THE UNIVERSITY
OF LAHORE
ISLAMABAD
CAMPUS

Data Structures & Algorithms (CS09203)

Lab Report

Name:	Muhammad Umer
Registration #:	CSU-F16-104
Lab Report #:	10
Dated:	18-06-2018
Submitted To:	Mr. Usman Ahmed

The University of Lahore, Islamabad Campus
Department of Computer Science & Information Technology

Experiment # 1
Implement BFS (Breath First Search) on the given graph.

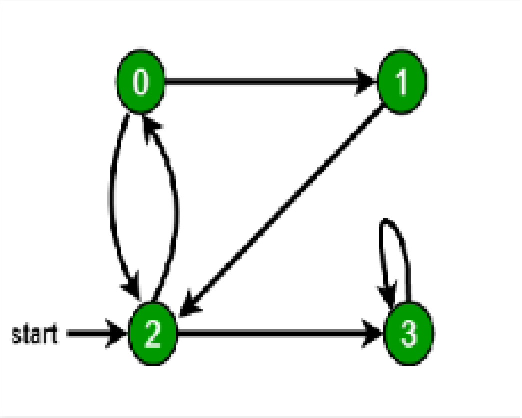
Objective
To understand and implement the Breath First Search on the graph with different cycles.

Software Tool
1. Sublime Text Editor
2. Dev C++
3. Window 7 (32 Bit)

1 Theory

Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.

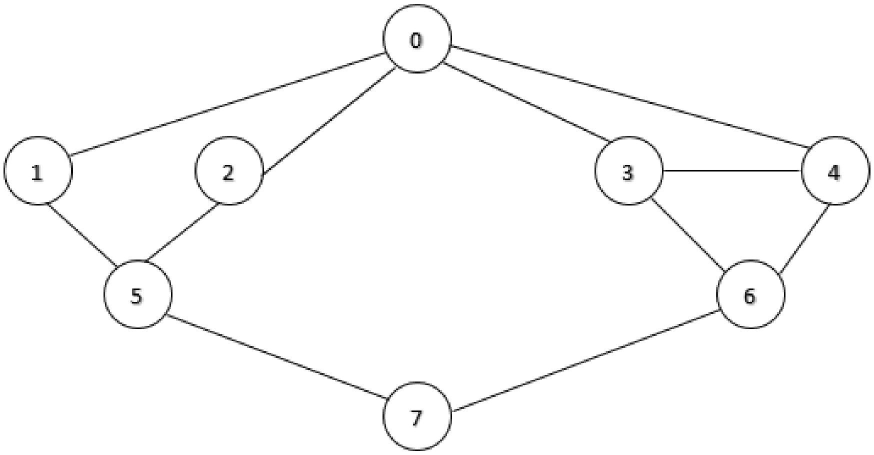
For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we dont mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Breadth First Traversal of the following graph is 2, 0, 3, 1.



2 Task

2.1 Procedure: Task 1 Implement BFS on graph

Implement the BFS Breath First Search on the following graph:



```
#include<iostream>
#include <list>
using namespace std;
```

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

    void addEdge(int v, int w)
        {
        adj[v].push_back(w);
        adj[w].push_back(v);
        }

        void printGraph(){
                list<int>::iterator v;
                for(int i=0; i<V; i++){
                        cout << "\n Adjacency list of vertex "<< i << "\n 
                        for(v = adj[i].begin(); v != adj[i].end(); v++){
                                cout << " -> " << *v;
                        }
                        cout<<"\n";
                }
                cout<<"\n\n";
        }

    void BFS(int s)
    {
            bool *visited = new bool[V];
            for(int i = 0; i < V; i++)
                visited[i] = false;

            list<int> queue;

            visited[s] = true;
            queue.push_back(s);
            list<int>::iterator i;

            while(!queue.empty())
            {
                s = queue.front();
                cout << s << " ";
                queue.pop_front();

                for (i = adj[s].begin(); i != adj[s].end(); ++i)
                {
                    if (!visited[*i])
                    {
                        visited[*i] = true;
                        queue.push_back(*i);
                    }
                }
        }
        }
};

int main()
{
    Graph g(8);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(0, 3);
    g.addEdge(0, 4);
    g.addEdge(1, 5);
    g.addEdge(2, 5);
    g.addEdge(3, 4);
    g.addEdge(3, 6);
    g.addEdge(4, 6);
    g.addEdge(5, 7);
    g.addEdge(6, 7);

        g.printGraph();
    cout << "Following is Breadth First Traversal " << "(starting from vert
    g.BFS(0);

    return 0;
```

```
}
```

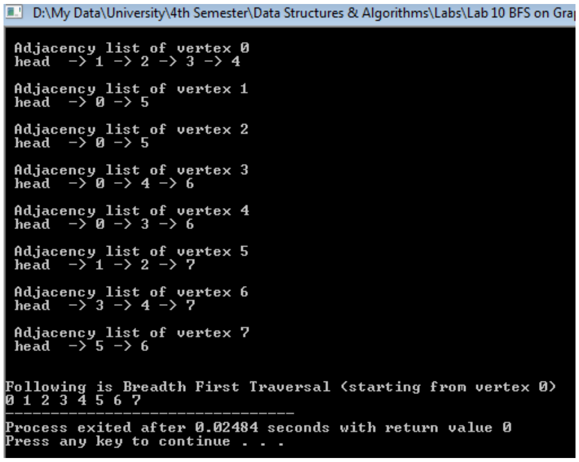
Output: Consider the Figure 1 for the output of the above code in the end of this document.

Source Code
https://goo.gl/ccBvqK

3 Conclusion

Graphs are used to represent many real life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Breath First Search algorithm can be implemented on graphs to visit every node in the shortest path, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.

____
(Concerned Teacher/Lab Engineer)


D:\My Data\University\4th Semester\Data Structures & Algorithms\Labs\Lab 10 BFS on Gra

```
 Adjacency list of vertex 0
 head  -> 1 -> 2 -> 3 -> 4

 Adjacency list of vertex 1
 head  -> 0 -> 5

 Adjacency list of vertex 2
 head  -> 0 -> 5

 Adjacency list of vertex 3
 head  -> 0 -> 4 -> 6

 Adjacency list of vertex 4
 head  -> 0 -> 3 -> 6

 Adjacency list of vertex 5
 head  -> 1 -> 2 -> 7

 Adjacency list of vertex 6
 head  -> 3 -> 4 -> 7

 Adjacency list of vertex 7
 head  -> 5 -> 6


Following is Breadth First Traversal (starting from vertex 0)
0 1 2 3 4 5 6 7
--------------------------------
Process exited after 0.02484 seconds with return value 0
Press any key to continue . . .
```

Figure 1: Breath First Search implementation on graph