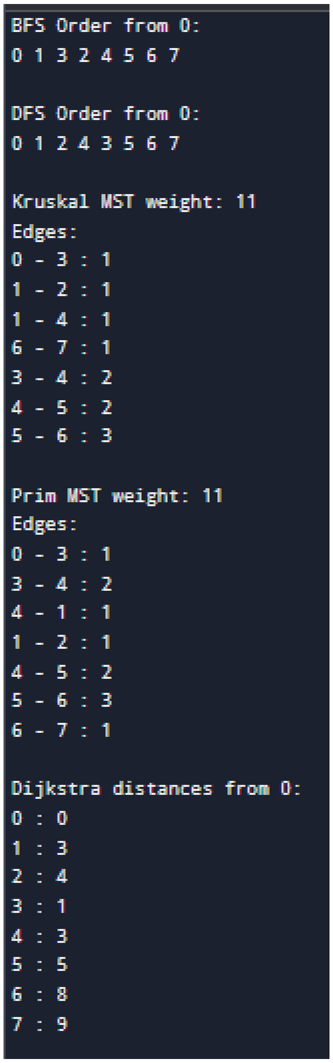
Assignment 9 (Graphs)

Name-Ujjwal Pant   Batch-2C24   Rollno. 1024030370

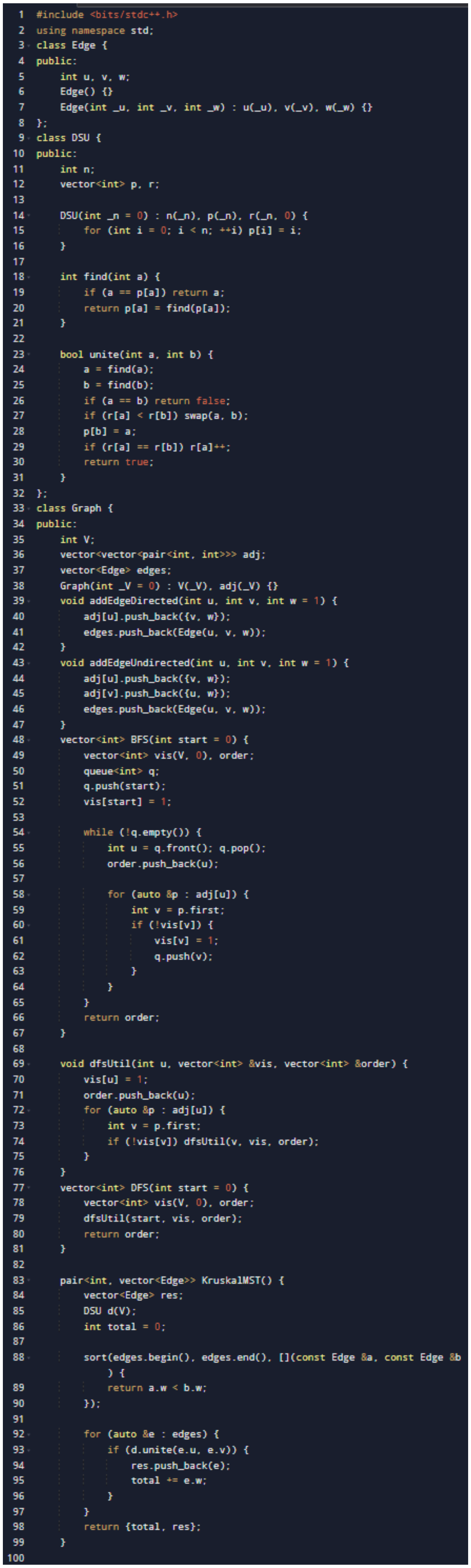
A graph G is defined as a pair (V, E) where V is a set of nodes/vertices and E is a set of edges connecting pairs of vertices. Graphs may be directed or undirected and may have weighted or unweighted edges. They can be represented using an adjacency matrix, adjacency list, or edge list.

Write a program to implement the following graph algorithms:

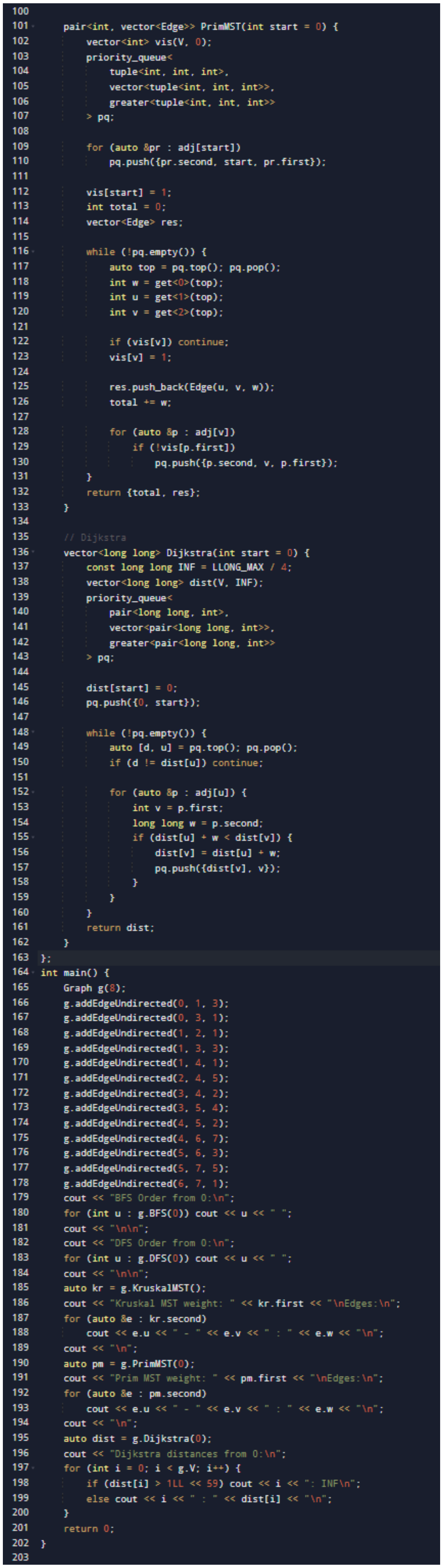
1. Breadth First Search (BFS)
2. Depth First Search (DFS)
3. Minimum Spanning Tree (Kruskal and Prim)
4. Dijkstra's Shortest Path Algorithm



```
BFS Order from 0:
0 1 3 2 4 5 6 7

DFS Order from 0:
0 1 2 4 3 5 6 7

Kruskal MST weight: 11
Edges:
0 - 3 : 1
1 - 2 : 1
1 - 4 : 1
6 - 7 : 1
3 - 4 : 2
4 - 5 : 2
5 - 6 : 3

Prim MST weight: 11
Edges:
0 - 3 : 1
3 - 4 : 2
4 - 1 : 1
1 - 2 : 1
4 - 5 : 2
5 - 6 : 3
6 - 7 : 1

Dijkstra distances from 0:
0 : 0
1 : 3
2 : 4
3 : 1
4 : 3
5 : 5
6 : 8
7 : 9
```



```cpp
#include <bits/stdc++.h>
using namespace std;
class Edge {
public:
    int u, v, w;
    Edge() {}
    Edge(int _u, int _v, int _w) : u(_u), v(_v), w(_w) {}
};
class DSU {
public:
    int n;
    vector<int> p, r;

    DSU(int _n = 0) : n(_n), p(_n), r(_n, 0) {
        for (int i = 0; i < n; ++i) p[i] = i;
    }

    int find(int a) {
        if (a == p[a]) return a;
        return p[a] = find(p[a]);
    }

    bool unite(int a, int b) {
        a = find(a);
        b = find(b);
        if (a == b) return false;
        if (r[a] < r[b]) swap(a, b);
        p[b] = a;
        if (r[a] == r[b]) r[a]++;
        return true;
    }
};
class Graph {
public:
    int V;
    vector<vector<pair<int, int>>> adj;
    vector<Edge> edges;
    Graph(int _V = 0) : V(_V), adj(_V) {}
    void addEdgeDirected(int u, int v, int w = 1) {
        adj[u].push_back({v, w});
        edges.push_back(Edge(u, v, w));
    }
    void addEdgeUndirected(int u, int v, int w = 1) {
        adj[u].push_back({v, w});
        adj[v].push_back({u, w});
        edges.push_back(Edge(u, v, w));
    }
    vector<int> BFS(int start = 0) {
        vector<int> vis(V, 0), order;
        queue<int> q;
        q.push(start);
        vis[start] = 1;

        while (!q.empty()) {
            int u = q.front(); q.pop();
            order.push_back(u);

            for (auto &p : adj[u]) {
                int v = p.first;
                if (!vis[v]) {
                    vis[v] = 1;
                    q.push(v);
                }
            }
        }
        return order;
    }

    void dfsUtil(int u, vector<int> &vis, vector<int> &order) {
        vis[u] = 1;
        order.push_back(u);
        for (auto &p : adj[u]) {
            int v = p.first;
            if (!vis[v]) dfsUtil(v, vis, order);
        }
    }
    vector<int> DFS(int start = 0) {
        vector<int> vis(V, 0), order;
        dfsUtil(start, vis, order);
        return order;
    }

    pair<int, vector<Edge>> KruskalMST() {
        vector<Edge> res;
        DSU d(V);
        int total = 0;

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

        for (auto &e : edges) {
            if (d.unite(e.u, e.v)) {
                res.push_back(e);
                total += e.w;
            }
        }
        return {total, res};
    }

```



```cpp

    pair<int, vector<Edge>> PrimMST(int start = 0) {
        vector<int> vis(V, 0);
        priority_queue<
            tuple<int, int, int>,
            vector<tuple<int, int, int>>,
            greater<tuple<int, int, int>>
        > pq;

        for (auto &pr : adj[start])
            pq.push({pr.second, start, pr.first});

        vis[start] = 1;
        int total = 0;
        vector<Edge> res;

        while (!pq.empty()) {
            auto top = pq.top(); pq.pop();
            int w = get<0>(top);
            int u = get<1>(top);
            int v = get<2>(top);

            if (vis[v]) continue;
            vis[v] = 1;

            res.push_back(Edge(u, v, w));
            total += w;

            for (auto &p : adj[v])
                if (!vis[p.first])
                    pq.push({p.second, v, p.first});
        }
        return {total, res};
    }

    // Dijkstra
    vector<long long> Dijkstra(int start = 0) {
        const long long INF = LLONG_MAX / 4;
        vector<long long> dist(V, INF);
        priority_queue<
            pair<long long, int>,
            vector<pair<long long, int>>,
            greater<pair<long long, int>>
        > pq;

        dist[start] = 0;
        pq.push({0, start});

        while (!pq.empty()) {
            auto [d, u] = pq.top(); pq.pop();
            if (d != dist[u]) continue;

            for (auto &p : adj[u]) {
                int v = p.first;
                long long w = p.second;
                if (dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                    pq.push({dist[v], v});
                }
            }
        }
        return dist;
    }
};
int main() {
    Graph g(8);
    g.addEdgeUndirected(0, 1, 3);
    g.addEdgeUndirected(0, 3, 1);
    g.addEdgeUndirected(1, 2, 1);
    g.addEdgeUndirected(1, 3, 3);
    g.addEdgeUndirected(1, 4, 1);
    g.addEdgeUndirected(2, 4, 5);
    g.addEdgeUndirected(3, 4, 2);
    g.addEdgeUndirected(3, 5, 4);
    g.addEdgeUndirected(4, 5, 2);
    g.addEdgeUndirected(4, 6, 7);
    g.addEdgeUndirected(5, 6, 3);
    g.addEdgeUndirected(5, 7, 5);
    g.addEdgeUndirected(6, 7, 1);
    cout << "BFS Order from 0:\n";
    for (int u : g.BFS(0)) cout << u << " ";
    cout << "\n\n";
    cout << "DFS Order from 0:\n";
    for (int u : g.DFS(0)) cout << u << " ";
    cout << "\n\n";
    auto kr = g.KruskalMST();
    cout << "Kruskal MST weight: " << kr.first << "\nEdges:\n";
    for (auto &e : kr.second)
        cout << e.u << " - " << e.v << " : " << e.w << "\n";
    cout << "\n";
    auto pm = g.PrimMST(0);
    cout << "Prim MST weight: " << pm.first << "\nEdges:\n";
    for (auto &e : pm.second)
        cout << e.u << " - " << e.v << " : " << e.w << "\n";
    cout << "\n";
    auto dist = g.Dijkstra(0);
    cout << "Dijkstra distances from 0:\n";
    for (int i = 0; i < g.V; i++) {
        if (dist[i] > 1LL << 59) cout << i << ": INF\n";
        else cout << i << " : " << dist[i] << "\n";
    }
    return 0;
}

```