```c
/** ---------------------------------------------------------------

Dijkstra's algorithm is an algorithm for finding the shortest paths between
nodes in a graph, which may represent, for example, road networks. It was
conceived by computer scientist Edsger W. Dijkstra in 1956 and published three
years later.

The Bellman–Ford algorithm is an algorithm that computes shortest paths from a
single source vertex to all of the other vertices in a weighted digraph.



Algotithm
function Dijkstra(Graph, source):
    create vertex set Q

    for each vertex v in Graph:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q

    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with min dist[u]
        remove u from Q

        for each neighbor v of u:
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]



Dijkstra's algorithm is a greedy algorithm that solves the shortest path
problem for a directed graph G.
Dijkstra's algorithm solves the single-source shortest-path problem when all
edges have non-negative weights.

------------------------------------------------------------- **/

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

#define MX 10
#define NF 786


void showMatrix(int graph[][MX], int vertices, const char *text){
    int i, j;

    printf("\n\t%s is....\n\n", text);
    printf("\t\t   u|v |");
    for(i=0; i < vertices; i++)
        printf("%4d ", i);
    printf("\n");

    printf("\t\t--------");
    for(i=0; i < vertices; i++)
        printf("-----");
    printf("\n");

    for(i=0; i < vertices; i++){
        printf("\t\t%5d  |", i);
        for(j = 0; j < vertices; j++)
            printf("%4d ", graph[i][j]);
        printf("\n");
    }
    printf("\n");

}

void initMatrix(int graph[][MX]){
    int i, j, weight;
    for(i=0; i < MX; i++)
        for(j = 0; j < MX; j++)
            graph[i][j] = NF;
}


int createGraph(int graph[][MX]){
    int i, j, vCnt=0, weight;
    int u, v, vertices, type;

    printf("\n\tGraph Creation [Undirected/Directed]...\n");

    printf("\t\tType of Graph [0: UnDirected] := ");
    scanf("%d", &type);

    if(type != 0)
        type = 1;

    do{
        printf("\t\tHow Many Vertices [upto %d vertices]?? ", MX);
        scanf("%d", &vertices);
    }while(vertices < 1 || vertices > MX);

    printf("\n");

    printf("\n\tVertices starts at 0 and terminates at %d\n", vertices-1);
    printf("\t\tVertex ID of -1 terminates Input\n");
    printf("\n\tEnter Existing Edges in the Graph\n\n");
    printf("\t\t------------------------------------------------------\n");
    printf("\t\tEdge#     'u'      'v'     Cost    Remark\n");
    printf("\t\t------------------------------------------------------\n");


    do{
        do{
            printf("\t\t  %2d       ", vCnt+1);
            scanf("%d%d%d", &u, &v, &weight );

            if(u == v && weight < 0)
```

```
                printf("\t\t\t\t\t\t\t\t\t\t\tNegative Cycle
\n");
        }while(u == v && (u != -1 && v != -1) && weight < 0);

        printf("  \t   ");

        if((u != -1 || v != -1) && u < vertices && v < vertices){
            if(graph[u][v] == NF){
                if(type)
                    graph[u][v] = weight;
                else
                    graph[u][v] = graph[v][u] = weight;

                printf("\t\t\t\t\t\t\t\t\t\tEdge Taken\n");

            } else
                printf("\t\t\t\t\t\t\t\t\t\tEdge Exists\n");

        }else
            printf("\t\t\t\t\t\t\t\t\t\tInvalid Edge\n");

        vCnt++;

    }while(u != -1 || v != -1);


    for(i = 0; i < MX; i++)
        if(graph[i][i] == NF)
            graph[i][i] = 0;

    return vertices;
}


int minDistance(int dist[], int visited[], int vertices){
    int min = NF, minPos, i;
    for(i = 0; i < vertices; i++)
        if(visited[i] == 0 && dist[i] <= min){
            min = dist[i];
            minPos = i;
        }

    return minPos;
}


void printPath(int prev[], int vt){

    if(prev[vt] == -1)
        return;
    printPath(prev, prev[vt]);
    printf("->%2d", vt);
}


int printDijkstra(int dist[], int vertices, int prev[], int source)
{
    int src = source, i;
    printf("\t\tPath: [u]->[v]  Distance  Shortest Path\n");
    printf("\t\t      --------  --------  -------------\n");

    for(i = 0; i < vertices; i++){
        if(i == src)
            continue;
        printf("\t\t      [%d]->[%d]    %4d    %2d", src, i, dist[i], src);
        printPath(prev, i);
        printf("\n");
    }
    printf("\n");
}


void printLine(int vertices){
    int i;
    printf("\t\t------------------");
    for(i = 0; i < vertices; i++)
        printf("-------");
    printf("\n");
}

void printTitle(int vertices){
    int i;
    printf("\t\t Data Structure   |");
    for(i = 0; i < vertices; i++)
        printf("  [%d] |", i);
    printf("\n");
}

void printArray(int arr[], int vertices, const char *text){
    int i;
    printf("\t\t  %14s  |", text);
    for(i = 0; i < vertices; i++)
        printf(" %3d  |", arr[i]);
    printf("\n");
}


void printDijkstraState(int dist[], int visited[], int prev[], int vertices){

    printf("\n");
    printLine(vertices);
        printTitle(vertices);
    printLine(vertices);
        printArray(dist, vertices, "  DIST[]   ");
    printLine(vertices);
        printArray(prev, vertices, "  PREV[]   ");
    printLine(vertices);
        printArray(visited, vertices, "  VISITED[]");
    printLine(vertices);
    printf("\n");
}


void spDijkstra(int graph[][MX], int src, int vertices){

    int dist[MX], visited[MX], prev[MX], i, u, v;
```

```c
    for(i = 0; i < vertices; i++){
        prev[i] = -1;
        dist[i] = NF;
        visited[i] = 0;
    }

    dist[src] = 0;

    for(i = 0; i < vertices-1; i++){
        u = minDistance(dist, visited, vertices);
        visited[u] = 1;

        for(v = 0; v < vertices; v++){
            if(!visited[v] && graph[u][v] && dist[u] + graph[u][v] < dist[v]){
                prev[v] = u;
                dist[v] = dist[u] + graph[u][v];
            }
        }

        printDijkstraState(dist, visited, prev, vertices);

        printf("\n\t\tPress Any Key to Proceed ...");
        getc(stdin);

    }

   printDijkstra(dist, vertices, prev, src);
}



/** ----------------------------------------------------------------- **/

int main(){

    int graph[MX][MX], vertices, source, kontinu;
    initMatrix(graph);
    vertices = createGraph(graph);

    printf("\nGraph with %2d vertices ...\n", vertices);
    showMatrix(graph,  vertices, "Adjacency Matrix");
    do{
        do{
            printf("\n\tEnter Source Vertex [0 thru %d]: ", vertices-1);
            scanf("%d", &source);

            spDijkstra(graph, source, vertices);

        }while(source < 0 || source >= vertices);

        printf("\n\tDijkstra's Algorithm for Different Source?? [0 to Stop] :
");
        scanf("%d", &kontinu);
    }while(kontinu);

    return 0;
}
```

```
    /** ----------------- E X E C U T I O N   T R A I L ----------------


        Graph Creation [Undirected/Directed]...
            Type of Graph [0: UnDirected] := 0
            How Many Vertices [upto 10 vertices]?? 4


        Vertices starts at 0 and terminates at 3
            Vertex ID of -1 terminates Input

        Enter Existing Edges in the Graph

            ------------------------------------------------------
            Edge#     'u'      'v'     Cost    Remark
            ------------------------------------------------------
               1       0        1       4
                                                Edge Taken
               2       0        2      -2
                                                Edge Taken
               3       3        2       1
                                                Edge Taken
               4       3        1      -1
                                                Edge Taken
               5       2        1       3
                                                Edge Taken
               6      -1       -1       0
                                                Invalid Edge

    Graph with  4 vertices ...

        Adjacency Matrix is....
               u|v |    0    1    2    3
            ---------------------------
                0  |    0    4   -2  786
                1  |    4    0    3   -1
                2  |   -2    3    0    1
                3  |  786   -1    1    0


        Enter Source Vertex [0 thru 3]: 0
            Path: [u]->[v]  Distance  Shortest Path
                  --------  --------  -------------
                  [0]->[1]      -2     0-> 2-> 3-> 1
                  [0]->[2]      -2     0-> 2
                  [0]->[3]      -1     0-> 2-> 3


        Dijkstra's Algorithm for Different Source?? [0 to Stop] : 1

        Enter Source Vertex [0 thru 3]: 3
            Path: [u]->[v]  Distance  Shortest Path
                  --------  --------  -------------
                  [3]->[0]      -1     3-> 2-> 0
                  [3]->[1]      -1     3-> 1
                  [3]->[2]       1     3-> 2
```

```
   Dijkstra's Algorithm for Different Source?? [0 to Stop] : 1

   Enter Source Vertex [0 thru 3]: 1
       Path: [u]->[v]  Distance  Shortest Path
             --------  --------  -------------
             [1]->[0]      -2     1-> 3-> 2-> 0
             [1]->[2]       0     1-> 3-> 2
             [1]->[3]      -1     1-> 3

   Dijkstra's Algorithm for Different Source?? [0 to Stop] : 2

   Enter Source Vertex [0 thru 3]: 2
       Path: [u]->[v]  Distance  Shortest Path
             --------  --------  -------------
             [2]->[0]      -2     2-> 0
             [2]->[1]       0     2-> 3-> 1
             [2]->[3]       1     2-> 3

   Dijkstra's Algorithm for Different Source?? [0 to Stop] : 0


----------------   E X E C U T I O N   T R A I L  ---------------- **/
```