ASSINGMENT 9

A graph G is defined as a pair (V, E) where V is a set of nodes/vertices and E is a set of edges connecting

pairs of vertices. Graphs may be directed or undirected and may have weighted or unweighted edges.

They can be represented using an adjacency matrix, adjacency list, or edge list.

Write a program to implement the following graph algorithms:

1. Breadth First Search (BFS)

2. Depth First Search (DFS)

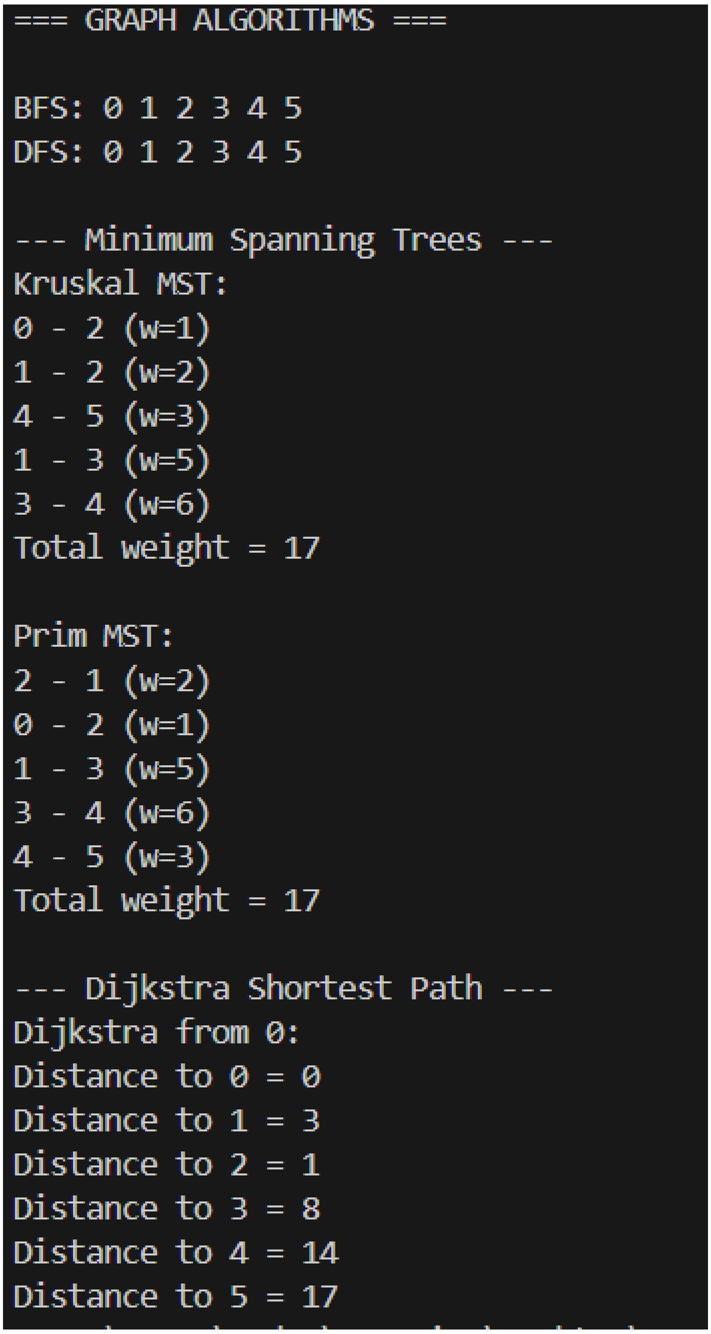
3. Minimum Spanning Tree (Kruskal and Prim)

4. Dijkstra's Shortest Path Algorithm

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

class Graph {
public:
    int V;
    vector<vector<pair<int,int>>> adj; // node, weight

    Graph(int v) {
        V = v;
    adj.resize(V);
}

void addEdge(int u, int v, int w=1, bool undirected=true) {
    adj[u].push_back({v, w});
    if (undirected)
        adj[v].push_back({u, w});
}

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

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

    cout << "BFS: ";
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (auto &nbr : adj[node]) {
            if (!visited[nbr.first]) {
                visited[nbr.first] = true;
                q.push(nbr.first);
            }
        }
    }
    cout << endl;
}

// ---------------- DFS ----------------
void dfsUtil(int node, vector<bool>& visited) {
    visited[node] = true;
    cout << node << " ";

    for (auto &nbr : adj[node]) {
        if (!visited[nbr.first])
            dfsUtil(nbr.first, visited);
    }
}

void DFS(int start) {
    vector<bool> visited(V, false);
    cout << "DFS: ";
    dfsUtil(start, visited);
    cout << endl;
}

// ---------------- Kruskal (MST) ----------------
struct Edge {
    int u, v, w;
};
```

```
int findSet(int x, vector<int>& parent) {
    if (parent[x] == x)
        return x;
    return parent[x] = findSet(parent[x], parent);
}

void unionSet(int u, int v, vector<int>& parent, vector<int>& rank) {
    u = findSet(u, parent);
    v = findSet(v, parent);

    if (rank[u] < rank[v])
        parent[u] = v;
    else if (rank[v] < rank[u])
        parent[v] = u;
    else {
        parent[v] = u;
        rank[u]++;
    }
}

void Kruskal() {
    vector<Edge> edges;

    // Convert adjacency list to edge list
    for (int u = 0; u < V; u++) {
        for (auto &p : adj[u]) {
            int v = p.first, w = p.second;
            if (u < v) edges.push_back({u, v, w});
        }
    }

    sort(edges.begin(), edges.end(),
         [](Edge a, Edge b) { return a.w < b.w; });

    vector<int> parent(V), rank(V, 0);
    for (int i = 0; i < V; i++)
        parent[i] = i;

    cout << "Kruskal MST:\n";
    int mst_cost = 0;

    for (auto &e : edges) {
        int pu = findSet(e.u, parent);
        int pv = findSet(e.v, parent);

        if (pu != pv) {
            cout << e.u << " - " << e.v << " (w=" << e.w << ")\n";
            mst_cost += e.w;
            unionSet(pu, pv, parent, rank);
        }
    }

    cout << "Total weight = " << mst_cost << "\n";
}

// ---------------- Prim (MST) ----------------
```

```
void Prim(int start) {
    vector<int> key(V, 1e9), parent(V, -1);
    vector<bool> inMST(V, false);

    key[start] = 0;
    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
    pq.push({0, start});

    cout << "Prim MST:\n";

    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        inMST[u] = true;

        for (auto &p : adj[u]) {
            int v = p.first, w = p.second;

            if (!inMST[v] && w < key[v]) {
                key[v] = w;
                parent[v] = u;
                pq.push({w, v});
            }
        }
    }

    int cost = 0;
    for (int v = 0; v < V; v++) {
        if (parent[v] != -1) {
            cout << parent[v] << " - " << v << " (w=" << key[v] << ")\n";
            cost += key[v];
        }
    }

    cout << "Total weight = " << cost << "\n";
}

// ---------------- Dijkstra ----------------
void Dijkstra(int src) {
    vector<int> dist(V, 1e9);
    dist[src] = 0;

    priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
    pq.push({0, src});

    while (!pq.empty()) {
        auto top = pq.top(); pq.pop();
        int d = top.first;
        int u = top.second;

        if (d > dist[u]) continue;

        for (auto &p : adj[u]) {
            int v = p.first, w = p.second;

            if (dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                    pq.push({dist[v], v});
                }
            }
        }

        cout << "Dijkstra from " << src << ":\n";
        for (int i = 0; i < V; i++)
            cout << "Distance to " << i << " = " << dist[i] << endl;
    }
};

int main() {
    int V = 6;
    Graph g(V);

    // Sample weighted graph
    g.addEdge(0, 1, 4);
    g.addEdge(0, 2, 1);
    g.addEdge(2, 1, 2);
    g.addEdge(1, 3, 5);
    g.addEdge(2, 3, 8);
    g.addEdge(3, 4, 6);
    g.addEdge(4, 5, 3);

    cout << "=== GRAPH ALGORITHMS ===\n\n";

    g.BFS(0);
    g.DFS(0);

    cout << "\n--- Minimum Spanning Trees ---\n";
    g.Kruskal();
    cout << endl;
    g.Prim(0);

    cout << "\n--- Dijkstra Shortest Path ---\n";
    g.Dijkstra(0);

    return 0;
}
```



```
=== GRAPH ALGORITHMS ===

BFS: 0 1 2 3 4 5
DFS: 0 1 2 3 4 5

--- Minimum Spanning Trees ---
Kruskal MST:
0 - 2 (w=1)
1 - 2 (w=2)
4 - 5 (w=3)
1 - 3 (w=5)
3 - 4 (w=6)
Total weight = 17

Prim MST:
2 - 1 (w=2)
0 - 2 (w=1)
1 - 3 (w=5)
3 - 4 (w=6)
4 - 5 (w=3)
Total weight = 17

--- Dijkstra Shortest Path ---
Dijkstra from 0:
Distance to 0 = 0
Distance to 1 = 3
Distance to 2 = 1
Distance to 3 = 8
Distance to 4 = 14
Distance to 5 = 17
```