



UNIVERSITY OF INFORMATION TECHNOLOGY AND SCIENCES (UITs)

LAB REPORT - 5

IT-214 : ALGORITHM LAB

Tree Traversal And Bfs

Submitted To:

Sumaiya Akhtar Mitu
Lecturer,
Department of IT, UITs

Submitted By:

Name: Nazmul Zaman
Student ID:2014755055
Department of IT UITs

24 MAY 2022

Contents

1	Abstrac	2
2	Introduction	2
3	Procedure	2
4	Working methods	4
4.1	(TreeTraversal)	4
4.2	BFS)	6
5	Conclusion	8
6	References	8

1 Abstrac

In mathematics and computer science, an algorithm is a finite sequence of well-defined instructions, typically used to solve a class of specific problems or to perform a computation. Algorithms are used as specifications for performing calculations and data processing.

2 Introduction

In this task we are going to learn about Tree traversal and Bfs there different methods.

Tree Traversal: A tree is a hierarchical data structure defined as a collection of nodes.

Traversing a tree means visiting every node in the tree. You might, for instance, want to add all the values in the tree or find the largest one. For all these operations, you will need to visit each node of the tree. Linear data structures like arrays, stacks, queues, and linked list have only one way to read the data. But a hierarchical data structure like a tree can be traversed in different ways.

Bfs : Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.

Time comeplexity : $O(V + E)$

Space complexity : $O(V)$.

3 Procedure

TreeTraversal :

Inorder traversal :

- 1.First, visit all the nodes in the left subtree
- 2.Then the root node
- 3.Visit all the nodes in the right subtree

Preorder traversal

- 1.Visit root node
- 2.Visit all the nodes in the left subtree

3. Visit all the nodes in the right subtree

Postorder traversal

1. Visit all the nodes in the left subtree
2. Visit all the nodes in the right subtree
3. Visit the root node

BFS:

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

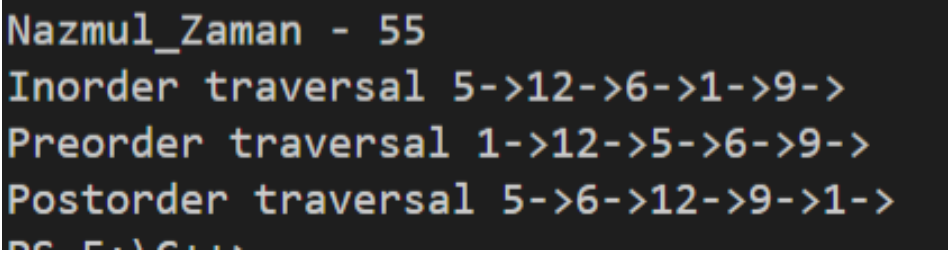
4 Working methods

4.1 (TreeTraversal)

```
1 // Tree traversal in C++
2
3 #include <iostream>
4 using namespace std;
5
6 struct Node {
7     int data;
8     struct Node *left, *right;
9     Node(int data) {
10         this->data = data;
11         left = right = NULL;
12     }
13 };
14
15 // Preorder traversal
16 void preorderTraversal(struct Node* node) {
17     if (node == NULL)
18         return;
19
20     cout << node->data << "->";
21     preorderTraversal(node->left);
22     preorderTraversal(node->right);
23 }
24
25 // Postorder traversal
26 void postorderTraversal(struct Node* node) {
27     if (node == NULL)
28         return;
29
30     postorderTraversal(node->left);
31     postorderTraversal(node->right);
32     cout << node->data << "->";
33 }
34
35 // Inorder traversal
36 void inorderTraversal(struct Node* node) {
37     if (node == NULL)
38         return;
39
40     inorderTraversal(node->left);
41     cout << node->data << "->";
42     inorderTraversal(node->right);
43 }
44
45 int main() {
46     struct Node* root = new Node(1);
47     root->left = new Node(12);
```

```
48     root->right = new Node(9);
49     root->left->left = new Node(5);
50     root->left->right = new Node(6);
51
52     cout << "Inorder traversal ";
53     inorderTraversal(root);
54
55     cout << "\nPreorder traversal ";
56     preorderTraversal(root);
57
58     cout << "\nPostorder traversal ";
59     postorderTraversal(root);
60 }
```

?



```
Nazmul_Zaman - 55
Inorder traversal 5->12->6->1->9->
Preorder traversal 1->12->5->6->9->
Postorder traversal 5->6->12->9->1->
```

Figure 1: OUTPUT

4.2 BFS)

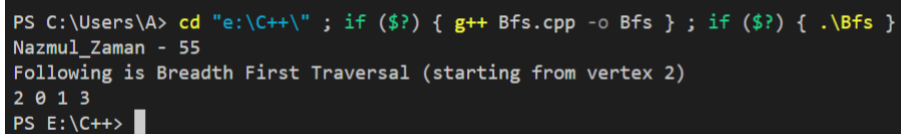
```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 // This class represents a directed graph using
5 // adjacency list representation
6 class Graph
7 {
8     int V; // No. of vertices
9
10
11     vector<list<int>> adj;
12 public:
13     Graph(int V); // Constructor
14
15     // function to add an edge to graph
16     void addEdge(int v, int w);
17
18     // prints BFS traversal from a given source s
19     void BFS(int s);
20 };
21
22 Graph::Graph(int V)
23 {
24     this->V = V;
25     adj.resize(V);
26 }
27
28 void Graph::addEdge(int v, int w)
29 {
30     adj[v].push_back(w); // Add w to adj[v] list.
31 }
32
33 void Graph::BFS(int s)
34 {
35     // Mark all the vertices as not visited
36     vector<bool> visited;
37     visited.resize(V, false);
38
39     // Create a queue for BFS
40     list<int> queue;
41
42     // Mark the current node as visited and enqueue it
43     visited[s] = true;
44     queue.push_back(s);
45
46     while(!queue.empty())
47     {
48         // Dequeue a vertex from queue and print it
49         s = queue.front();
50         cout << s << " ";
```

```

51     queue.pop_front();
52
53
54     for (auto adjacent: adj[s])
55     {
56         if (!visited[adjacent])
57         {
58             visited[adjacent] = true;
59             queue.push_back(adjacent);
60         }
61     }
62 }
63 }
64
65 // Driver program to test methods of graph class
66 int main()
67 {
68     // Create a graph given in the above diagram
69
70     cout<<"Nazmul_Zaman - 55"<<endl;
71     Graph g(4);
72     g.addEdge(0, 1);
73     g.addEdge(0, 3);
74     g.addEdge(1, 2);
75     g.addEdge(2, 0);
76     g.addEdge(2, 2);
77     g.addEdge(3, 2);
78
79     cout << "Following is Breadth First Traversal "
80         << "(starting from vertex 2) \n";
81     g.BFS(2);
82
83     return 0;
84 }

```

?



```

PS C:\Users\A> cd "e:\C++\" ; if ($?) { g++ Bfs.cpp -o Bfs } ; if ($?) { .\Bfs }
Nazmul_Zaman - 55
Following is Breadth First Traversal (starting from vertex 2)
2 0 1 3
PS E:\C++>

```

Figure 2: OUTPUT

5 Conclusion

In this lab report we learn about different type of tree algorithm and searching algorithm method like (BFS) that help us to find we can search for a given key in moderate time (quicker than Linked List and slower than arrays and also searching algorithm for finding the shortest path by learning those method we already clear our deep knowledge .

6 References

1.https://www.w3schools.com/algorithm/algorithm_intro.asp

2.https://www.javatpoint.com/algorithm_intro.asp

3.<https://github.com/ZamanNazmul>