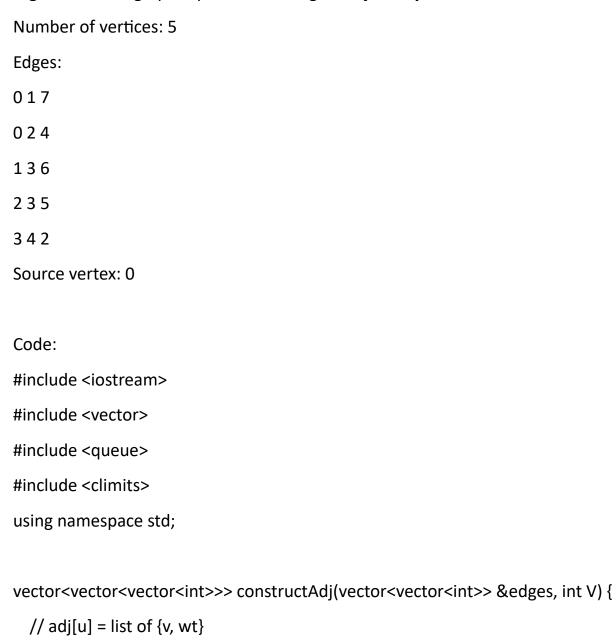
Modern Operating System and computer Network

Assignment

_							
Q		Δ	C.	н	n	n	•
u	u	C	3	LI	v		

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



```
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}); // undirected graph
  }
  return adj;
}
vector<int> dijkstra(int V, vector<vector<int>> &edges, int src) {
  vector<vector<int>>> adj = constructAdj(edges, V);
  priority_queue<vector<int>>, vector<vector<int>>> pq;
  vector<int> dist(V, INT_MAX);
  pq.push({0, src});
  dist[src] = 0;
  while (!pq.empty()) {
    int u = pq.top()[1];
    pq.pop();
```

```
for (auto x : adj[u]) {
       int v = x[0];
       int weight = x[1];
       if (dist[v] > dist[u] + weight) {
          dist[v] = dist[u] + weight;
         pq.push({dist[v], v});
       }
    }
  }
  return dist;
}
int main() {
  int V = 5;
  int src = 0;
  vector<vector<int>> edges = {
    \{0, 1, 7\}, \{0, 2, 4\},
    {1, 3, 6}, {2, 3, 5},
    {3, 4, 2}
  };
```

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

cout << "Shortest distances from source vertex " << src << ":\n";
for (int i = 0; i < V; i++)
      cout << "Vertex " << i << " : " << result[i] << endl;

return 0;
}

Output:
Shortest distances from source vertex 0:
Vertex 0 : 0
Vertex 1 : 7
Vertex 2 : 4
Vertex 3 : 9
Vertex 4 : 11</pre>
```

```
main.cpp
 1 #include <iostream>
                                                                         Shortest distances from source vertex 0:
                                                                          Vertex 0 : 0
                                                                          Vertex 2 : 4
                                                                          Vertex 3 : 9
                                                                          Vertex 4 : 11
 7 - vector<vector<int>>> constructAdj(vector<vector<int>> &edges,
       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}); // undirected graph
       return adj;
20 - vector<int> dijkstra(int V, vector<vector<int>> &edges, int src) {
       vector<vector<int>>>> adj = constructAdj(edges, V);
23
        priority_queue<vector<int>, vector<vector<int>>, greater<vector</pre>
```