Department of
Computer Science and Engineering

Title: Implementation of Tree and Tree Traversals

Data Structure Lab

CSE 106

Green University of Bangladesh

1 Objective(s)

- To be familiar with Tree and Tree Traversal.
- To learn problem solving techniques using C.

2 Problem analysis

Tree: In computer science, a tree is a widely used abstract data type that simulates a hierarchical tree structure, with a root value and sub-trees of children with a parent node, represented as a set of linked nodes. Other data structures such as arrays, linked list, stack, and queue are linear data structures that store data sequentially. In order to perform any operation in a linear data structure, the time complexity increases with the increase in the data size. But it is not acceptable in today's computational world. Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure.

Example of a Tree:

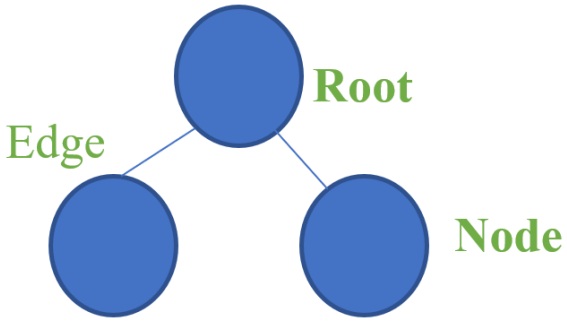

Figure 1: The structure of a Tree

Tree Traversal: Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree:

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Example of Tree Traversal:

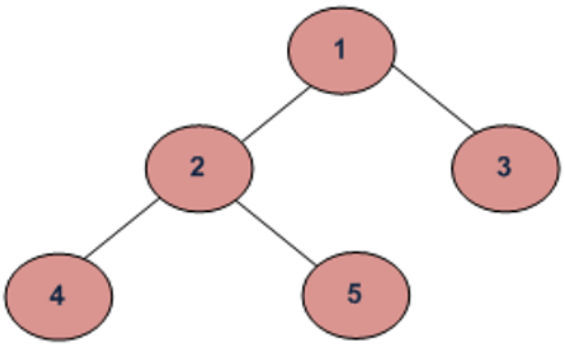

Figure 2: Tree Traversal of a Tree

Depth First Traversals:

- In-order (Left, Root, Right): 4 2 5 1 3

- Pre-order (Root, Left, Right): 1 2 4 5 3
- Post-order (Left, Right, Root): 4 5 2 3 1

Breadth First or Level Order Traversal: 1 2 3 4 5

In-order Traversal: In this traversal method, the left sub-tree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a sub-tree itself. If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.

Example of In-order Tree Traversal:

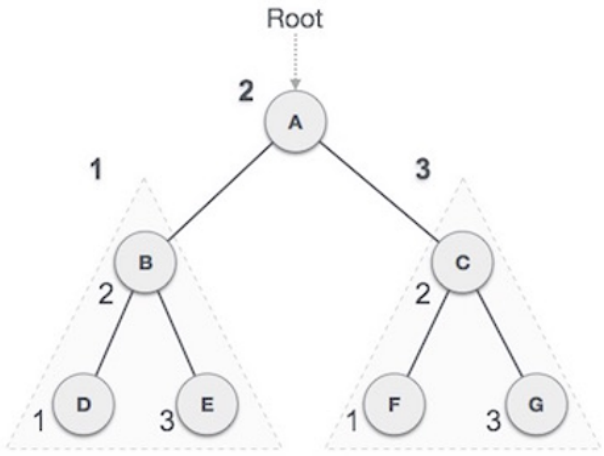

Figure 3: In-order Tree Traversal of a Tree

We start from A, and following in-order traversal, we move to its left sub-tree B. B is also traversed in-order. The process goes on until all the nodes are visited. The output of in-order traversal of this tree will be $- D \rightarrow B \rightarrow E \rightarrow A \rightarrow F \rightarrow C \rightarrow G$

Pre-order Traversal: In this traversal method, the root node is visited first, then the left sub-tree and finally the right sub-tree.

Example of Pre-order Tree Traversal:

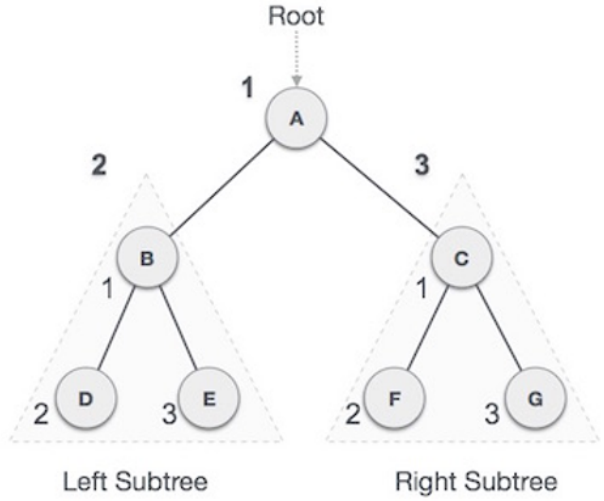

Figure 4: Pre-order Tree Traversal of a Tree

We start from A, and following pre-order traversal, we first visit A itself and then move to its left sub-tree B. B is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be $- A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$

Post-order Traversal: In this traversal method, the root node is visited last, hence the name. First, we traverse the left sub-tree, then the right sub-tree and finally the root node.

Example of Post-order Tree Traversal:

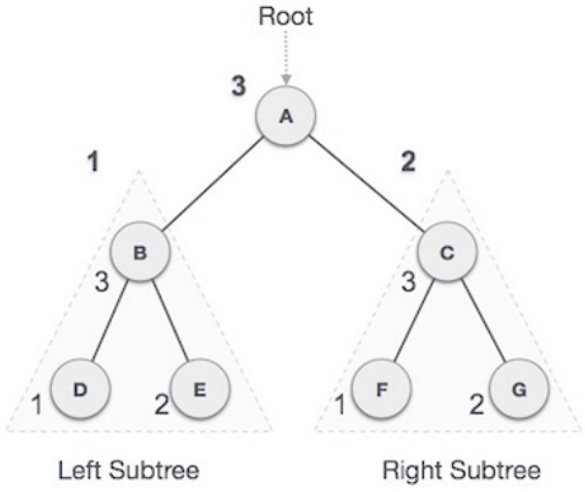

Figure 5: Post-order Tree Traversal of a Tree

We start from A, and following post-order traversal, we first visit the left sub-tree B. B is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be – $D \rightarrow E \rightarrow B \rightarrow F \rightarrow G \rightarrow C \rightarrow A$

3 Flowchart

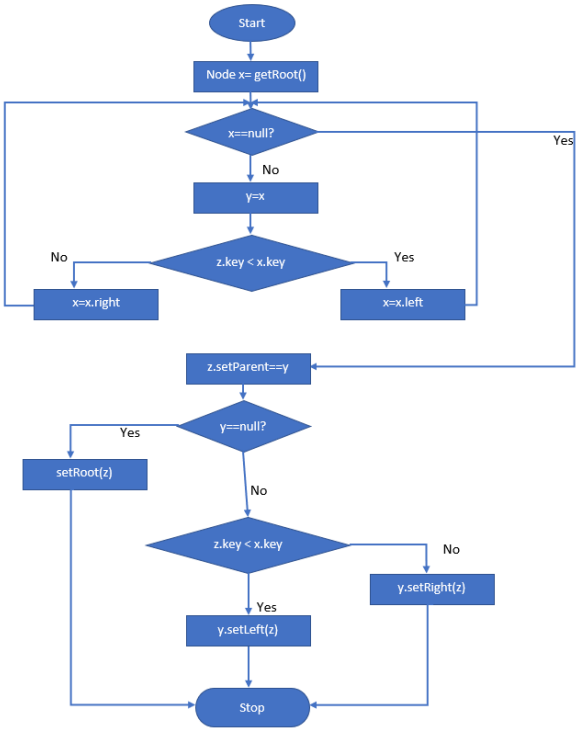

Figure 6: Flowchart of Inserting a Node in a Tree

4 Algorithm

Algorithm 1: Inserting A Node

Input: Element

`/* Algorithm for Inserting a node in a tree */`

1. Create a new node and assign values to it. insert(node, key)
2. If root == NULL, return the new node to the calling function.
3. If root=>data < key Call the insert function with root=>right and assign the return value in root=>right. root->right = insert(root=>right,key)
4. if root=>data > key
5. call the insert function with root->left and assign the return value in root=>left. root=>left = insert(root=>left,key)
6. Finally, return the original root pointer to the calling function.

Algorithm 2: Pre-order Traversal

Input: Element

`/* Algorithm for In-order Traversal of a tree */`

1. Recursively traverse left sub-tree.
2. Visit root node.
3. Recursively traverse right sub-tree.

Algorithm 3: In-order Traversal

Input: Element

`/* Algorithm for Pre-order Traversal of a tree */`

1. Visit root node.
2. Recursively traverse left sub-tree.
3. Recursively traverse right sub-tree.

Algorithm 4: Post-order Traversal

Input: Element

`/* Algorithm for Post-order Traversal of a tree */`

1. Recursively traverse left sub-tree.
2. Recursively traverse right sub-tree.
3. Visit root node.

5 Implementation in C (Inserting a Node in a Tree)

```c
// C program to demonstrate insert
// operation in binary
// search tree.
#include <stdio.h>
#include <stdlib.h>

struct node {
    int key;
    struct node *left, *right;
};

// A utility function to create a new BST node
struct node* newNode(int item)
{
    struct node* temp
        = (struct node*)malloc(sizeof(struct node));
    temp->key = item;
```

```
18      temp->left = temp->right = NULL;
19      return temp;
20  }
21
22  // A utility function to do inorder traversal of BST
23  void inorder(struct node* root)
24  {
25      if (root != NULL) {
26          inorder(root->left);
27          printf("%d \n", root->key);
28          inorder(root->right);
29      }
30  }
31
32  /* A utility function to insert
33     a new node with given key in
34   * BST */
35  struct node* insert(struct node* node, int key)
36  {
37      /* If the tree is empty, return a new node */
38      if (node == NULL)
39          return newNode(key);
40
41      /* Otherwise, recur down the tree */
42      if (key < node->key)
43          node->left = insert(node->left, key);
44      else if (key > node->key)
45          node->right = insert(node->right, key);
46
47      /* return the (unchanged) node pointer */
48      return node;
49  }
50
51  // Driver Code
52  int main()
53  {
54      /* Let us create following BST
55                50
56             /     \
57            30      70
58           /  \    /  \
59         20   40  60   80 */
60      struct node* root = NULL;
61      root = insert(root, 50);
62      insert(root, 30);
63      insert(root, 20);
64      insert(root, 40);
65      insert(root, 70);
66      insert(root, 60);
67      insert(root, 80);
68
69      // print inroder traversal of the BST
70      inorder(root);
71
72      return 0;
73  }
```

6 Sample Input/Output (Compiling, Debugging & Testing)

Output:
20, 30, 40, 50, 60, 70, 80

7 Implementation in C (In-order Traversal)

```c
#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 left child */
    printInorder(node->left);

    /* then print the data of node */
    printf("%d ", node->data);

    /* now recur on right child */
    printInorder(node->right);
}

/* Driver program to test above functions*/
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);


```

```
51
52      printf("\nInorder traversal of binary tree is \n");
53      printInorder(root);
54
55
56      getchar();
57      return 0;
58  }
```

8 Sample Input/Output (Compiling, Debugging & Testing)

Output:
Inorder traversal of binary tree is 4 2 5 1 3

9 Discussion & Conclusion

Based on the focused objective(s) to understand about the TREE TRAVERSALS, the additional lab exercise made me more confident towards the fulfilment of the objectives(s).

10 Lab Task (Please implement yourself and show the output to the instructor)

1. Implement Pre-order Traversal.

11 Lab Exercise (Submit as a report)

- Implement Post-order Traversal using linked list.

12 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.