\documentclass{article}

% Include the titlesec package

\usepackage{tikz}

\usetikzlibrary{trees}

\title{Transverse Binary Trees}

\author{Sagar Zujam(B-04) \and Prasad Kolte(B-02)}

\date{\today}

\begin{document}

\begin{titlepage}

\begin{center}

\textsc{\LARGE Experiment no. 10 :}\\[1.5cm]

\textsc{\LARGE Transverse Binary Trees}\\[1.5cm]

\end{center}

\section{Aim}

To Explore Transverse Binary Tree

\section{Learning Objectives}

\begin{enumerate}

\item Lab Objective: Students should able to design and analyze simple linear and non linear data structures.

\item It strengthen the ability to the students to identify and apply the suitable data structure for the given real world problem.

\item It enables them to gain knowledge in practical applications of data structures .

\end{enumerate}

\section{Theory}

\subsection{Introduction to Tree Traversals}

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways.

\subsubsection{Inorder Traversal}

Algorithm Inorder(tree):

\begin{enumerate}

\item Traverse the left subtree, i.e., call Inorder(left->subtree)

\item Visit the root.

\item Traverse the right subtree, i.e., call Inorder(right->subtree)

\end{enumerate}

\end{titlepage}

\begin{figure}

\centering

\begin{tikzpicture}[

level distance=1.5cm,

level 1/.style={sibling distance=3cm},

level 2/.style={sibling distance=1.5cm}

]

\node {1}

child {node {2}

child {node {4}}

child {node {5}}

}

child {node {3}};

\end{tikzpicture}

\caption{Binary Tree}

\end{figure}

Uses of Inorder Traversal:

In the case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal is reversed can be used.

Inorder Traversal: 4 2 5 1 3

\subsubsection{Preorder Traversal}

Preorder traversal is a depth-first tree traversal technique used to systematically visit nodes in a binary tree by first processing the current node, followed by its left and right subtrees. It explores nodes in a parent-first order, making it a fundamental method for tasks such as expression evaluation, generating prefix notation, creating copies of trees, parsing source code, and navigating file systems. In a simple pseudocode representation, it involves visiting the current node, traversing the left subtree, and then traversing the right subtree, making it a versatile tool for tree-based data structures and algorithms.

\subsubsection{Postorder Traversal}

Postorder traversal is a depth-first tree traversal technique used to systematically visit nodes in a binary tree. It explores nodes in a child-first order, first traversing the left and right subtrees before visiting the current node. Postorder traversal is integral to a range of applications, including expression evaluation and conversion in postfix notation, memory management, deletion in file systems, and mathematical calculations involving tree structures. The pseudocode for postorder traversal involves visiting the left and right subtrees and then processing the current node. This traversal method plays a crucial role in various tree-based data structures and algorithms, facilitating tasks that require a specific order of node processing.

\subsubsection{Some Other Tree Traversals Techniques}

Some of the other tree traversals are:

\paragraph{Level Order Traversal}

For each node, first, the node is visited, and then its child nodes are put in a FIFO queue. Then again the first node is popped out, and then its child nodes are put in a FIFO queue and repeat until the queue becomes empty.

\paragraph{Boundary Traversal}

The Boundary Traversal of a Tree includes:

\begin{enumerate}

\item Left boundary (nodes on the left excluding leaf nodes)

\item Leaves (consist of only the leaf nodes)

\item Right boundary (nodes on the right excluding leaf nodes)

\end{enumerate}

\paragraph{Diagonal Traversal}

In the Diagonal Traversal of a Tree, all the nodes in a single diagonal will be printed one by one.

\section{Lab Outcome}

After completing this experiment, participants are expected to achieve the following outcomes:

\begin{enumerate}

\item Write functions to implement linear and non-linear data structure

operations

\item Suggest appropriate linear / non-linear data structure operations for

solving a given problem

\end{enumerate}

\section{Conclusion}

\begin{enumerate}

\item As a result, this experiment gave us important new knowledge

about binary trees and how to traverse them.\\\\

\item In order to grasp binary trees' structure and importance in data

structures, we first explored their basic ideas.\\\\

\item We investigated a wide range of tree traversal methods, including

Inorder, Preorder, Postorder, Level Order, and others.\\\\

\item The algorithmic foundations of each traversal technique and the practical uses for which they are most useful were better understood.

\end{enumerate}

\section{Program}

\subsubsection{Code Implementation of Inorder Traversal}

% Your code can be placed here

\begin{verbatim}

// C program for different tree traversals

#include <stdio.h>

#include <stdlib.h>

// A binary tree node has data, pointer to left child

// and a pointer to right child

struct node {

int data;

struct node\* left;

struct node\* right;

};

// Helper function that allocates a new node with the

// given data and NULL left and right pointers.

struct node\* newNode(int data) {

struct node\* node = (struct node\*)malloc(sizeof(struct node));

node->data = data;

node->left = NULL;

node->right = NULL;

return (node);

}

// Given a binary tree, print its nodes in inorder

void printInorder(struct node\* node) {

if (node == NULL)

return;

// First recur on the left child

printInorder(node->left);

// Then print the data of the node

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

// Now recur on the right child

printInorder(node->right);

}

// Driver code

int main() {

struct node\* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

root->left->left = newNode(4);

root->left->right = newNode(5);

// Function call

printf("Inorder traversal of the binary tree is \n");

printInorder(root);

getchar();

return 0;

}

\end{verbatim}

\section{Output of the Program}

The provided C program creates a binary tree and performs an Inorder traversal of the tree. Here's the expected output of the program:

Inorder traversal of the binary tree is

\textbf{4 2 5 1 3}

\\

\end{document}