Subject: Data Structures and Algorithms Lab

Assignment No. 06

Roll No: **21118**

Date: 24-May-2021

Problem Statement:

Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and perform DFS and BFS on that.

Objectives:

- To understand about graph data structure
- Learn about various graph representations in computer memory
- To perform two most popular ways to traverse the graph:
    - Breadth First Traversal
    - Depth First Traversal

Outcomes:

- Implement adjacency list presentation of graph
- Implement depth first search and breadth first search, as a ways to traverse the graph

Hardware Requirements:

- Manufacturer: Acer
- System Type: x-64 based PC
- Memory and Processor: Intel core i5-8265U@1.60GHz, 8 GB DDR4 RAM, 512 GB SSD

Software Requirements:

- Operating System: Windows-10 (Home Edition)
- IDE: eclipse (2020 – Edition)
- Compiler: GCC (version 6.3.0)

Theory:

1. Graphs are mathematical abstractions that are useful for solving many types of problems in computer science.
2. Any graph finite number of vertices and edges.
3. There are two major ways to represent graphs in computer memory. One can use either one depends on situation and problem.
    a. Adjacency Matrix Representation – It is time efficient to check edge between pair of vertices but it is space inefficient
    b. Adjacency List Representation – It is space efficient but not time efficient when we want to check edge between pair of vertices.
4. Graph traversal is one of the majorly performed operation on graph and it has wide number of applications. There are two main types of graph traversals (both methods have time complexity O(n)):
    a. Breadth-First-Traversal: In this method we explore nodes level wise and hence this method is useful for finding shortest paths in undirected graphs
    b. Depth-First-Traversal: In this method we use concept of backtracking. It has other applications like topological ordering etc.

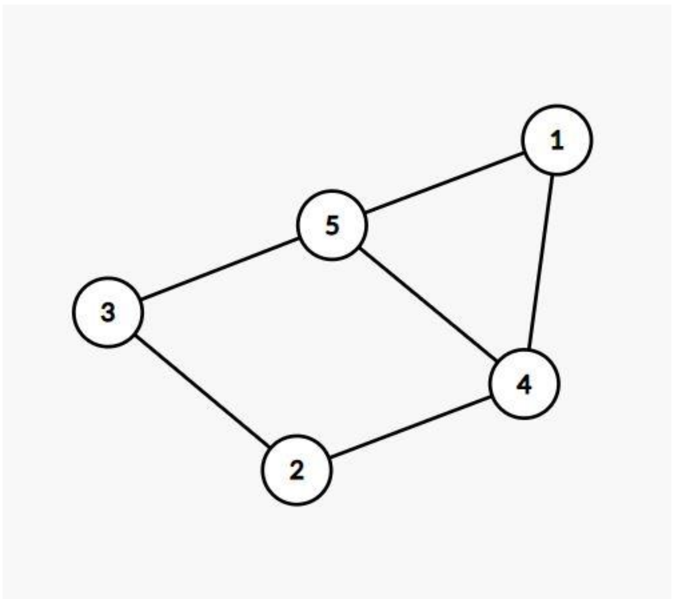


Fig: undirected graph
BFS (1 as source vertex): 1 5 4 3 2
DFS (1 as source vertex): 1 5 3 2 5

ADT:

```
class Node {
private:
	int data;
	Node *next;
public:
	Node(int x = 0); // constructor
	friend class Graph;
};

class Graph {
private:
        int vertices, edges;
        Node **adj_list;

        Node* Insert(Node *&head, int x);
public:
        Graph(int n, int m); // constructor
        void addEdge(int u, int v); // add edge in graph
        void PrintGraph();
        void bfs(int src); // Breadth first traversal
        void dfs(int src); // Depth first traversal
};
```

<u>Pseudocodes:</u>

Depth first traversal:

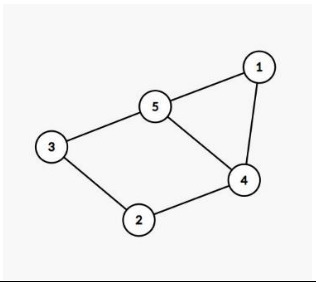
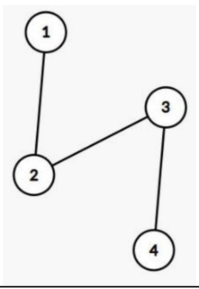
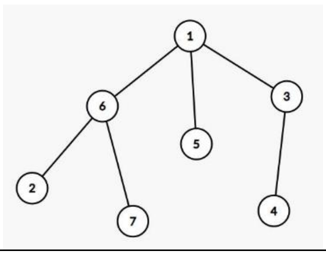
```
procedure dfs(G, v) is
    label v as explored
    for all edges e in G.incidentEdges(v) do
        if edge e is unexplored then
            w ← G.adjacentVertex(v, e)
            if vertex w is unexplored then
                label e as a discovered edge
                recursively call dfs(G, w)
```

Breadth first traversal:

```
procedure bfs(G, v) is
    create a queue Q
    enqueue v onto Q
    mark v visited
    while Q is not empty do
        w ← Q.dequeue()
        // process w
        for all edges e in G.adjacentEdges(w) do
            x ← G.adjacentVertex(w, e)
            if x is not marked then
                mark x
                enqueue x onto Q
```

<u>Testcases</u>: (1-based indexing)

Testcase No.	Input	Expected Output	Actual Output	Verdict

1	Vertices: 5 Edges: 6 1 5 1 4 4 5 5 3 2 4 2 3  	BFS: 1 5 4 3 2 DFS: 1 5 3 2 4	BFS: 1 5 4 3 2 DFS: 1 5 3 2 4	Passed
2	Vertices: 4 Edges: 3 1 2 2 3 3 4  	BFS: 1 2 3 4 DFS: 1 2 3 4	BFS: 1 2 3 4 DFS: 1 2 3 4	Passed
3	Vertices: 7 Edges: 6 1 3 1 5 1 6 2 6 3 4 6 7  	BFS: 1 6 5 3 2 7 4 DFS: 1 6 2 7 5 3 4	BFS: 1 6 5 3 2 7 4 DFS: 1 6 2 7 5 3 4	Passed

Analysis of algorithms:

No.	Algorithm	Time complexity	Space complexity

1	Breadth-first-traversal	O(n)	O(n) for queue data structure
2	Depth-first-traversal	O(n)	O(n) for stack data structure

<u>Applications:</u>
Graph data structure has wide range of applications. Applications of graph traversal are mentioned below.

- finding all vertices within one connected component
- finding the shortest path between two vertices
- testing a graph for bipartiteness
- serialization/deserialization of a binary tree vs serialization in sorted order (allows the tree to be re-constructed in an efficient manner)
- maze generation algorithms
- analysis of networks and relationships.

<u>Conclusion:</u>
I have learned about representation of graph data structure in computer memory and graph traversals in this assignment. Also implemented two graph traversal techniques and their real world applications.