ADS ASSIGNMENT – 3

Q1. Write a program to implement Depth FirstSearch.

Ans.

```
#include <iostream>

#include <vector>
using namespace std;
class Graph {
    int V; // No of vertice
    vector<vector<int>> adj; // Adjacency list
    void DFSUtil(int v, vector<bool> &visited) {
        visited[v] = true;
        cout << v << " ";
        // Recur for all adjacent vertices
        for (int u : adj[v]) {   //Range Based loop
            if (!visited[u]) {
                DFSUtil(u, visited);        }    }
    }
public:
    Graph(int V) {
        this->V = V;
        adj.resize(V);
    }
    void addEdge(int v, int w) {
        adj[v].push_back(w); // Directed graph
        // For undirected graph, also add: adj[w].push_back(v);
    }
    void DFS(int start) {
        vector<bool> visited(V, false);
        DFSUtil(start, visited);
    }
};
int main() {
    Graph g(6);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
    g.addEdge(1, 4);
    g.addEdge(2, 5);
    cout << "Depth First Search starting from node 0:\n";
    g.DFS(0);
    return 0;
}
OUTPUT:
Depth First Search starting from node 0:
0 1 3 4 2 5
```

Q2. Write a program to implement Breadth First Search.

Ans.

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

class Graph {
    int V; // No of vertices
    vector<vector<int>> adj; // Adjacency list

public:
    Graph(int V) {
        this->V = V;
        adj.resize(V);
    }

    void addEdge(int v, int w) {
        adj[v].push_back(w); // Directed graph
        // For undirected graph, also add: adj[w].push_back(v);
    }

    void BFS(int start) {
        vector<bool> visited(V, false);
        queue<int> q;

        visited[start] = true;
        q.push(start);

        while (!q.empty()) {
            int v = q.front();
            q.pop();
            cout << v << " ";
            // Visit all unvisited neighbors
            for (int u : adj[v]) { //
                if (!visited[u]) {
                    visited[u] = true;
                    q.push(u);
                }
            }
        }
    }
};
int main() {
    Graph g(6);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 3);
```

```
    g.addEdge(1, 4);
    g.addEdge(2, 5);
    cout << "Breadth First Search starting from node 0:\n";
    g.BFS(0);
    return 0;
}
OUTPUT:
Breadth First Search starting from node 0:
0 1 2 3 4 5
```