Modern Operating system And computer Network  
Assignment

**Question:**

**Q.** Write a C++ program to implement **Dijkstra’s Single Source Shortest Path Algorithm** for a graph represented using an **adjacency matrix**.

Number of vertices: 5

Edges:

0 1 4

0 2 8

1 4 6

2 3 2

3 4 10

Source vertex: 0

------------------------------------------------------------------------------------------------------------------

#include <iostream>

#include <vector>

#include <queue>

#include <climits>

using namespace std;

// Function to construct adjacency

vector<vector<vector<int>>> constructAdj(vector<vector<int>>

                             &edges, int V) {

    // adj[u] = list of {v, wt}

    vector<vector<vector<int>>> adj(V);

    for (const auto &edge : edges) {

        int u = edge[0];

        int v = edge[1];

        int wt = edge[2];

        adj[u].push\_back({v, wt});

        adj[v].push\_back({u, wt});

    }

    return adj;

}

// Returns shortest distances from src to all other vertices

vector<int> dijkstra(int V, vector<vector<int>> &edges, int src){

    // Create adjacency list

    vector<vector<vector<int>>> adj = constructAdj(edges, V);

    // Create a priority queue to store vertices that

    // are being preprocessed.

    priority\_queue<vector<int>, vector<vector<int>>,

                   greater<vector<int>>> pq;

    // Create a vector for distances and initialize all

    // distances as infinite

    vector<int> dist(V, INT\_MAX);

    // Insert source itself in priority queue and initialize

    // its distance as 0.

    pq.push({0, src});

    dist[src] = 0;

    // Looping till priority queue becomes empty (or all

    // distances are not finalized)

    while (!pq.empty()){

        // The first vertex in pair is the minimum distance

        // vertex, extract it from priority queue.

        int u = pq.top()[1];

        pq.pop();

        // Get all adjacent of u.

        for (auto x : adj[u]){

            // Get vertex label and weight of current

            // adjacent of u.

            int v = x[0];

            int weight = x[1];

            // If there is shorter path to v through u.

            if (dist[v] > dist[u] + weight)

            {

                // Updating distance of v

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

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

            }

        }

    }

    return dist;

}

// Driver program to test methods of graph class

int main(){

    int V = 5;

    int src = 0;

    // edge list format: {u, v, weight}

    vector<vector<int>> edges = {{0, 1, 4}, {0, 2, 8}, {1, 4, 6},

                                 {2, 3, 2}, {3, 4, 10}};

    vector<int> result = dijkstra(V, edges, src);

    // Print shortest distances in one line

    for (int dist : result)

        cout << dist << " ";

    return 0;

}  
