VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELGAUM

A Mini Project Report On

"IMPLEMENTATION OF BINARY TREE"

Submitted in partial fulfilment of the requirements for the award of the 3rd Sem of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING IN ARTIFICIAL INTELLIGENCE

Submitted by: Dharmendra Reddy M S [1VE21CA011]
Garela Vaishnavi [1VE21CA012]
Govind Ram n [1VE21CA013]
Hemanth Gowda C[1VE21CA015]

Under the guidance of
Mrs.PRATHIBHA. A
Asst. Prof, Dept. of CSE-AI

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

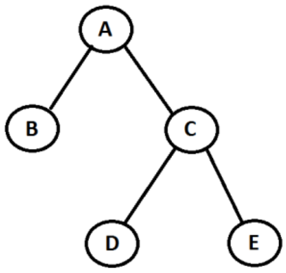
SRI VENKATESHWARA COLLEGE OF ENGINEERING

Affiliated to VTU Belgaum & Approved by AICTE New Delhi) an ISO 9001: 2008 Certified Kempegowda International Airport Road, Vidyanagar, Bengaluru, Karnataka, India-562157

2022-2023

Binary Tree: -

A binary tree is a data structure consisting of nodes, where each node has at most two children, referred to as left child and right child. The binary tree is called "binary" because each node has a maximum of two children.



DESCRIPTION OF BINARY TREE: -

1. A binary tree is a tree data structure where each node has at most two child nodes, referred to as the left child and the right child. The nodes in a binary tree are connected by edges, and the topmost node of the tree is called the root node. Each node in a binary tree can have either zero, one, or two children.

2. The structure of a binary tree is such that the left child of a node is always less than the parent node, and the right child is always greater than the parent node. This property makes binary trees useful in searching and sorting algorithms.

3. Binary trees can be classified into different types based on their properties, such as complete binary trees, balanced binary trees, and binary search trees.

4. A complete binary tree is a binary tree where all levels of the tree are completely filled, except possibly the last level, which is filled from left to right.

5. A balanced binary tree is a binary tree where the difference in height between the left and right subtrees of any node is at most one. Balanced binary trees are useful in maintaining efficient search and insertion operations.

6. A binary search tree is a binary tree where the left child of a node contains only nodes with values less than the parent node, and the right child contains only nodes with values greater than the parent node. This property allows for efficient searching and insertion operations.

7. Binary trees can be traversed in different ways, including preorder, inorder, and postorder traversals. Traversal involves visiting all the nodes in the tree in a specific order, based on the type of traversal being used.

PROBLEM: -

STEPS TO IMPLEMENT BINARY TREE PRE ORDER, INORDER, POSTORDER WITH VISUALIATION.

SOLUTION: -

1. Define a structure for the binary tree node: The binary tree node should have at least two properties, one for the node value and two for the left and right child nodes. You can also include additional properties such as the node depth, height, and parent node reference.

2. Implement the binary tree operations: You need to implement the basic binary tree operations, such as inserting a node, deleting a node, and searching for a node. You can also implement other operations, such as traversing the tree, finding the minimum and maximum values, and checking if the tree is balanced.

3. Visualize the binary tree: To visualize the binary tree, you can use a graphics library or a graph visualization tool. You need to create a visual representation of the binary tree by assigning positions to the nodes and drawing lines to connect them.

4. Traverse the binary tree: You can traverse the binary tree using one of the three common traversal methods: inorder, preorder, and postorder. Inorder traversal visits the left subtree, then the root, then the right subtree. Preorder traversal visits the root, then the left subtree, then the right subtree. Postorder traversal visits the left subtree, then the right subtree, then the root.

5. Output the visualization: You can output the visualization of the binary tree to a file, such as a PNG or SVG image, or display it directly on the screen.

AFTER GETTING TO KNOW ALL THESE STEPS OF IMPLEMENTATION A QUESTION WILL RISE LIKE **"HOW DOES THE CODE RUN?"**

HERE IS THE EXPLANATION REGARDING THIS: -

1. The code starts by declaring a struct node with three members: data, left, and right.

2. The new Node function allocates memory for the structure and then initializes it to point to itself.

3. The inorder function iterates over the nodes from left to right.

4. It prints out each node's data member before moving on to its next sibling.

5. The preorder function iterates over the nodes from right to left.

6. It prints out each node's data member before moving on to its next sibling.

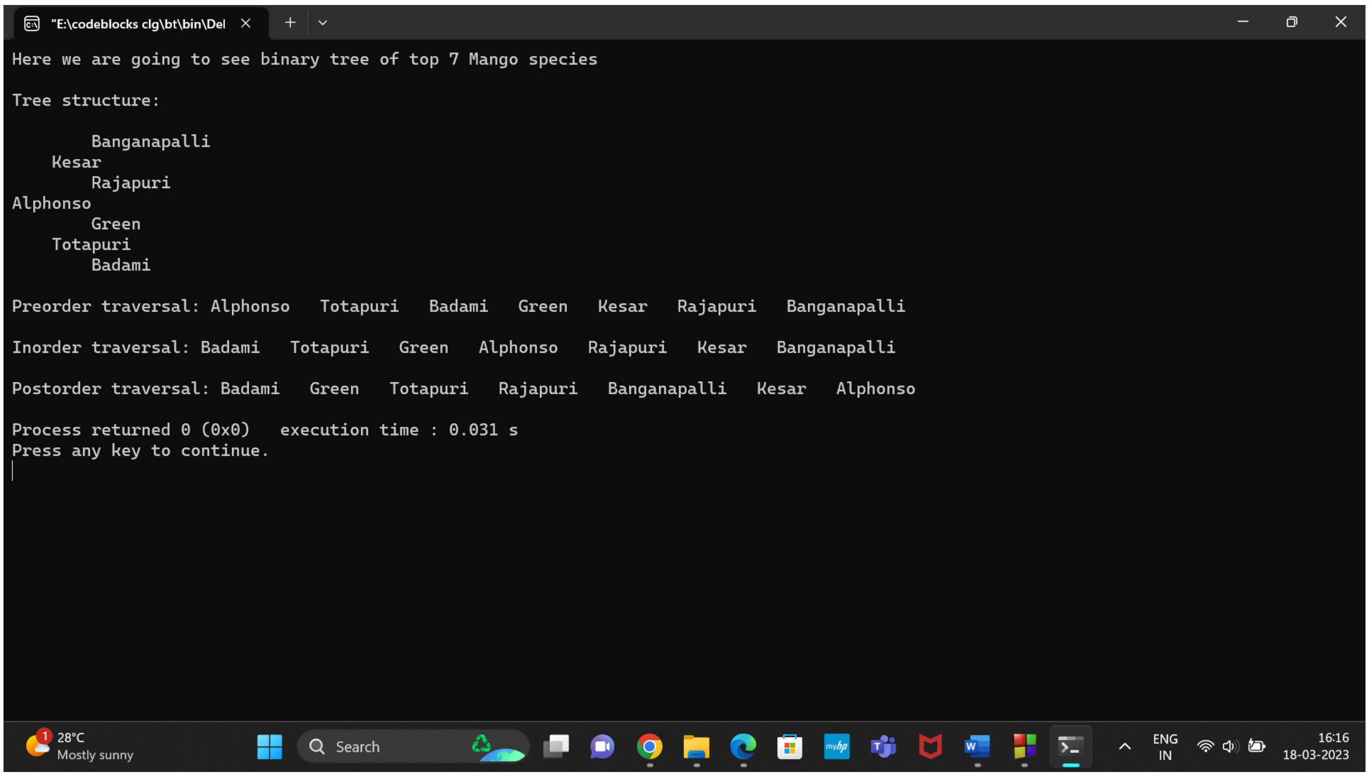
7. The postorder function iterates over the nodes from left to right, printing out each node's data member after it has visited all of its siblings (pre-siblings).

CODE: -

HERE IS THE CODE FOR BINARY TREE PREORDER, INORDER, POSTORDER WITH VISUALISATION: -

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node {
    char name[20];
    struct node* left;
    struct node* right;
};

struct node* create_node(char* name) {
    struct node* new_node = (struct node*) malloc(sizeof(struct node));
    strcpy(new_node->name, name);
    new_node->left = NULL;
    new_node->right = NULL;
    return new_node;
}

void print_tree(struct node* root, int level) {
    if (root == NULL) {
        return;
    }
    print_tree(root->right, level + 1);
    for (int i = 0; i < level; i++) {
        printf("    ");
    }
    printf("%s\n", root->name);
    print_tree(root->left, level + 1);
}

void preorder_traversal(struct node* root) {
    if (root == NULL) {
        return;
    }
    printf("%s   ", root->name);
    preorder_traversal(root->left);
    preorder_traversal(root->right);
}

void inorder_traversal(struct node* root) {
    if (root == NULL) {
        return;
    }
    inorder_traversal(root->left);
    printf("%s   ", root->name);
    inorder_traversal(root->right);
}

void postorder_traversal(struct node* root) {
    if (root == NULL) {
        return;
    }
    postorder_traversal(root->left);
    postorder_traversal(root->right);
    printf("%s   ", root->name);
}

int main() {
    struct node* root = create_node("Alphonso");
    root->left = create_node("Totapuri");
    root->right = create_node("Kesar");
    root->left->left = create_node("Badami");
    root->left->right = create_node("Green");
    root->right->left = create_node("Rajapuri");
    root->right->right = create_node("Banganapalli");

    printf("Here we are going to see binary tree of top 7 Mango species \n");

    printf("\nTree structure: -\n\n");
    print_tree(root, 0);

    printf("\nPreorder traversal: - ");
    preorder_traversal(root);
    printf("\n");

    printf("\nInorder traversal: - ");
    inorder_traversal(root);
    printf("\n");

    printf("\nPostorder traversal: -");
    postorder_traversal(root);
    printf("\n");



    return 0;
}
```

OUTPUT OF THE ABOVE CODE: -



```
Here we are going to see binary tree of top 7 Mango species

Tree structure:

        Banganapalli
    Kesar
        Rajapuri
Alphonso
        Green
    Totapuri
        Badami

Preorder traversal: Alphonso   Totapuri   Badami   Green   Kesar   Rajapuri   Banganapalli

Inorder traversal: Badami   Totapuri   Green   Alphonso   Rajapuri   Kesar   Banganapalli

Postorder traversal: Badami   Green   Totapuri   Rajapuri   Banganapalli   Kesar   Alphonso

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

VISUALIZATION: -

THE LINK BELOW PASTED HERE SHOWS THE VISUALIZATION OF HOW THE ELEMENTS ARE INSERTED IN BINARY TREE AND HOW THE PRE ORDER TRAVERSAL, INORDER TRAVERSAL AND POSTORDER TRAVERSAL HAS BEEN TAKEN PLACED

https://drive.google.com/file/d/1rmTrzBP5LlsoCvMPP99Kl51Cs2Li6MWZ/view?usp=drivesdk

DRAWBACKS OF BINARY TREE: -

1. Unbalanced Trees: A binary tree is said to be unbalanced if the height of the left subtree and the height of the right subtree differ significantly. If the tree is unbalanced, the search operation can take longer since the search operation starts at the root of the tree and continues down to the leaf node in the worst case. An unbalanced binary tree can also cause other issues, such as memory waste.

2. Limited Search Methods: Binary trees only support search operations using the key values. If you want to search for a specific value that is not the key, you need to perform a full tree traversal, which can be inefficient for large trees.
3. Not Suitable for Large Data Sets: Binary trees are not efficient for large data sets. If the binary tree is large, the search operation can take a long time, and it may not be feasible to store the tree in memory.

4. No Guarantees for Balanced Trees: Although there are balanced binary trees like AVL trees and red-black trees, not all binary trees are balanced. If the binary tree is not balanced, the performance of the search operation can suffer significantly.

5. Insertion and Deletion Can be Costly: If the binary tree is unbalanced, the cost of insertion and deletion operations can be high. These operations require rearranging the tree, which can result in a significant number of nodes being moved, causing a performance impact.