MIT World Peace University
Advanced Data Structures

Assignment 2

Naman Soni Roll No. 10

Contents

1 Problem Statement

Implement binary tree using C++ and perform following operations: Creation of binary tree and traversal (recursive and non- recursive).

2 Objective

1. To study data structure : Tree and Binary Tree.
2. To study different traversals in Binary Tree
3. To study recursive and non-recursive approach of programming

3 Theory

3.1 *Trees*

Trees are a type of data structure used to represent hierarchical relationships between elements. In computer science, trees are often used to represent hierarchical relationships between elements such as files and directories in a file system, or in the representation of a decision tree for machine learning algorithms.

A tree consists of nodes, which are connected by edges. Each node in a tree has zero or more child nodes, and a single parent node, except for the root node, which has no parent. The root node is the topmost node in the tree, and all other nodes are descended from it. Nodes that have no children are called leaf nodes.

Trees have several important properties, such as being able to represent hierarchical relationships efficiently, supporting searching, insertion, and deletion operations in logarithmic time, and being able to be easily traversed and manipulated.

There are several different types of trees, such as binary trees, AVL trees, B-trees, and heap trees, each with its own unique properties and uses.

3.2 *Different definitions related to binary tree*

In computer science, a binary tree is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child. Some common definitions related to binary trees are:

- Root: The topmost node in a binary tree is called the root. It has no parent node.
- Leaf: A node that has no children is called a leaf node.
- Parent: A node that has one or more child nodes is called a parent node.
- Child: A node that is a direct descendant of a parent node is called a child node.
- Siblings: Nodes that have the same parent are called siblings.
- Depth: The depth of a node is the number of edges from the root node to that node.
- Height: The height of a binary tree is the number of edges from the root node to the deepest leaf node.
- Subtree: The set of nodes and edges that are descendants of a given node is called a subtree.
- Full binary tree: A binary tree is considered full if every node has either 0 or 2 children.
- Perfect binary tree: A binary tree is considered perfect if all its leaf nodes are at the same depth and every parent node has exactly two children.

3.3 *Different Traversals (Inorder, Preorder and Postorder)*

1. Inorder traversal: In an inorder traversal, the left subtree of a node is visited first, then the node itself, and finally the right subtree. This results in visiting the nodes in ascending order if the tree is used to represent an ordered set.

2. Preorder traversal: In a preorder traversal, the node itself is visited first, then the left subtree, and finally the right subtree. This is useful for creating a copy of the tree or for creating an output that represents the structure of the tree.

3. Postorder traversal: In a postorder traversal, the left subtree is visited first, then the right subtree, and finally the node itself. This is useful for freeing up memory in a tree-like data structure, as the children can be deleted before the parent.

4 Implementation

4.1 *Platform*

Operating System: Mac OS 64-bit
IDE Used: Visual Studio Code

4.2 *Test Conditions*

1. Input at least 10 nodes.

2. Display all traversals of binary tree with 10 nodes. (recursive and non-recursive)

4.3 *Input-Output Code*

```
#include <iostream>
using namespace std;
class treenode {
  public :
  string data;
  treenode *left;
  treenode *right;
  friend class tree;
};
class stack{
  int top;
  treenode *data [30];
  public:
  stack()
  {
    top=-1;
  }
  void push (treenode *temp)
  {
    top++;
    data[top] = temp;
  }
  treenode*pop()
  {
    return data[top--];
  }
  int isempty(){
    if (top == -1)
    {
      return 1;
    }
    else
```

```
    {
      return 0;
    }
  }
  friend class tree;
};

class tree {
  public:
  treenode *root;
  tree(){
    root = NULL;
  }
  void create_r()
  {
    root = new treenode;
    cout<<"Enter ROOT Data:"<<endl;
    cin>>root->data;
    root->left = NULL;
    root->right = NULL;
    create_r(root);
  }

  void create_r(treenode * temp){
    char choice1, choice2;
    cout << "Enter whether you want to add the data to the left of \""<< temp->data <<"\"
  or not (y/n): ";
    cin >> choice1;
    if (choice1 == 'y'){
      treenode *curr1 = new treenode();
      cout << "Enter the data for the left of \""<< temp->data << "\" node: ";
      cin >> curr1 -> data;
      temp -> left = curr1;
      create_r(curr1);
    }
    cout << "Enter whether you want to add the data to the right of \""<<temp->data<<"\"
  or not (y/n): ";
    cin >> choice2;
    if (choice2 == 'y'){
      treenode *curr2 = new treenode();
      cout << "Enter the data for the Right of \""<< temp->data <<"\" node:";
      cin >> curr2 -> data;
      temp -> right = curr2;
      create_r(curr2);

    }
  }
  //inorder recursive
  void inorder_traversal_r (treenode * node){
    if (node == NULL){
      return;
    }
    inorder_traversal_r(node->left);
    cout << node->data << "\n";
    inorder_traversal_r(node->right);
  }
  //preorder recursive
  void preorder_traversal_r(treenode* temp)
  {
    if (temp!= NULL)
    {
      cout<< temp->data << endl;
      preorder_traversal_r(temp->left);
      preorder_traversal_r(temp->right);
    }
  }
  // postorder recursive
  void postorder_traversal_r(treenode*temp)
```

```
    {
      if(temp!=NULL)
      {
        postorder_traversal_r(temp->left);
        postorder_traversal_r(temp->right);
        cout<<temp->data<<endl;
      }
    }
    //Inorder Non recursive
    void inorder_traversal_non_recursive(treenode*temp)
    {
      temp =root;
      stack st;
      while (1)
      {
        while(temp!=NULL)
        {
          st.push(temp);
          temp = temp->left;
        }
        if(st.isempty()==1)
        {
          break;
        }
        temp=st.pop();
        cout<< temp->data<<endl;
        temp = temp->right;
      }
    }

    //Postorder non recursive
    void postorder_traversal_non_recursive(treenode*temp)
    {
      temp = root;
      stack st;
      while(1)
      {
        while(temp!=NULL)
        {
          st.push(temp);
          temp = temp->left;
        }
        if (st.data[st.top]->right == NULL)
        {
          temp=st.pop();
          cout<<temp->data<<endl;
        }
        while(st.isempty()==0 && st.data[st.top]->right==temp)
        {
          temp=st.pop();
          cout<<temp->data;
        }
        if (st.isempty()==1)
        {
          break;
        }
        temp = st.data[st.top]->right;
      }
    }

    //Preorder Non recursive
    void preorder_traversal_non_recursive(treenode*temp)
    {
      temp = root;
      stack st;
      while(1)
      {
        while(temp!=NULL)
```

```
        {
          cout<<temp->data<<endl;
          st.push(temp);
          temp =temp->left;
        }
        if(st.isempty()==1)
        {
          break;
        }
        temp=st.pop();
        temp = temp->right;
      }
    }
};
/*Test cases
1. left skewed
2. Right skewed
3. Complete Tree
4. Full tree
5. Normal Tree
6. Binary search Tree
*/
int main() {
  tree bt;
  // treenode * root = new treenode();
  // cout << "Enter the data for the root node: ";
  // cin >> root -> data;
  int ch;
  char c;
  do {

    cout<<"\nWhat you want to do:\n1.Enter Tree\n2.Inorder Recursive\n3.Preorder Recursive
  \n4.Postorder Recursive\n5.Inorder Non Recursive\n6.Preorder Non Recursive\n7.Postorder
  Non Recursive\nchoose:"<<endl;
    cin>>ch;
    switch (ch)
    {
      case 1:
      bt.create_r();
      break;
      case 2:
      cout << "The inorder display of the tree is: " << endl;
      bt.inorder_traversal_r(bt.root);
      break;
      case 3:
      cout<<"The preorder display of the tree is:" << endl;
      bt.preorder_traversal_r(bt.root);
      break;
      case 4:
      cout<<"The postorder display of the tree is:"<<endl;
      bt.postorder_traversal_r(bt.root);
      break;
      case 5:
      cout<<"The Inorder Non Recursive Display of the tree is:"<<endl;
      bt.inorder_traversal_non_recursive(bt.root);
      break;
      case 6:
      cout<<"The Preorder Non Recursive Display of the tree is:"<<endl;
      bt.preorder_traversal_non_recursive(bt.root);
      break;
      case 7:
      cout<<"The Postorder Non Recursive Display of the tree is:"<<endl;
      bt.postorder_traversal_non_recursive(bt.root);
      break;
      default:
      cout<<"Invalid choice";
      break;
    }
```

```
233
234         cout << "Do you want to continue(y/n):";
235         cin>> c;
236       }while (c == 'y');
237
238
239       return 0;
240     }
```

Listing 1: Input

```
1     What you want to do:
2   1. Enter Tree
3   2.Inorder Recursive
4   3.Preorder Recursive
5   4.Postorder Recursive
6   5.Inorder Non Recursive
7   6.Preorder Non Recursive
8   7.Postorder Non Recursive
9   choose:
10  1
11
12  Enter ROOT Data:
13  1
14  Enter whether you want to add the data to the left of "1" or not (y/n):  y
15  Enter the data for the left of "1" node: 2
16  Enter whether you want to add the data to the left of "2" or not (y/n):  y
17  Enter the data for the left of "2" node: 3
18  Enter whether you want to add the data to the left of "3" or not (y/n):  y
19  Enter the data for the left of "3" node: 4
20  Enter whether you want to add the data to the left of "4" or not (y/n): y
21  Enter the data for the left of "4" node: 5
22  Enter whether you want to add the data to the left of "5" or not (y/n):  n
23  Enter whether you want to add the data to the right of "5" or not (y/n): n
24  Enter whether you want to add the data to the right of "4" or not (y/n): n
25  Enter whether you want to add the data to the right of "3" or not (y/n): n
26  Enter whether you want to add the data to the right of "2" or not (y/n): n
27  Enter whether you want to add the data to the right of "1" or not (y/n): n
28
29  Do you want to continue(y/n): y
30
31  What you want to do:
32  1. Enter Tree
33  2.Inorder Recursive
34  3.Preorder Recursive
35  4.Postorder Recursive
36  5.Inorder Non Recursive
37  6.Preorder Non Recursive
38  7.Postorder Non Recursivechoose:
39  2
40  The inorder display of the tree is:
41  5 4 3 2 1
42  Do you want to continue(y/n): y
43
44  What you want to do:
45  1. Enter Tree
46  2.Inorder Recursive
47  3.Preorder Recursive
48  4.Postorder Recursive
49  5.Inorder Non Recursive
50  6.Preorder Non Recursive
51  7.Postorder Non Recursive
52  choose:
53  3
54  The preorder display of the tree is:
55  1 2 3 4 5
56  Do you want to continue(y/n): y
57
```

```
What you want to do:
1. Enter Tree
2.Inorder Recursive
3.Preorder Recursive
4.Postorder Recursive
5.Inorder Non Recursive
6.Preorder Non Recursive
7.Postorder Non Recursive
choose:
4
The postorder display of the tree is:
5 4 3 2 1
Do you want to continue(y/n): y

What you want to do:
1. Enter Tree
2.Inorder Recursive
3.Preorder Recursive
4.Postorder Recursive
5.Inorder Non Recursive
6.Preorder Non Recursive
7.Postorder Non Recursive
choose:
5
The Inorder Non Recursive Display of the tree is:
5 4 3 2 1
Do you want to continue(y/n): y

What you want to do:
1. Enter Tree
2.Inorder Recursive
3.Preorder Recursive
4.Postorder Recursive
5.Inorder Non Recursive
6.Preorder Non Recursive
7.Postorder Non Recursive
choose:
 6
The Preorder Non Recursive Display of the tree is:
1 2 3 4 5
Do you want to continue(y/n): y

What you want to do:
1. Enter Tree
2.Inorder Recursive
3.Preorder Recursive
4.Postorder Recursive
5.Inorder Non Recursive
6.Preorder Non Recursive
7.Postorder Non Recursive
choose:
7
The Postorder Non Recursive Display of the tree is:
5 4 3 2 1
Do you want to continue(y/n): n
```

Listing 2: output

5 Conclusion

Thus, implemented different operations on CLL.

6 FAQ's

1. Explain any one application of binary tree with suitable example.

   Ans. One common application of binary trees is in the implementation of search algorithms, such as binary search.

   In a binary search algorithm, a sorted list of elements is represented as a binary tree, with the value of the root node being the middle element of the list. If the value being searched for is less than the root node, the search continues in the left subtree. If the value being searched for is greater than the root node, the search continues in the right subtree. This process continues recursively until the value is found or it is determined that the value is not in the list.

   For example, consider the following sorted list of integers: [2, 5, 7, 9, 12, 15, 20]. To perform a binary search for the value 12, we can represent the list as a binary tree as follows:

```
            12
           /  \
          7    15
          / \   /
        5   9 20
        /
      2

```

Listing 3: Example

   Starting at the root node, 12, we see that the value 12 is equal to the root node, so the search is successful and we return the index of the node. If we were searching for the value 7, we would follow the left subtree and continue the search, until we find the node with value 7 and return its index.

2. Explain sequential representation of binary tree with example.

   Ans. A binary tree can be represented in a sequential manner, also known as an array representation, where the tree nodes are stored in an array in a specific order. This representation is commonly used when implementing binary trees in programming languages.

   In the sequential representation, the root node is stored at the first position in the array (index 0), and its children are stored at the next two positions (index 1 and 2). The children of the node at position i are stored at positions 2i + 1 and 2i + 2, where i is the index of the node.

   Here is an example of a binary tree and its sequential representation:

```
  18
  /  \
            15     30
           / \           / \
      40 50 100 40

```

Listing 4: Example

   Array representation: [18, 15, 30, 40, 50, 100, 40]

   In this example, the root node 18 is stored at index 0, its left child 15 is stored at index 1, and its right child 30 is stored at index 2. The left child of 15 is stored at index 3, the right child of 15 is stored at index 4, the left child of 30 is stored at index 5, and the right child of 30 is stored at index 6.

This sequential representation allows us to access the nodes of the binary tree in an efficient manner, as we can calculate the position of a node's children and parent based on its index in the array.

3. Write inorder, preorder and postorder for following tree.

Ans.

- Inorder of the tree is: 8 40 7 15 9 50 18 100 30 40
- Postorder of the tree is: 8 7 40 9 50 15 100 40 30 18
- Preorder of the tree is: 18 15 40 8 7 50 9 30 100 40