Write a C++ program to implement **Dijkstra's Single Source Shortest Path Algorithm** for a given weighted, undirected graph using an **adjacency matrix representation**.

1. **Problem Setup**

- We have **9 vertices** (0 to 8).

```
// A C++ program for Dijkstra's single source shortest path
// algorithm. The program is for adjacency matrix
// representation of the graph

#include <limits.h>
#include <stdio.h>

// Number of vertices in the graph
#define V 9

// A utility function to find the vertex with minimum
// distance value, from the set of vertices not yet included
// in shortest path tree
int minDistance(int dist[], bool sptSet[])
{
    // Initialize min value
```

```
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed distance
// array
void printSolution(int dist[], int n)
{
    printf("Vertex   Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("\t%d \t\t\t\t %d\n", i, dist[i]);
}

// Function that implements Dijkstra's single source
// shortest path algorithm for a graph represented using
// adjacency matrix representation
void dijkstra(int graph[V][V], int src)
{
    int dist[V]; // The output array.  dist[i] will hold the
```

```
            // shortest
// distance from src to i

bool sptSet[V]; // sptSet[i] will be true if vertex i is
                // included in shortest
// path tree or shortest distance from src to i is
// finalized

// Initialize all distances as INFINITE and stpSet[] as
// false
for (int i = 0; i < V; i++)
   dist[i] = INT_MAX, sptSet[i] = false;

// Distance of source vertex from itself is always 0
dist[src] = 0;

// Find shortest path for all vertices
for (int count = 0; count < V - 1; count++) {
   // Pick the minimum distance vertex from the set of
   // vertices not yet processed. u is always equal to
   // src in the first iteration.
   int u = minDistance(dist, sptSet);

   // Mark the picked vertex as processed
```

```
        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the
        // picked vertex.
        for (int v = 0; v < V; v++)

            // Update dist[v] only if is not in sptSet,
            // there is an edge from u to v, and total
            // weight of path from src to  v through u is
            // smaller than current value of dist[v]
            if (!sptSet[v] && graph[u][v]
                && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    // print the constructed distance array
    printSolution(dist, V);
}

// driver program to test above function
int main()
{
    /* Let us create the example graph discussed above */
```

```
    printf("KASSIM SESSAY\n")
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);

    return 0;
}
```

```
[Running] cd "/home/khaleepa/Desktop/Networking-Lab/Assiggment 2/" && g++ dijstra.c++ -o dijstra && "/home/khaleepa/Desktop/Networking-Lab/Assiggment 2/"dijstra
KASSIM SESSAY
Vertex   Distance from Source
    0              0
    1              4
    2              12
    3              19
    4              21
    5              11
    6              9
    7              8
    8              14

[Done] exited with code=0 in 0.291 seconds
```