ASSIGNMENT NO:7 **DATE:29/03/2016**

PROGRAM TITLE:Find the Minimum Path of all vertices from a given source in a Simple Graph using Dijkstra's Algorithm.

PROGRAM ALGORITHM:

```
Algo_Dijkstra's(G,S)
{               //G is the Graph and S is the source vertex.
     Create a vertex set Q           //Initially it is empty.
     for each vertex v in G
     {          //Initialization.
          dist[v]=INFINITY      //Unknown distance from source
          prev[v]=UNDEFINED     //Previous Node is shortest path from source.
          add v to Q            //All Nodes are in Q.
     }
     dist[S]=0
     remove S from Q
     while (Q is not empty)
     {
          u=vertex in Q with minimum dist[u]
          remove u from Q
          for each neighbour v of u
          {                     //Update all distances via u to all
               dist1=dist[u]+length(u,v)  //unexplored vertices.
               if dist1<dist[v]
               {
                    dist[v]=dist1
                    prev[v]=u
               }
          }
     }
     return (dist[],prev[])
}
```

PROGRAM CODE:

```
//C Program to find distance between two vertices using Dijkstra's
#include <stdio.h>
#include <stdlib.h>
#define datatype int
void create_graph(datatype **graph,int n)
{
     int i,j,x;
     datatype w;
     for(i=0;i<n;i++)
     {
          for(j=0;j<n;j++)
          {
               graph[i][j]=9999;
          }
     }
     printf("\tIdentify the adjoining vertices:");
     for(i=0;i<n;i++)
     {
          printf("\n\tEnter the adjoining vertices of %d.Enter -99 to
stop::",i);
          for(j=0;j<n;j++)
          {
               scanf("%d",&x);
               if(x==-99)
```

```
                        break;
                else
                {
                        if(graph[x][i]==9999)
                        {
                                printf("\tEnter corresponding weight::");
                                scanf("%d",&w);
                                graph[i][x]=w;
                        }
                        else
                                graph[i][x]=graph[x][i];
                        printf("\tEnter next adjoining vertex::");
                }
        }
    }
}
void print_graph(datatype **graph,int n)
{
    int i,j;
    printf("\n\tThe Adjacency Matrix of the Graph is::");
    for(i=-1;i<n;i++)
    {
        printf("\n");
        for(j=-1;j<n;j++)
        {
            if(i==-1)
                printf("%d",j);
            else if(j==-1)
                printf("%d",i);
            else
                printf("%d",graph[i][j]);
            printf("\t");
        }
    }
    printf("\n");
}
void print_path(int *prev,int n,int dest)
{
    int u=dest,i=n-1;
    int *s = (int*)malloc(n*sizeof(int));
    while(prev[u]!=-1)
    {
        s[i--]=u;
        u=prev[u];
    }
    s[i]=u;
    for(;i<n;i++)
    {
        printf("%d-",s[i]);
    }
}
void dijkstra(datatype **graph,int n)
{
    int i,s,q=n,min,dest,u;
    datatype dist1;
    char ch='y';
    int *mark = (int*)malloc(n*sizeof(int));
```

```
datatype *dist = (datatype*)malloc(n*sizeof(datatype));
int *prev = (int*)malloc(n*sizeof(int));
for(i=0;i<n;i++)
{
      mark[i]=1;
      dist[i]=9999;
      prev[i]=-1;
}
printf("\n\tEnter the source vertex::");
scanf("%d",&s);
dist[s]=0;
while(q!=0)
{
      for(i=0,min=9999;i<n;i++)
      {
            if((mark[i]==1)&&(dist[i]<min))
            {
                  min=dist[i];
                  u=i;
            }
      }
      mark[u]=0;
      q--;
      for(i=0;i<n;i++)
      {
            if((graph[u][i]!=9999)&&(mark[i]==1))
            {
                  dist1=dist[u]+graph[u][i];
                  if(dist1<dist[i])
                  {
                        dist[i]=dist1;
                        prev[i]=u;
                  }
            }
      }
}
printf("\n\tThe distances from %d are:",s);
for(i=0;i<n;i++)
      printf("\n\tto %d = %d",i,dist[i]);
do
{
      printf("\n\tDo you want to display a path(y/n)::");
      getchar();
      scanf("%c",&ch);
      if(ch=='n'||ch=='N')
            break;
      printf("\tEnter destination::");
      scanf("%d",&dest);
      if(dist[dest]!=9999)
      {
            printf("\tThe path from %d to %d is::",s,dest);
            print_path(prev,n,dest);
      }
      else
            printf("\n\tNo Path.");
}
while(ch=='y'||ch=='Y');
```

```
}
int main()
{
      int n,i;
      printf("\n\tEnter the number of vertices::");
      scanf("%d",&n);
      datatype **graph= (datatype**)malloc(n*sizeof(datatype*));
      for(i=0;i<n;i++)
      {
            graph[i]=(datatype*)malloc(n*sizeof(datatype));
      }
      create_graph(graph,n);
      print_graph(graph,n);
      dijkstra(graph,n);
      return 0;
}
```

OUTPUT:

```
      Enter the number of vertices::6
      Identify the adjoining vertices:
      Enter the adjoining vertices of 0.Enter -99 to stop::1
      Enter corresponding weight::7
      Enter next adjoining vertex::2
      Enter corresponding weight::9
      Enter next adjoining vertex::5
      Enter corresponding weight::14
      Enter next adjoining vertex::-99

      Enter the adjoining vertices of 1.Enter -99 to stop::0
      Enter next adjoining vertex::2
      Enter corresponding weight::10
      Enter next adjoining vertex::3
      Enter corresponding weight::15
      Enter next adjoining vertex::-99

      Enter the adjoining vertices of 2.Enter -99 to stop::0
      Enter next adjoining vertex::1
      Enter next adjoining vertex::5
      Enter corresponding weight::10
      Enter next adjoining vertex::3
      Enter corresponding weight::11
      Enter next adjoining vertex::-99

      Enter the adjoining vertices of 3.Enter -99 to stop::2
      Enter next adjoining vertex::1
      Enter next adjoining vertex::4
      Enter corresponding weight::6
      Enter next adjoining vertex::-99

      Enter the adjoining vertices of 4.Enter -99 to stop::3
      Enter next adjoining vertex::5
      Enter corresponding weight::9
      Enter next adjoining vertex::-99

      Enter the adjoining vertices of 5.Enter -99 to stop::0
      Enter next adjoining vertex::2
```

```
      Enter next adjoining vertex::4
      Enter next adjoining vertex::-99

      The Adjacency Matrix of the Graph is::
-1    0     1     2     3     4     5
0     9999  7     9     9999  9999  14
1     7     9999  10    15    9999  9999
2     9     10    9999  11    9999  10
3     9999  15    11    9999  6     9999
4     9999  9999  9999  6     9999  9
5     14    9999  10    9999  9     9999

      Enter the source vertex::0
      The distances from 0 are:
      to 0 = 0
      to 1 = 7
      to 2 = 9
      to 3 = 20
      to 4 = 23
      to 5 = 14
      Do you want to display a path(y/n)::y
      Enter destination::4
      The path from 0 to 4 is::0-5-4-
      Do you want to display a path(y/n)::y
      Enter destination::5
      The path from 0 to 5 is::0-5-
      Do you want to display a path(y/n)::y
      Enter destination::3
      The path from 0 to 3 is::0-2-3-
      Do you want to display a path(y/n)::y
      Enter destination::2
      The path from 0 to 2 is::0-2-
      Do you want to display a path(y/n)::y
      Enter destination::1
      The path from 0 to 1 is::0-1-
      Do you want to display a path(y/n)::y
      Enter destination::0
      The path from 0 to 0 is::0-
      Do you want to display a path(y/n)::n
```

DISCUSSION:

1. We use Adjacency Matrix for storing the graph.
2. Complexity is $O(|V|^2)$
3. The default weight is set arbitrarily high so that it is not considered in the computations.
4. The program works for both directed and undirected graph.