```
// Example code to demonstrate Dijkstra's algorithm for shortest path in a weighted graph

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

#define MAX_VERTICES 100

// Structure for an adjacency list node
typedef struct AdjListNode {
    int dest;
    int weight;
    struct AdjListNode* next;
} AdjListNode;

// Structure for an adjacency list
typedef struct AdjList {
    AdjListNode* head;
} AdjList;

// Structure for a graph
typedef struct Graph {
    int V; // Number of vertices
    AdjList* array;
} Graph;

// Function to create a new adjacency list node
AdjListNode* newAdjListNode(int dest, int weight) {
    AdjListNode* newNode = (AdjListNode*) malloc(sizeof(AdjListNode));
    newNode->dest = dest;
    newNode->weight = weight;
    newNode->next = NULL;
    return newNode;
}

// Function to create a graph with V vertices
Graph* createGraph(int V) {
    Graph* graph = (Graph*) malloc(sizeof(Graph));
    graph->V = V;
    graph->array = (AdjList*) malloc(V * sizeof(AdjList));
    for (int i = 0; i < V; ++i)
        graph->array[i].head = NULL;
    return graph;
}

// Function to add an edge to an undirected graph
void addEdge(Graph* graph, int src, int dest, int weight) {
    AdjListNode* newNode = newAdjListNode(dest, weight);
```

```
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;
    newNode = newAdjListNode(src, weight);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}

// Function to find the vertex with the minimum distance value,
// from the set of vertices not yet included in the shortest path tree
int minDistance(int dist[], bool sptSet[], int V) {
    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;
}

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

// Function that implements Dijkstra's single source shortest path algorithm
// for a graph represented using adjacency list representation
void dijkstra(Graph* graph, int src) {
    int V = graph->V;
    int dist[V];
    bool sptSet[V];

    for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet, V);
        sptSet[u] = true;

        AdjListNode* node = graph->array[u].head;
        while (node != NULL) {
            int v = node->dest;
            if (!sptSet[v] && dist[u] != INT_MAX && dist[u] + node->weight < dist[v])
                dist[v] = dist[u] + node->weight;
            node = node->next;
        }
    }
```

```
    printSolution(dist, V);
}

int main() {
    int V = 9;
    Graph* graph = createGraph(V);
    addEdge(graph, 0, 1, 4);
    addEdge(graph, 0, 7, 8);
    addEdge(graph, 1, 2, 8);
    addEdge(graph, 1, 7, 11);
    addEdge(graph, 2, 3, 7);
    addEdge(graph, 2, 8, 2);
    addEdge(graph, 2, 5, 4);
    addEdge(graph, 3, 4, 9);
    addEdge(graph, 3, 5, 14);
    addEdge(graph, 4, 5, 10);
    addEdge(graph, 5, 6, 2);
    addEdge(graph, 6, 7, 1);
    addEdge(graph, 6, 8, 6);
    addEdge(graph, 7, 8, 7);

    dijkstra(graph, 0);

    return 0;
}
```