**NAME = HARSH RAJ**

**COURSE = MCA (1ST YEAR)**

**Enrollment no = 2503B09912**

Write a C++ program to implement **Dijkstra’s Single Source Shortest Path Algorithm** for a given weighted, undirected graph using an **adjacency matrix representation**.

1. **Problem Setup**

* We have **9 vertices** (0 to 8).

#include <iostream>

#include <limits.h>

using namespace std;

#define V 9 // Number of vertices in the graph

// Function to find the vertex with the minimum distance value

int minDistance(int dist[], bool visited[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (!visited[v] && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

// Function to print the final shortest distance from source

void printSolution(int dist[]) {

cout << "Vertex \t Distance from Source" << endl;

for (int i = 0; i < V; i++)

cout << i << " \t\t " << dist[i] << endl;

}

// Dijkstra's Single Source Shortest Path Algorithm

void dijkstra(int graph[V][V], int src) {

int dist[V]; // Output array. dist[i] will hold the shortest distance from src to i

bool visited[V]; // visited[i] will be true if vertex i's shortest distance is finalized

// Initialize all distances as infinite and visited[] as false

for (int i = 0; i < V; i++) {

dist[i] = INT\_MAX;

visited[i] = false;

}

// Distance of source vertex from itself is always 0

dist[src] = 0;

// Find shortest path for all vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum distance vertex from the set of vertices not yet processed

int u = minDistance(dist, visited);

// Mark the picked vertex as processed

visited[u] = true;

// Update dist[v] only if it is not visited, there is an edge from u to v,

// and total weight of path from src to v through u is smaller than current dist[v]

for (int v = 0; v < V; v++)

if (!visited[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

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

}

// Print the constructed distance array

printSolution(dist);

}

int main() {

// Adjacency matrix for the graph (weighted and undirected)

int graph[V][V] = {

{0, 4, 0, 0, 0, 0, 0, 8, 0},

{4, 0, 8, 0, 0, 0, 0, 11, 0},

{0, 8, 0, 7, 0, 4, 0, 0, 2},

{0, 0, 7, 0, 9, 14, 0, 0, 0},

{0, 0, 0, 9, 0, 10, 0, 0, 0},

{0, 0, 4, 14, 10, 0, 2, 0, 0},

{0, 0, 0, 0, 0, 2, 0, 1, 6},

{8, 11, 0, 0, 0, 0, 1, 0, 7},

{0, 0, 2, 0, 0, 0, 6, 7, 0}

};

int source = 0; // You can change the source vertex

dijkstra(graph, source);

return 0;

}

**output**