ALGORITHM:

```
function Dijkstra(Graph, source):
      for each vertex v in Graph:                                  //
Initializations
      dist[v]  := infinity ;                                  //
Unknown distance function from
                                                              //
source to v
      previous[v]  := undefined ;                             //
Previous node in optimal path
      end for                                                      //
from source

      dist[source]  := 0 ;                                        //
Distance from source to source
      Q := the set of all nodes in Graph ;                        //
All nodes in the graph are
                                                              //
unoptimized - thus are in Q
      while Q is not empty:                                       //
The main loop
      u := vertex in Q with smallest distance in dist[] ;     //
Source node in first case
      remove u from Q ;
      if dist[u] = infinity:
         break ;                                              // all
remaining vertices are
      end if                                                  //
inaccessible from source

      for each neighbor v of u:                               //
where v has not yet been
                                                              //
removed from Q.
          alt := dist[u] + dist_between(u, v) ;
          if alt < dist[v]:                                   //
Relax (u,v,a)
              dist[v]  := alt ;
              previous[v]  := u ;
              decrease-key v in Q;                            //
Reorder v in the Queue
          end if
      end for
      end while
      return dist;
end function
```

CODE:

```
/
*****************************************************************************
************
*File       : Dijkstra.cpp
```

```
*Description    : Program to find shortest paths to other vertices
              using Dijkstra's algorithm.
*Author     : Prabodh C P
*Compiler   : gcc compiler 4.6.3, Ubuntu 12.04
*Date       : Friday 22 November 2013

*************************************************************************
************/


 #include<ostream>
 #include<cstdio>
using namespace std;

const int MAXNODES = 10,INF = 9999;

void fnDijkstra(int [][MAXNODES], int [], int [], int[], int, int,
int);

/
*************************************************************************
**********
*Function   : main
*Input parameters: no parameters
*RETURNS    :   0 on success
*************************************************************************
**********/

int main(void)
{
    int n,cost[MAXNODES]
[MAXNODES],dist[MAXNODES],visited[MAXNODES],path[MAXNODES],i,j,sourc
e,dest;

    cout << "\nEnter the number of nodes\n";
    cin >> n;
    cout << "Enter the Cost Matrix\n" << endl;
    for (i=0;i<n;i++)
    for (j=0;j<n;j++)
        cin >> cost[i][j];


    for (source = 0; source < n; source++)
    {
    getchar();
    cout << "\n//For Source Vertex : " << source  << " shortest path
to other vertices//"<< endl;
    for (dest=0; dest < n; dest++)
    {
        fnDijkstra(cost,dist,path,visited,source,dest,n);


        if (dist[dest] == INF)
            cout << dest << " not reachable" << endl;
```

```
        else
        {
            cout << endl;
            i = dest;
        do
            {
                cout << i << "<--";
                i = path[i];
            }while (i!= source);
            cout << i << " = " << dist[dest] << endl;
        }
    }
    cout << "Press Enter to continue...";
    }

    return 0;
}



/
**********************************************************************
**********
*Function   : fnDijkstra
*Description    : Function to find shortest paths to other vertices
              using Dijkstra's algorithm.
*Input parameters:
*   int c[][] - cost adjacency matrix of the graph
*   int d[] - distance vector
*   int p[] - path vector
*   int s[] - vector to store visited information
*   int so  - source vertex
*   int de  - destination vertex
*   int n   - no of vertices in the graph
*RETURNS    : no value
**********************************************************************
**********/

void fnDijkstra(int c[MAXNODES][MAXNODES], int d[MAXNODES], int
p[MAXNODES], int s[MAXNODES], int so, int de, int n)
{
    int i,j,a,b,min;

    for (i=0;i<n;i++)
    {
    s[i] = 0;
    d[i] = c[so][i];
    p[i] = so;
    }

    s[so] = 1;

    for (i=1;i<n;i++)
    {
```

```
    min = INF;
    a = -1;
    for (j=0;j<n;j++)
    {
        if (s[j] == 0)
        {
            if (d[j] < min)
            {
                min = d[j];
                a = j;
            }
        }
    }

    if (a == -1) return;

    s[a] = 1;

    if (a == de) return;

    for (b=0;b<n;b++)
    {
        if (s[b] == 0)
        {
            if (d[a] + c[a][b] <d[b])
            {
                d[b] = d[a] + c[a][b];
                p[b] = a;
            }
        }
    }
    }
}
```

Output

Enter the number of nodes 5 Enter the Cost Matrix

0 3 9999 7 9999 3 0 4 2 9999 9999 4 0 5 6 7 2 5 0 4 9999 9999 6 4 0

//For Source Vertex : 0 shortest path to other vertices//

0<--0 = 0

1<--0 = 3

2<--1<--0 = 7

3<--1<--0 = 5

4<--3<--1<--0 = 9 Press Enter to continue...

//For Source Vertex : 1 shortest path to other vertices//

0<--1 = 3

1<--1 = 0

2<--1 = 4

3<--1 = 2

4<--3<--1 = 6 Press Enter to continue...

//For Source Vertex : 2 shortest path to other vertices//

0<--1<--2 = 7

1<--2 = 4

2<--2 = 0

3<--2 = 5

4<--2 = 6 Press Enter to continue...

//For Source Vertex : 3 shortest path to other vertices//

0<--1<--3 = 5

1<--3 = 2

2<--3 = 5

3<--3 = 0

4<--3 = 4 Press Enter to continue...

//For Source Vertex : 4 shortest path to other vertices//

0<--1<--3<--4 = 9

1<--3<--4 = 6

2<--4 = 6

3<--4 = 4

4<--4 = 0 Press Enter to continue...