MODERN OPERATING SYSTEMS AND COMPUTER NETWORK -ASSIGNMENT 1.1

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

Edges:

0 1 4

0 2 8

1 4 6

2 3 2

3 4 10

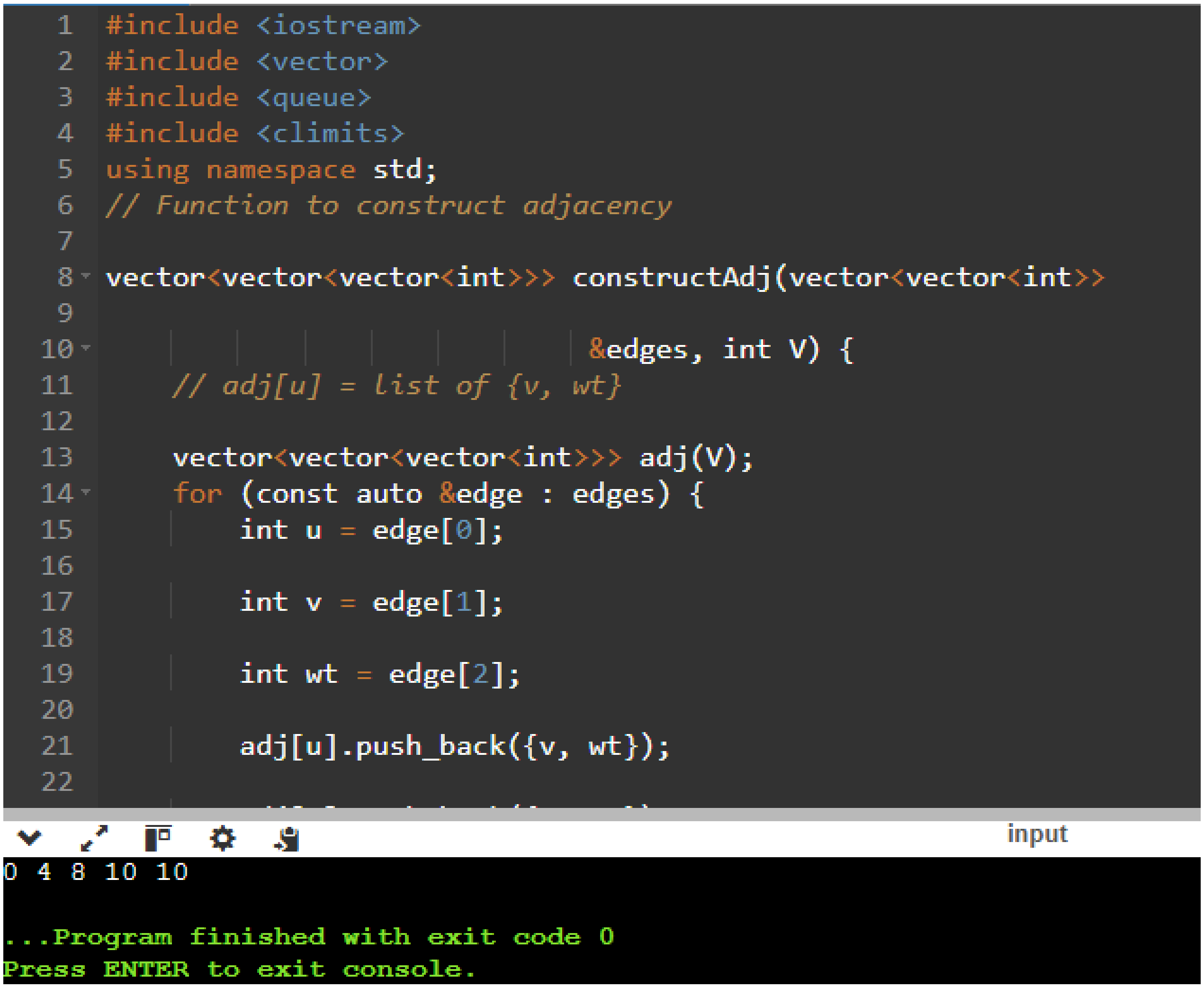
Source vertex: 0

Source Code:

```
#include &lt;iostream&gt;
#include &lt;vector&gt;
#include &lt;queue&gt;
#include &lt;climits&gt;
using namespace std;

// Function to construct adjacency
vector&lt;vector&lt;vector&lt;int&gt;&gt;&gt; constructAdj(vector&lt;vector&lt;int&gt;&gt;
                            &amp;edges, int V) {

    // adj[u] = list of {v, wt}

    vector&lt;vector&lt;vector&lt;int&gt;&gt;&gt; adj(V);

    for (const auto &amp;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});
    }

    return adj;
}

// Returns shortest distances from src to all other vertices
vector&lt;int&gt; dijkstra(int V, vector&lt;vector&lt;int&gt;&gt; &amp;edges, int src){

    // Create adjacency list
    vector&lt;vector&lt;vector&lt;int&gt;&gt;&gt; adj = constructAdj(edges, V);

    // Create a priority queue to store vertices that
    // are being preprocessed.
    priority_queue&lt;vector&lt;int&gt;, vector&lt;vector&lt;int&gt;&gt;,
                   greater&lt;vector&lt;int&gt;&gt;&gt; pq;

    // Create a vector for distances and initialize all
    // distances as infinite
    vector&lt;int&gt; dist(V, INT_MAX);

    // Insert source itself in priority queue and initialize
    // its distance as 0.
    pq.push({0, src});
    dist[src] = 0;

    // Looping till priority queue becomes empty (or all
    // distances are not finalized)
    while (!pq.empty()){

        // The first vertex in pair is the minimum distance
        // vertex, extract it from priority queue.
        int u = pq.top()[1];
        pq.pop();
        // Get all adjacent of u.
        for (auto x : adj[u]){

            // Get vertex label and weight of current
            // adjacent of u.
            int v = x[0];
            int weight = x[1];

            // If there is shorter path to v through u.
            if (dist[v] &gt; dist[u] + weight)
            {
                // Updating distance of v

                dist[v] = dist[u] + weight;
                pq.push({dist[v], v});
            }
        }
    }

    return dist;
}

// Driver program to test methods of graph class
int main(){
    int V = 5;
    int src = 0;

    // edge list format: {u, v, weight}
    vector&lt;vector&lt;int&gt;&gt; edges = {{0, 1, 4}, {0, 2, 8}, {1, 4, 6},
                                 {2, 3, 2}, {3, 4, 10}};

    vector&lt;int&gt; result = dijkstra(V, edges, src);
    // Print shortest distances in one line
    for (int dist : result)
        cout &lt;&lt; dist &lt;&lt; &quot; &quot;;


    return 0;
}
```

Output:

0 4 8 10 10



```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;
// Function to construct adjacency

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});

```

```
0 4 8 10 10

...Program finished with exit code 0
Press ENTER to exit console.
```