**Shortest path**

#include <iostream>

#include <vector>

#include <queue>

#include <limits>

using namespace std;

// Define infinity as a large value

const int INF = numeric\_limits<int>::max();

// Function to perform Dijkstra's algorithm

vector<int> dijkstra(int start, const vector<vector<pair<int, int>>>& adjList) {

    int n = adjList.size();

    vector<int> dist(n, INF); // Distance from start to each node

    vector<bool> visited(n, false); // Track visited nodes

    priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<>> minHeap;

    // Start from the start node

    dist[start] = 0;

    minHeap.push({0, start});

    while (!minHeap.empty()) {

        int u = minHeap.top().second;

        minHeap.pop();

        if (visited[u]) continue;

        visited[u] = true;

        // Explore all adjacent nodes of u

        for (const auto& neighbor : adjList[u]) {

            int v = neighbor.first;

            int weight = neighbor.second;

            if (dist[u] + weight < dist[v]) {

                dist[v] = dist[u] + weight;

                minHeap.push({dist[v], v});

            }

        }

    }

    return dist;

}

int main() {

    int n, m;

    cout << "Enter number of nodes and edges: ";

    cin >> n >> m;

    vector<vector<pair<int, int>>> adjList(n);

    cout << "Enter edges in the format (u v w) where u and v are nodes and w is the weight:\n";

    for (int i = 0; i < m; ++i) {

        int u, v, w;

        cin >> u >> v >> w;

        adjList[u].emplace\_back(v, w);

        adjList[v].emplace\_back(u, w); // If the graph is undirected

    }

    int start;

    cout << "Enter the start node: ";

    cin >> start;

    vector<int> distances = dijkstra(start, adjList);

    cout << "Shortest distances from node " << start << ":\n";

    for (int i = 0; i < n; ++i) {

        if (distances[i] == INF) {

            cout << "Node " << i << ": INF\n";

        } else {

            cout << "Node " << i << ": " << distances[i] << endl;

        }

    }

    return 0;

}