P9) From a given a vertex in weighted connected graph, find the shortest paths to other vertices using Dijkstras algorithm. Write the program in Java.

```
import java.util.*;
public class Lab7 {
  public static void main(String[] args) {
  int i, j;
  int dist[] = new int[10], visited = new int[10],
   cost[][] = new int[10][10], path[] = new [10];
  Scanner in = new Scanner(System.in);
  System.out.println("** Dijkstra's Algorithm **");
  System.out.println("Enter the number of node:");
  int n = in.nextInt();
  System.out.println("Enter cost matrix:");
  for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  cost[i][j] = in.nextInt();
  System.out.println("Cost matrix Entered:");
  for(i=1;i<=n;i++){
    for(j=1;j<=n;j++) {
      System.out.print(cost[i][j]+"\t");
    }
    System.out.println();
  }
```

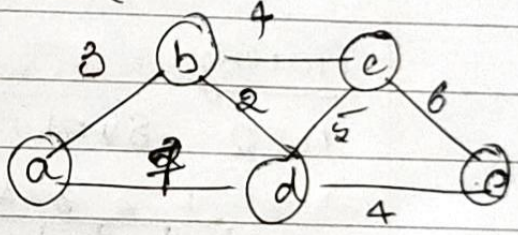
```
System.out.println("Enter the source vertex:");
int sv=in.nextInt();
dij(cost,dist,sv,n,path,visited);
printpath(sv,n,dist,path,visited);
System.out.println("\n ***");
}

static void dij(int cost[][],int dist[],
int sv,int n,int path[],int visited[])
{
int count=2,min,v=0;
for(int i=1;i<=n;i++)
{
visited[i]=0;
dist[i]=cost[sv][i];
if(cost[sv][i]==999)
path[i]=0;
else
path[i]=sv;
}
visited[sv]=1;

while(count<=n)
{
min=999;
```

```
for (int w=1; w<=n ; w++) {
if ((dist[w] <min) && (visited[w]==0))
{
    min=dist[w];
      v=w;
}}
visited[v]=1
count++;
for(int w=1; w<=n; w++)
{
  if ((dist[w]) >(dist[v]+cost[v][w]))
  {
    dist[w] = dist[v] + cost[v] [w];
    path[w] =v;
  }
}}}
static void printpath (int sv, int n, int
  dist[], int path[], int visited[])
{
 for (int w=1; w<=n; w++)
 {
  if(visited [w]==1 && w!=sv)
  {
    System.out.println("The shortest
  distance between "+sv+"->"+w" is :"+
```

```
dist[w]);
int t=path[w];
System.out.print("the path is:");
System.out.print(" "+w);
while(t!=sv)
{
  System.out.print("<-->"+t);
  t=path[t];
}
System.out.println("<-->"+sv);
}}}
```

Output:

```
** Dijkstras algorithm **
Enter the total number of nodes:
5
Enter the cost
0 3 999 7 999
3 0 4 2 999
999 4 0 5 6
7 2 5 0 4
999 999 6 4 0
The cost matrix:
0    3    999   7    999
3    0    4     2    999
```



999	4	0	5	6
7	2	5	0	4
999	999	6	4	0

Enter the source vertex:
1
The shortest distance between 1 → 2 is : 3
The path is : 2 ↔ 1
The shortest distance between 1 → 3 is : 7
The path is : 3 ↔ 2 ↔ 1
The shortest distance between 1 → 4 is : 5
The path is : 4 ↔ 2 ↔ 1
The shortest distance between 1 → 5 is : 9
The path is : 5 ↔ 4 ↔ 2 ↔ 1

* * * *

Tracing:

n = 5 sv = 1 cost :

	1	2	3	4	5
1	0	3	999	7	999
2	3	0	4	2	999
3	999	4	0	5	6
4	7	2	5	0	4
5	999	999	6	4	0

visited

1	2	3	4	5
0	0	0	0	0

dist (another row of the cost matrix)

1	2	3	4	5
0	3	999	7	999

path

1	2	3	4	5
1	1	0	1	0

no path from source to 3

can be reached from source

visited [1] = 1

1	2	3	4	5
1	0	0	0	0

* count = 2 ⇒ 2 <= 5

min = 999

→ w = 1   dist[1] ⇒ 0 < 999 && vis[1] ≠ 0

w = 2   dist[2] ⇒ 3 < 999 && vis[2] == 0

min = dist[2] = 3   v = 2

w = 3   dist[3] ⇒ 999 ≮ 3

w = 4   dist[4] ⇒ 7 ≮ 3 && vis[4] == 0

w = 5   dist[5] ⇒ 999 ≮ 3

(Selecting vertex (which is not visited) whose dist from source is smallest. || dist from source is stored in dist[])

visited [2] = 1   count = 3   v = 2

→ w = 1, if (dist[1] > dist[2] + cost[2][1])

0 > 3 + 3 ⇒ 0 ≯ 6

w = 2 if (dist[2] > dist[2] + cost[2][2])

(wasn't reachable from source) 3 ≯ 3 + 0   (3) reachable & dist has written

w = 3 if (dest[3] > dest[2] + cost[2][3])

999 > 3 + 4 ⇒ true

dist[3] = 7   path[3] = 2

w = 4 if (dest[4] > dest[2] + cost[2][4])

7 > 3 + 2 ⇒ true

dest[4] = 5   path[4] = 2

w = 5 if (dist[5] > dest[2] + cost[2][5])

999 ≯ 3 + 999

(Once vertex is selected Now this vertex act as source & modify dist & path.
1) Check if vertex is reachable from present source
if yes - check the distance obtained dist)

	1	2	3	4	5
visited	1	1	0	0	0

	1	2	3	4	5
dist	0	8	7	5	999

	1	2	3	4	5
path	1	1	2	2	0

count=3 ⇒ 3<=5 ⇒ true

min=999

↳ w=1 dest[w] < min && v[w] != 0

w=2 v[2] != 0

w=3 dest[3] < 999 && v[3] == 0 ⇒ true

7 < 999 && v[3] == 0

min=7 v=3

w=4 dist[4] < 7 && v[4] == 0

5 < 7 && v[4] == 0

min=5 v=4

w=5 dist[5] ≮ 5

visited[v]=1 count=4 v=4

↳ w=1 dest[1] < dist[4] + cost[4][1]

0 ≮ 5 + 999

w=2 dist[2] < dist[4] + cost[4][2]

3 ≮ 5 + 4

w=3 dist[3] ≮ dist[4] + cost[4][3] = 7 ≮ 5+5 (10)

w=4 dist[4] ≮ dist[4] + cost[4][4]

5 ≮ 3 + 5

w=5 dist[2] < dist[4] + cost[4][5]

999 < 5 + 4 = true

dist[5] = 9 path[5] = v = 4

	1	2	3	4	5
visited:	1	1	0	1	0
dist	0	3	7	5	9
path	1	1	2	2	4

↳ w=1 dist[1] > dist[5]

* count=4 < 5 , min=999.

↳ w=1 to 5 v[w]≠0

w=3 dist[w] = 7 < 999 & v[3] = 0

min = 7 v=3

visited[3] = 1 count = 5 v=3 d[3] = 7

↳ w=1 0 ∉ 7+..

w=2 3 ∉ 7+..

w=3 7 ∉ 7+..

w=4 5 ∉ 7+..

w=5 9 ∉ 7+6

	1	2	3	4	5
v:	1	1	1	1	0
dist	0	3	7	5	9
path	1	1	2	2	4

* count=5 ≤ 5 min=999

↳ w=5 d[5] < 999 & v[5]=0.

min=9 v=5

visited[5] = 1 count=5 d[5]=9

↳ w=1 d[0] ∉ d[5]+cost[5][0] w=4 5 ∉ 9+..

w=2 3 ∉ 9+.. w=5 9 ∉ 9+..

w=3 7 < 9+..

Printpath: sv = 1

↳ w = 1 v[w] = 1 & w ≠ sv → false

w = 2 v[w] = 1 & w ≠ sv → true

⇒ S.D 1 → 2 : dist[2] = 3

t = path[w] = 1

⇒ Path is : 2 ↔ 1

(w) (sv)

w = 3 v[w] = 1 & w ≠ sv → true

⇒ S.D 1 → 3 : 7

t = path[3] = 2

⇒ path is : 3 ↔ 2 ↔ 1

if t ≠ sv ⇒ 2 ≠ 1 (print t)

t = path[2] = 1

if 1 ≠ 1

print sv

Illy for w = 4 & 5