Source Code

```
// Dijkstra's Shortest Path C++ code
// Using adjacency matrix for graph representation

#include <limits.h>
#include <stdio.h>
#include<iostream>
using namespace std;

// Globally declare the Number of vertices for the graph
#define V 9

// Find the vertex with minimum
// distance value, from the set of vertices not yet included
// in shortest path route
int minDist(int dist[], bool visitedArray[])
{
    // Initialize min value
    int min = INT_MAX, min_index; //INT_MAX is a C++ constant with max integer
value

    for (int i = 0; i < V; i++)
    {
         if (visitedArray[i] == false && dist[i] <= min)
         {
                 min = dist[i], min_index = i;
         }
    }

    return min_index;
}

// A utility function to print the constructed distance
// array
void printSolution(int dist[], int n)
{
    cout<<"Vertex  \t\t\t 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 visitedArray[V]; // visitedArray[i] will be true if vertex i is visited

    // Initialize all distances as INFINITE and visitedArray[] as false
```

```
    for (int i = 0; i < V; i++)
    {
         dist[i] = INT_MAX,
         visitedArray[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 = minDist(dist, visitedArray);

         // Mark the visited vertex as true
         visitedArray[u] = true;

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

                 // Update dist[v] only if is not in visitedArray,
                 // 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 (!visitedArray[v] && graph[u][v] && dist[u] != INT_MAX &&
dist[u] + graph[u][v] < dist[v])
                 {
                         dist[v] = dist[u] + graph[u][v];
                 }
    }

    // print the shorted constructed path
    printSolution(dist, V);
}

// driver program to test above function
int main()
{
    //int V;

    /* Adjacency Matrix for the graph for which Shortest path is to be
calculated*/
    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;
}

Output -
Vertex                          Distance from Source
        0                               0
        1                               4
        2                               12
        3                               19
        4                               21
        5                               11
        6                               9
        7                               8
        8                               14
```