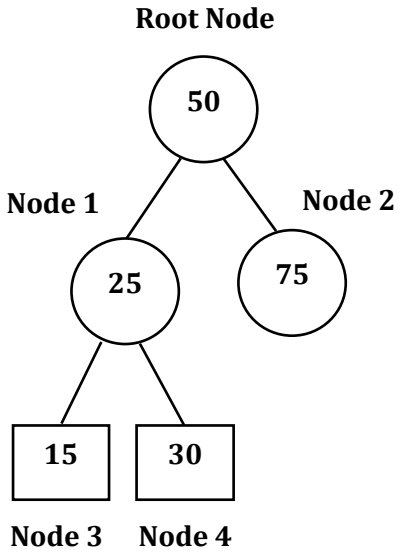

Figure 2. Binary Tree Example

Program:

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   struct  Treenode
   {
        int data;
        Treenode *left;
        Treenode *right;
   };

   Treenode *root_node,*node1,*node2,*node3,*node4;
   root_node= new  Treenode;

   node1= new Treenode;
   node2= new Treenode;
   node3= new Treenode;
   node4= new Treenode;

   root_node->data=50;
   root_node->left=node1;
   root_node->right=node2;

   node1->data=25;
   node1->left=node3;
   node1->right=node4;

   node2->data=75;
   node2->left=NULL;
```

```
    node2->right=NULL;

    node3->data=15;
    node3->left=NULL;
    node3->right=NULL;


    node4->data=30;
    node4->left=NULL;
    node4->right=NULL;


    cout<<root_node->left->data<<endl;
    cout<<root_node->right->data<<endl;

    return 0;
}
```

Do It Yourself:

1. Write a piece of code in the program given above which checks whether a particular node is a leaf node or not. To test your program, check for node 2 and node 4 as shown in figure 2.