Assignment No. 12

Name – Manish Namdev Barage

PRN – 22520007

Batch – T7

Problem Statement – From given vertex in weighted connected graph , find shortest path to other vertices using Dijkstra's Algorithm

Code –

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>

using namespace std;

// Define a structure to represent edges in the graph.
struct Edge {
    int destination;
    int weight;
};

// Function to find shortest paths from a given source vertex to all
other vertices using Dijkstra's algorithm.
void dijkstra(vector<vector<Edge>>& graph, int source) {
    int V = graph.size();
    vector<int> distance(V, INT_MAX); // Initialize distances to all
vertices as infinity.
    vector<int> parent(V, -1); // Initialize parent array to keep
track of shortest paths.
    distance[source] = 0; // Distance from the source vertex to itself
is 0.

    priority_queue<pair<int, int>, vector<pair<int, int>>,
greater<pair<int, int>>> pq;
    pq.push(make_pair(0, source));

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

        // Update the distances of adjacent vertices.
        for (const Edge& edge : graph[u]) {
            int v = edge.destination;
            int weight = edge.weight;
            if (distance[u] + weight < distance[v]) {
                distance[v] = distance[u] + weight;
                parent[v] = u;
                pq.push(make_pair(distance[v], v));
            }
        }
    }

    // Print the shortest paths and distances from the source vertex
to all other vertices.
    for (int i = 0; i < V; ++i) {
        if (i != source) {
            cout << "The minimum distance from " << source << " to "
<< i << " = " << distance[i] << ". ";

            int node = parent[i];
            vector<int> path;
            // Exclude the source vertex from the path.
            while (node != -1 && node != source) {
                path.push_back(node);
                node = parent[node];
            }
            // Print the path in reverse order.
            cout << "And the path is: " << source;
            for (int j = path.size() - 1; j >= 0; --j) {
                cout << "->" << path[j];
            }
            cout << "->" << i << endl;
        }
    }
}


int main() {
    // Given graph represented as an adjacency list.
    // vector<vector<Edge>> graph = {
    //     {{1, 3}, {2, 2}},  // Vertex 0
    //     {{3, 1}},          // Vertex 1
    //     {{3, 5}},           // Vertex 2
    //     {{4, 2}},          // Vertex 3
    //     {{}}
    // };

    int numVertices;
    cout << "Enter the number of vertices: ";
    cin >> numVertices;

    // // Create an empty adjacency list.
    vector<vector<Edge>> graph(numVertices);

    // Ask the user for input to create the adjacency list.
    for (int i = 0; i < numVertices; ++i) {
        int numEdges;
        cout << "Enter the number of edges for vertex " << i << ": ";
        cin >> numEdges;

        for (int j = 0; j < numEdges; ++j) {
            int destination, weight;
```

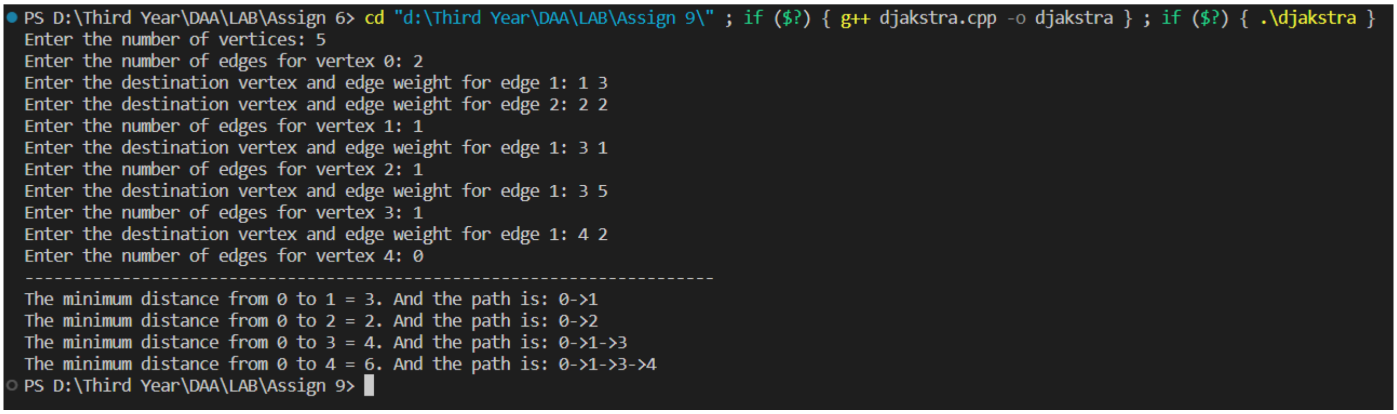
```
                cout << "Enter the destination vertex and edge weight for
edge " << j+1 << ": ";
                cin >> destination >> weight;
                graph[i].push_back({destination, weight});
            }
        }

        cout<<"-----------------------------------------------------------
------------"<<endl;

        int sourceVertex = 0; // Starting vertex for Dijkstra's algorithm.

        // Call Dijkstra's algorithm function with the graph and source
vertex.
        dijkstra(graph, sourceVertex);

        return 0;
}
```



```
PS D:\Third Year\DAA\LAB\Assign 6> cd "d:\Third Year\DAA\LAB\Assign 9\" ; if ($?) { g++ djakstra.cpp -o djakstra } ; if ($?) { .\djakstra }
Enter the number of vertices: 5
Enter the number of edges for vertex 0: 2
Enter the destination vertex and edge weight for edge 1: 1 3
Enter the destination vertex and edge weight for edge 2: 2 2
Enter the number of edges for vertex 1: 1
Enter the destination vertex and edge weight for edge 1: 3 1
Enter the number of edges for vertex 2: 1
Enter the destination vertex and edge weight for edge 1: 3 5
Enter the number of edges for vertex 3: 1
Enter the destination vertex and edge weight for edge 1: 4 2
Enter the number of edges for vertex 4: 0
-----------------------------------------------------------------------
The minimum distance from 0 to 1 = 3. And the path is: 0->1
The minimum distance from 0 to 2 = 2. And the path is: 0->2
The minimum distance from 0 to 3 = 4. And the path is: 0->1->3
The minimum distance from 0 to 4 = 6. And the path is: 0->1->3->4
PS D:\Third Year\DAA\LAB\Assign 9>
```