Lab 5

Introduction:

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. It uses a queue (First In First Out) instead of a stack and it checks whether a vertex has been discovered before enqueueing the vertex.

Relevant Lecture Readings:

1. Revise Lecture given on BFS.

2. Objective of the Experiment

After completing this lab, the student should be able to:

- Clearly understand Implementation of Breadth First Search.

3. Concept Map

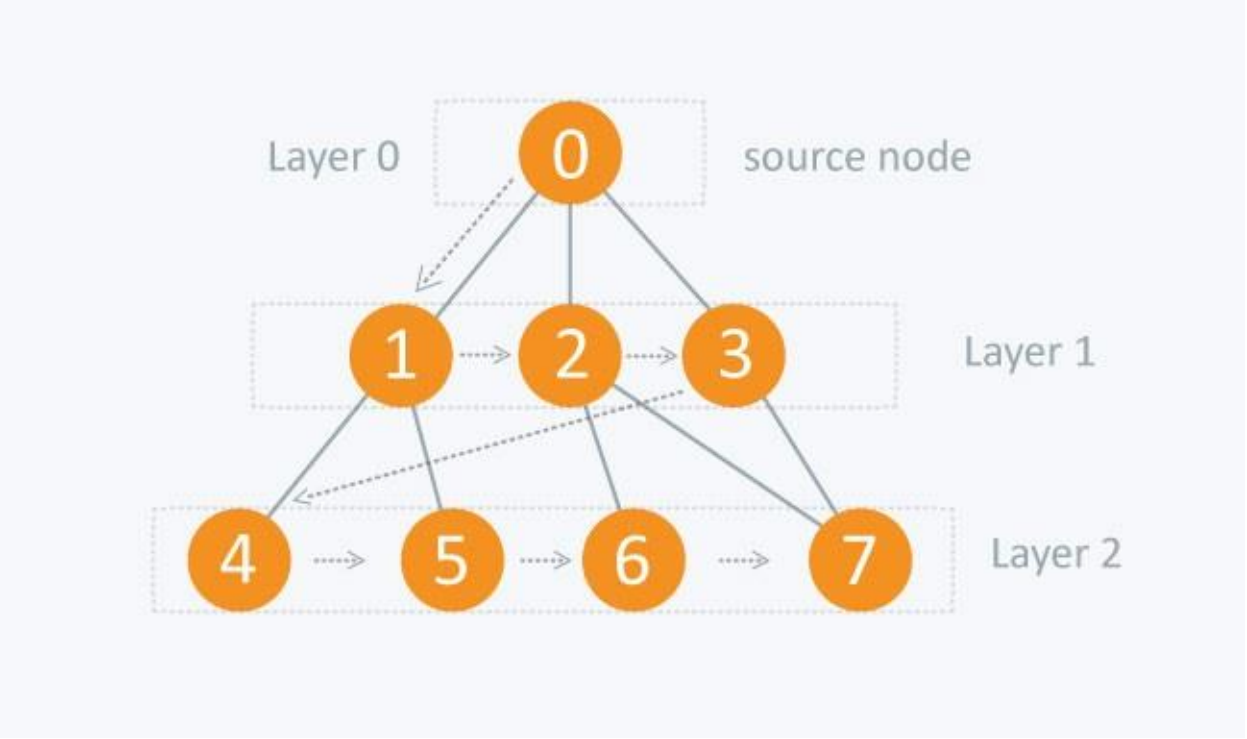
3.1 Breadth First Search (BFS)

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbour nodes (nodes which are directly
connected to source node). You must then move towards the next-level neighbour

nodes. As the name BFS suggests, you are required to traverse the graph breadthwise as

follows:

1. First move horizontally and visit all the nodes of the current layer.
2. Move to the next layer.



Pseudocode:

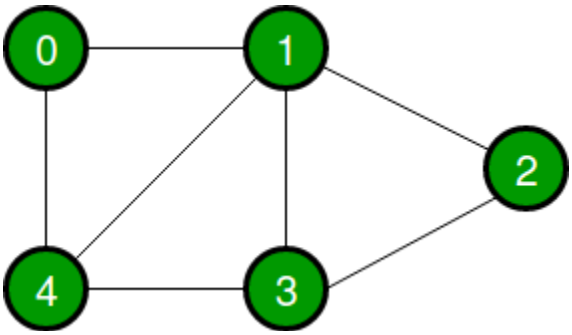
```
BFS (G, s)                                  //Where G is the graph and s is the source node let Q be queue.
        Q.enqueue( s ) //Inserting s in queue until all its neighbor vertices are marked.

        mark s as visited. while (Q is not
        empty)
          //Removing that vertex from queue, whose neighbor will be visited now v = Q.dequeue( )

            //processing all the neighbours of v for all
            neighbours w of v in Graph G
                    if w is not visited
                       Q.enqueue( w )//Stores w in Q to further visit its neighbor mark w as visited.
```

4. Task:1

Attempt the following questions.

4.1 Problem description

Write a program that will implement the following graph in python.



5. Procedure & Tools

5.1 Tools

Anaconda, python language

5.2 Walkthrough Task [Expected time = 40 mins]

5.2.1 Implementation of BFS in c++.

```
#include<iostream>
#include <list>

using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph
{
    int V; // No. of vertices

    // Pointer to an array containing adjacency
    // lists
    list<int> *adj;
public:
    Graph(int V); // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // prints BFS traversal from a given source s
    void BFS(int s);
};
```

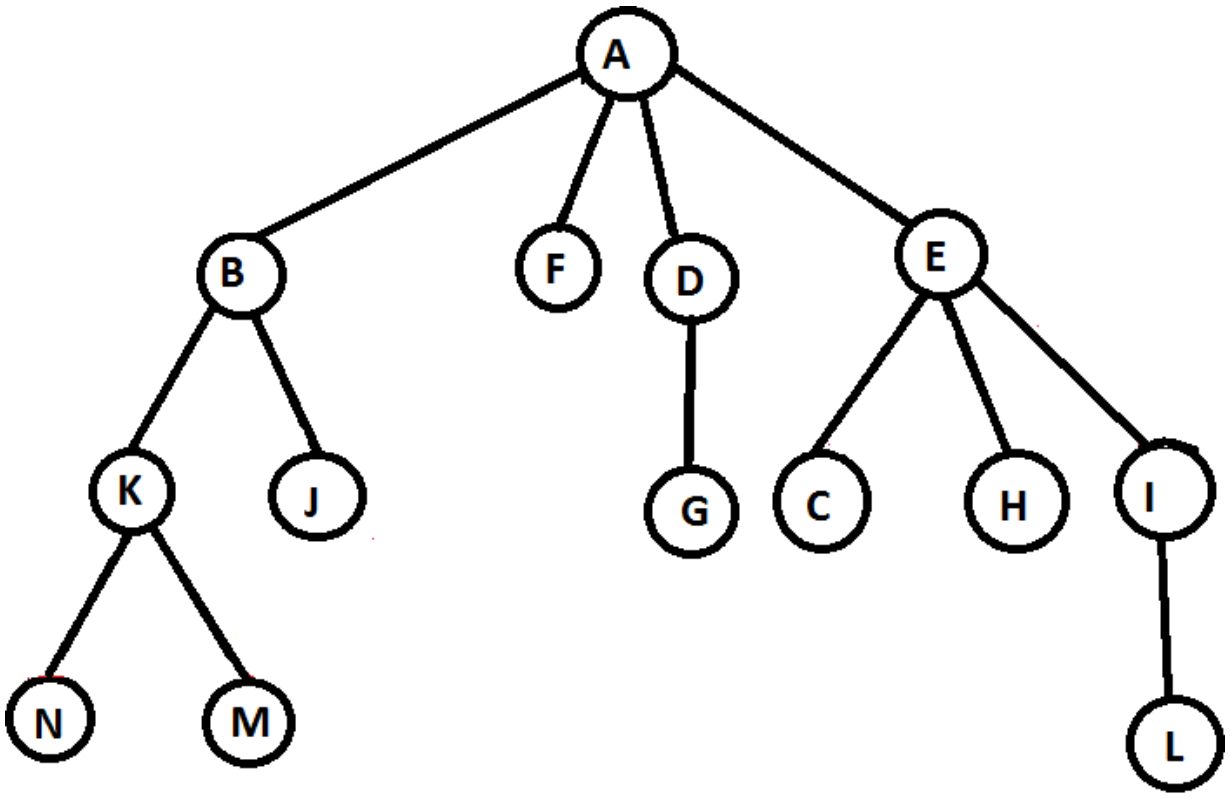
```cpp
Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::BFS(int s)
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;
    // Create a queue for BFS
    list<int> queue;
    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue.push_back(s);
    // 'i' will be used to get all adjacent
    // vertices of a vertex
    list<int>::iterator i;
    // vertices of a vertex
    list<int>::iterator i;

    while(!queue.empty())
    {
        // Dequeue a vertex from queue and print it
        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (i = adj[s].begin(); i != adj[s].end(); ++i)
        {
            if (!visited[*i])
            {
                visited[*i] = true;
                queue.push_back(*i);
            }
        }
    }
}
```

```
// Driver program to test methods of graph class
int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
         << "(starting from vertex 2) \n";
    g.BFS(2);

    return 0;
}
```

6. Tasks

6.1 Task 2 [Expected time = 40 mins]

You are given the following tree whose starting node is **A** and Goal node is **G**



You are required to implement the following graph in python and then apply the following search(Stop the search when goal is achieved)

a) Breadth First Search using Queue.

1.1 Task3: Implement priority Queue.

1.2 Outcomes.

After completing this lab, students will know the implementation of BFS in Python. Students will be able to solve problems using BFS.