8. Write a program to find the shortest path between vertices using bellman-ford algorithm

Distance Vector Algorithm is a decentralized routing algorithm that requires that each router simply inform its neighbours of its routing table. For each network path, the receiving routers pick the neighbour advertising the lowest cost, then add this entry into its routing table for re-advertisement. To find the shortest path, Distance Vector Algorithm is based on one of two basic algorithms: Bellman-Ford and Dijkstra algorithms. Routers that use this algorithm have to maintain the distance tables (which is a one- dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always up date by exchanging information with the neighbouring nodes. The number of data in the table equals to that of all nodes in networks (excluded itself). The columns of table represent the directly attached neighbours whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc. 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. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman–Ford algorithm can detect negative cycles and report their existence.

Source Code:

```
import java.util.Scanner;
public class BellmanFord
{
	private int distances[];
	private int nv;
	public static int MAX_VALUE = 999;
	public BellmanFord(int nv)
	{
		this.nv= nv;
		distances = new int[nv+1];
	}
	public void BellmanFordEvaluation(int source,int adm[][])
	{
		for(int node=1;node<=nv;node++)
		{
			distances[node]=MAX_VALUE;
		}
		distances[source]=0;
		for(int node=1;node<=nv-1;node++)
		{
			for(int sn=1;sn<=nv;sn++)
			{
```

```
                    for(int dn=1;dn<=nv;    dn++)
                    {
                        if(adm[sn][dn]!=MAX_VALUE)
                        {
                            if(distances[dn]>distances[sn] + adm[sn][dn])
                                distances[dn]=distances[sn]+adm[sn][dn];
                        }
                    }
            }
        }
        for (int sn=1;sn<=nv;sn++)
        {
            for(int dn=1;dn<=nv; dn++)
            {
                if(adm[sn][dn]!=MAX_VALUE)
                {
                    if(distances[dn]>distances[sn] + adm[sn][dn])
                        System.out.println("The graph contains negative edge
                                                cycle");
                }
            }
        }
        for(int vertex=1;vertex<=nv;vertex++)
        {
            System.out.println("Distance of source "+ source + " to " + vertex
                                + " is " + distances[vertex]);
        }
}

public static void main(String args[])
{
        int nv, source;
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of vertices");
        nv=scanner.nextInt();
        int adm[][]= new int[nv+1][nv+1];
        System.out.println("Enter the adjacency matrix");
        for(int sn=1;sn<=nv;sn++)
        {
            for(int dn=1;dn<=nv; dn++)
            {
                adm[sn][dn]=scanner.nextInt();
                if(sn==dn)
                {
                    adm[sn][dn]=0;
                    continue;
                }
                if(adm[sn][dn]==0)
                {
                    adm[sn][dn]=MAX_VALUE;
                }
            }
```

```
            }
            System.out.println("Enter the source vertex");
            source=scanner.nextInt();
            BellmanFord bellmanford = new BellmanFord(nv);
            bellmanford.BellmanFordEvaluation(source,adm);
        }
}
```

OUTPUT

Output 1

```
cs@cs-desktop:~$ java BellmanFord
Enter the number of vertices
4
Enter the adjacency matrix
0 3 2 0
0 0 -2 0
0 0 0 -1
0 3 0 0
Enter the source vertex
1
Distance of source 1 to  1 is 0
Distance of source 1 to  2 is 3
Distance of source 1 to  3 is 1
Distance of source 1 to  4 is 0
```

Output 2

```
cs@cs-desktop:~$ java BellmanFord
Enter the number of vertices
4
Enter the adjacency matrix
0 1 0 3
0 0 -3 0
0 0 0 -1
0 2 0 0
Enter the source vertex
1
The graph contains negative edge   cycle
Distance of source 1 to  1 is 0
Distance of source 1 to  2 is -5
Distance of source 1 to  3 is -6
Distance of source 1 to  4 is -7
```