DVR:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class DVR {
 static int graph[][];
 static int via[][];
 static int rt[][];
 static int v;
 static int e;
 public static void main(String args[]) throws IOException {
 BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
 System.out.println("Please enter the number of Vertices: ");
 v = Integer.parseInt(br.readLine());
 System.out.println("Please enter the number of Edges: ");
 e = Integer.parseInt(br.readLine());
 graph = new int[v][v];
 via = new int[v][v];
 rt = new int[v][v];
 for (int i = 0; i < v; i++)
 for (int j = 0; j < v; j++) {
 if (i == j)
 graph[i][j] = 0;
 else
 graph[i][j] = 9999;
 }
 for (int i = 0; i < e; i++) {
 System.out.println("Please enter data for Edge " + (i + 1) +
":");
 System.out.print("Source: ");
 int s = Integer.parseInt(br.readLine());
 s--;
 System.out.print("Destination: ");
 int d = Integer.parseInt(br.readLine());
 d--;
 System.out.print("Cost: ");
 int c = Integer.parseInt(br.readLine());
 graph[s][d] = c;
 graph[d][s] = c;
```

```
 }
 dvr_calc_disp("The initial Routing Tables are: ");
 System.out.print("Please enter the Source Node for the edge whose
cost has changed: ");
 int s = Integer.parseInt(br.readLine());
 s--;
 System.out.print("Please enter the Destination Node for the edge
whose cost has changed: ");
 int d = Integer.parseInt(br.readLine());
 d--;
 System.out.print("Please enter the new cost: ");
 int c = Integer.parseInt(br.readLine());
 graph[s][d] = c;
 graph[d][s] = c;
 dvr_calc_disp("The new Routing Tables are: ");
 }
 static void dvr_calc_disp(String message) {
 System.out.println();
 init_tables();
 update_tables();
 System.out.println(message);
 print_tables();
 System.out.println();
 }
 static void update_table(int source) {
 for (int i = 0; i < v; i++) {
 if (graph[source][i] != 9999) {
 int dist = graph[source][i];
 for (int j = 0; j < v; j++) {
 int inter_dist = rt[i][j];
 if (via[i][j] == source)
 inter_dist = 9999;
 if (dist + inter_dist < rt[source][j]) {
 rt[source][j] = dist + inter_dist;
 via[source][j] = i;
 }
 }
 }
 }
 }
```

```
 static void update_tables() {
 int k = 0;
 for (int i = 0; i < 4 * v; i++) {
 update_table(k);
 k++;
 if (k == v)
 k = 0;
 }
 }
 static void init_tables() {
 for (int i = 0; i < v; i++) {
 for (int j = 0; j < v; j++) {
 if (i == j) {
 rt[i][j] = 0;
 via[i][j] = i;
 } else {
 rt[i][j] = 9999;
 via[i][j] = 100;
 }
 }
 }
 }
 static void print_tables() {
 for (int i = 0; i < v; i++) {
 for (int j = 0; j < v; j++) {
 System.out.print("Dist: " + rt[i][j] + " ");
 }
 System.out.println();
 }
 }
}
```

```
Please enter the number of Vertices:
4
Please enter the number of Edges:
5
Please enter data for Edge 1:
Source: 1
Destination: 2
Cost: 3
Please enter data for Edge 2:
Source: 2
Destination: 3
Cost: 4
Please enter data for Edge 3:
Source: 3
Destination: 4
Cost: 2
Please enter data for Edge 4:
Source: 4
Destination: 1
Cost: 1
Please enter data for Edge 5:
Source: 1
Destination: 3
Cost: 6

The initial Routing Tables are:
Dist: 0    Dist: 3    Dist: 3    Dist: 1
Dist: 3    Dist: 0    Dist: 4    Dist: 4
Dist: 3    Dist: 4    Dist: 0    Dist: 2
Dist: 1    Dist: 4    Dist: 2    Dist: 0

Please enter the Source Node for the edge whose cost has changed: 2
Please enter the Destination Node for the edge whose cost has changed: 3
Please enter the new cost: 1
```

```
The new Routing Tables are:
Dist: 0    Dist: 3    Dist: 3    Dist: 1
Dist: 3    Dist: 0    Dist: 1    Dist: 3
Dist: 3    Dist: 1    Dist: 0    Dist: 2
Dist: 1    Dist: 3    Dist: 2    Dist: 0
```

LSR:

```
import java.util.*;
public class LSR {
 public static void main(String[] args) {
```

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter the number of nodes : ");
int nodes = sc.nextInt();
int[] preD = new int[nodes];
int min = 999, nextNode = 0;
int[] distance = new int[nodes];
int[][] matrix = new int[nodes][nodes];
int[] visited = new int[nodes];
System.out.println("Enter the cost matrix");
// Input processing
for (int i = 0; i < nodes; i++) {
visited[i] = 0;
preD[i] = 0;
for (int j = 0; j < nodes; j++) {
matrix[i][j] = sc.nextInt();
if (matrix[i][j] == 0)
matrix[i][j] = 999;
}
}
distance = matrix[0];
visited[0] = 1;
distance[0] = 0;
// Dijkstra's algorithm
for (int counter = 0; counter < nodes; counter++) {
min = 999;
for (int i = 0; i < nodes; i++) {
if (min > distance[i] && visited[i] != 1) {
min = distance[i];
nextNode = i;
}
}
visited[nextNode] = 1;
for (int i = 0; i < nodes; i++) {
if (visited[i] != 1) {
if (min + matrix[nextNode][i] < distance[i]) {
distance[i] = min + matrix[nextNode][i];
preD[i] = nextNode;
}
}
}
```

```
 }
 // Output shortest paths and costs
 int j;
 for (int i = 0; i < nodes; i++) {
 if (i != 0) {
 System.out.print("Path from 0 to " + i + ": " + i);
 j = i;
 do {
 j = preD[j];
 System.out.print(" <- " + j);
 } while (j != 0);
 System.out.println();
 System.out.println("Cost = " + distance[i]);
 }
 System.out.println();
 }
 }
}
```

```
Enter the number of nodes : 4
Enter the cost matrix
0 1 3 0
1 0 1 7
3 1 0 2
0 7 2 0

Path from 0 to 1: 1 <- 0
Cost = 1

Path from 0 to 2: 2 <- 1 <- 0
Cost = 2

Path from 0 to 3: 3 <- 2 <- 1 <- 0
Cost = 4
```