7. From a given vertex in a weighted connected graph, find shortest paths to other vertices using **Dijkstra's algorithm**. Write the program in Java.

```
import java.util.*;
public class DijkstraDemo
{
 static int dist [] = new int[20];       /* Array to store Min Distance */
 static int visited [] = new int[20];  /* Array to keep track of visited nodes */
 static int path[] = new int[20];      /* Array to keep track of shortest path from source node
                                                      to all other nodes */
 static int source;
 public static void main(String [] args)
 {
    int u,v,n;
    int w[][] = new int[20][20]; /* 2D-array to read the values of weighted graph */
    int i,j;

    Scanner in = new Scanner(System.in);

    System.out.println("Enter the no of nodes");
    n = in.nextInt();

    System.out.println("Enter the weight matrix");
    for(i=1;i<=n;i++)
    {
     for(j=1;j<=n;j++)
     {
      w[i][j] = in.nextInt();
     }
    }

    System.out.println("Enter the source vertex");
    source = in.nextInt();
    /* Initially set all the nodes as unvisited */
    for(i=1;i<=n;i++)
    {
      visited[i]=0;
    }
    /* Set the source node as visited */
     visited[source] = 1;

    for(i=1;i<=n;i++)
    {
      dist[i] = w[source][i];
    }
    /* Initially set the shortest path from source node to all other nodes is through source */
    for(i=1;i<=n;i++)
```

```
0  3 999 7 999
3 0 4 2 999
999 4 0 6 6
7 2 6 0 4
999 999 6 4 0
```

```
    {
      path[i] = source;
    }
    /* Set the shortest path to the source node as -1 */
    path[source] = -1;

   for(i=1;i<n;i++)
   {
     u = minDistance(n);                  u=2 u=4  visited[2]=1,visited[1]=1
     visited[u] = 1;                               visited= 1 1 0 0 0 visited=1 1 0 1 0
     v = 1;                                                 w= 0 3 999 7 999   dist = 0 3 999 7 999
                                                                3 0 4 2 999    dist= 0 3 7 5 999
     while(v<=n)                                                               dist= 0 3 7 5 9
     {                                                          7 2 6 0 4
       if((dist[u] + w[u][v]<dist[v]) && visited[v]==0)                   path[3]=2
       {                                                                  path[5]=4
         dist[v] = dist[u] + w[u][v];                                    path =-1 1 1 1 1
         path[v] = u;                                                           -1 1 2 2 1
       }                                                                         -1 1 2 2 4
       v++;
     } end of while loop
   }  end of for loop
   printShortest(n);
}
/* Method to find next closest node to the souce node */
public static int minDistance(int n)
{                                                               dist= 0 3 999 7 999
    int i,minDist,minIndex=0;                                          0 3 7 5 999
    minDist = 999;

    for(i=1;i<=n;i++)                                             visited=1 1 0 0 0
    {
     if(dist[i] < minDist && visited[i] == 0)
     {
        minDist = dist[i];                  mindist=3           mindist=7  mindist=5
        minIndex = i;                       minIndex=2          minIndex=3 minindex=4
     }
    }
    return minIndex;
}
/* Method to print shortest distance from source node to all other nodes*/
public static void  printShortest(int n)
{
  int i;

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

```
    {
      if(i!=source)
      {
       System.out.println("The shortest distance from node " + source + " to " + i +"=" + dist[i]);
       System.out.print("The shortest path from node " + source+ " to " + i +" is: ");
       System.out.print(source);
       printPath(path,i);
       System.out.println();                                  -1 1 2 2 4
      }
    }
  }
 /*Method to print shortest path */
 public static void printPath(int path[],int j)  printpath(path,2)        printpath(path,1)
 {
   if(path[j]== -1)
    return ;

   printPath(path,path[j]);   printpath(path,1)
   System.out.print("--->"+ j +" ");
 }
}
```