# Data Structures and Algorithms

**Lab Journal - Lab 11**

Name: Saad Ahmad

Enrollment #: 01-134222-130

Class/Section: BSCS(3A)

**Task 1 :**

Open and run your implementation of Adjacency Matrix of Graphs (Task 2 of last lab) from previous lab. Write down the following functions in the class Graph.

**Exercise 1**

|  |
| --- |
| Write a function named DepthFirstSearch() in the class for the implementation of DFS traversal. In main insert edges to make a valid graph and then call this function. |

**Solution :**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val)

{

data = val;

left = NULL;

right = NULL;

}

};

class BinaryTree

{

private:

Node\* root;

public:

Node\* insert(Node\* node, int key)

{

if (node == nullptr)

{

return new Node(key);

}

else

{

if (key < node->data)

{

node->left = insert(node->left, key);

}

else

{

node->right = insert(node->right, key);

}

}

return node;

}

BinaryTree()

{

root = NULL;

}

void dfs(Node\* node)

{

if (node != nullptr)

{

dfs(node->left);

cout << node->data << " ";

dfs(node->right);

}

}

void insert(int key)

{

root = insert(root, key);

}

void depthFirstSearch()

{

dfs(root);

cout << endl;

}

};

int main() {

BinaryTree tree;

tree.insert(10);

tree.insert(20);

tree.insert(3);

tree.insert(40);

tree.insert(50);

tree.insert(70);

tree.insert(5);

cout << "DFS Traversal: ";

tree.depthFirstSearch();

return 0;

}

**Output :**

**A screenshot of a computer

Description automatically generated**

**Exercise 2**

|  |
| --- |
| Write another function named BreadthFirstSearch() in your previous code for the implementation of BFS traversal in graph.  In main insert edges to make a valid graph and then call this function. |

**Solution :**

#include <iostream>

#include <queue>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val)

{

data = val;

left = NULL;

right = NULL;

}

};

class BinaryTree {

private:

Node\* root;

public:

Node\* insert(Node\* node, int key)

{

if (node == nullptr)

{

return new Node(key);

}

else {

if (key < node->data)

{

node->left = insert(node->left, key);

}

else

{

node->right = insert(node->right, key);

}

}

return node;

}

BinaryTree() {

root = NULL;

}

void dfs(Node\* node) {

if (node != nullptr)

{

dfs(node->left);

cout << node->data << " ";

dfs(node->right);

}

}

void BreadthFirstSearch() {

if (root == nullptr)

return;

queue<Node\*> q;

q.push(root);

while (!q.empty()) {

Node\* current = q.front();

q.pop();

cout << current->data << " ";

if (current->left != nullptr)

q.push(current->left);

if (current->right != nullptr)

q.push(current->right);

}

cout << endl;

}

void insert(int key) {

root = insert(root, key);

}

void depthFirstSearch() {

dfs(root);

cout << endl;

}

};

int main() {

BinaryTree tree;

tree.insert(10);

tree.insert(20);

tree.insert(3);

tree.insert(40);

tree.insert(50);

tree.insert(70);

tree.insert(5);

cout << "DFS Traversal: ";

tree.depthFirstSearch();

cout << "BFS Traversal: ";

tree.BreadthFirstSearch();

return 0;

}

**Output :**

**A screenshot of a computer

Description automatically generated**

+++++++++++++++++++++++++