

# 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 7

0 2 4

1 3 6

2 3 5

3 4 2

Source vertex: 0

Code:

```
#include <iostream>
```

```
#include <vector>
```

```
#include <queue>
```

```
#include <climits>
```

```
using namespace std;
```

```
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}); // undirected graph
}
return adj;
}

```

```

vector<int> dijkstra(int V, vector<vector<int>> &edges, int src) {
    vector<vector<vector<int>>> adj = constructAdj(edges, V);

    priority_queue<vector<int>, vector<vector<int>>, greater<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

main.cpp	Output
<pre> 1  #include &lt;iostream&gt; 2  #include &lt;vector&gt; 3  #include &lt;queue&gt; 4  #include &lt;climits&gt; 5  using namespace std; 6 7  vector&lt;vector&lt;vector&lt;int&gt;&gt;&gt; constructAdj(vector&lt;vector&lt;int&gt;&gt;&gt; &amp;edges,     int V) { 8      // adj[u] = list of {v, wt} 9      vector&lt;vector&lt;vector&lt;int&gt;&gt;&gt; adj(V); 10     for (const auto &amp;edge : edges) { 11         int u = edge[0]; 12         int v = edge[1]; 13         int wt = edge[2]; 14         adj[u].push_back({v, wt}); 15         adj[v].push_back({u, wt}); // undirected graph 16     } 17     return adj; 18 } 19 20 vector&lt;int&gt; dijkstra(int V, vector&lt;vector&lt;int&gt;&gt;&gt; &amp;edges, int src) { 21     vector&lt;vector&lt;vector&lt;int&gt;&gt;&gt; adj = constructAdj(edges, V); 22 23     priority_queue&lt;vector&lt;int&gt;, vector&lt;vector&lt;int&gt;&gt;, greater&lt;vector&lt;int&gt;&gt;&gt; pq; 24     vector&lt;int&gt; dist(V, INT_MAX); </pre>	<pre> Shortest distances from source vertex 0: Vertex 0 : 0 Vertex 1 : 7 Vertex 2 : 4 Vertex 3 : 9 Vertex 4 : 11  === Code Execution Successful === </pre>