Name: Snehal Jayprakash Borji
UID: 2023510008
Course: F.Y.M.C.A.
Subject: DS

Practical 07

Aim: Implement Single Source Shortest Path Algorithm, find the shortest path between the pair of vertices given by user.

Coding:

Prac7.cpp

```cpp
#include <iostream>

#include <vector>

#include <queue>

#include <climits>



using namespace std;



class Graph {

public:

   int vertices;

   vector<vector<pair<int, int>>> adjList; // {neighbor, weight}



   Graph(int v) : vertices(v), adjList(v) {}
```

```
    void addEdge(int u, int v, int weight) {
        adjList[u].push_back({v, weight});
        adjList[v].push_back({u, weight}); // Assuming an undirected graph
    }

    vector<int> dijkstra(int startVertex) {
        vector<int> distance(vertices, INT_MAX);
        distance[startVertex] = 0;

        priority_queue<pair<int, int>, vector<pair<int, int>>,
greater<pair<int, int>>> pq;
        pq.push({0, startVertex}); // {distance, vertex}

        while (!pq.empty()) {
            int dist = pq.top().first;
            int currentVertex = pq.top().second;
            pq.pop();

            if (dist > distance[currentVertex]) {
```

```
            // Skip if this is not the shortest path
            continue;
        }

        for (const auto &neighbor : adjList[currentVertex]) {
            int nextVertex = neighbor.first;
            int weight = neighbor.second;

            if (distance[currentVertex] + weight <
distance[nextVertex]) {
                // Update the distance if a shorter path is found
                distance[nextVertex] = distance[currentVertex] +
weight;
                pq.push({distance[nextVertex], nextVertex});
            }
        }
    }

    return distance;
}
```

```
};



int main() {

   int vertices, edges;

   cout << "Enter the number of vertices: ";

   cin >> vertices;



   Graph graph(vertices);



   cout << "Enter the number of edges: ";

   cin >> edges;



   cout << "Enter the edges and weights (u v weight):" << endl;

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

       int u, v, weight;

       cin >> u >> v >> weight;

       graph.addEdge(u, v, weight);

   }
```

```
    int startVertex, endVertex;

    cout << "Enter the source vertex for shortest path: ";

    cin >> startVertex;

    cout << "Enter the destination vertex for shortest path: ";

    cin >> endVertex;



    vector<int> shortestDistances = graph.dijkstra(startVertex);



    if (shortestDistances[endVertex] == INT_MAX) {

        cout << "There is no path from vertex " << startVertex << " to
vertex " << endVertex << "." << endl;

    } else {

        cout << "Shortest distance from vertex " << startVertex << " to
vertex " << endVertex << ": "

             << shortestDistances[endVertex] << endl;

    }



    return 0;

}
```

OUTPUT :

```
mca@mca-HP-280-G3-SFF-Business-PC:~/snehal$ g++ prac7.cpp
mca@mca-HP-280-G3-SFF-Business-PC:~/snehal$ ./a.out
Enter the number of vertices: 5
Enter the number of edges: 8
Enter the edges and weights (u v weight):
0 1 2
0 2 4
1 2 1
1 3 7
2 4 3
3 4 1
0 4 6
3 1 5
Enter the source vertex for shortest path: 0
Enter the destination vertex for shortest path: 4
Shortest distance from vertex 0 to vertex 4: 6
```