```java
//import java.util.Scanner;
/*
 * demo
 * vertices=5,
    wMat[][]=
    {{0,3,0,1,0},
    {3, 0, 1, 0, 0},
    {0, 1, 0, 0, 3},
   {1, 0, 0, 0, 7},
    {0,0, 3, 7, 0}}
 */
class Graph{
    int distance[],
     p[],
    vertices=8,
    wMat[][]=
    {{0, 2, 5, 7, 0, 0, 0, 0},
    {2, 0, 0, 9, 1, 0, 0, 0},
    {5, 0, 0, 3, 0, 0, 7, 2},
    {7, 9, 3, 0, 10, 0, 0, 5},
    {0, 1, 0, 10, 0, 2, 0, 4},
    {0, 0, 0, 0, 2, 0, 6, 4},
    {0, 0, 7, 0, 0, 6, 0, 8},
    {0, 0, 2, 5, 4, 4, 8, 0}}

    ,visited[];

    int nill=-1;
    int inft=1000;


    Graph(int v){
        //to create a user defined array put the value of v to vertices
        distance=new int[vertices];
        p=new int[vertices];
        distance[0]=0; //distance from source to source is zero.

        visited= new int[vertices];

        //wMat=new int[vertices][vertices];

       // Scanner sc= new Scanner(System.in);
        for(int i =0;i<vertices;i++){
            //for(int j=0;j<vertices;j++){
            //    System.out.println("if"+(i+1)+"th node & "+(j+1)+"th node are connected
put weight otherwise put 0");
            //    wMat[i][j]=sc.nextInt();

            //}
            if(i!=0){
                distance[i]=inft; //initially we know nothing about the distance from
source to all other nodes

            }
            p[i]=nill;
            visited[i]=0;// not yet visited

        }


    }
```

```
60      int mindistance(int ndistance[]){
61          /*it is going to return the index of the smallest element in this array,
    ndistance.
62          Ideally it needs to be implemented by priority queue, but to make it simple i am
    using basic linear approach.
63          */
64          //System.out.println("newdistance");
65          //for ( int i : ndistance) {
66          //    //System.out.print(i+",");
67          //}
68          int min=inft,index=0;
69          for(int i=0;i<vertices;i++){
70
71              if(visited[i]==0 && min>ndistance[i]){
72                  min=ndistance[i];
73                  index=i;
74              }
75          }
76          //System.out.println("min"+min+","+index);
77          return index;
78
79      }
80      /*
81       *
82       * function dijkstra(G, S)
83              for each vertex V in G
84                  distance[V] <- infinite
85                  previous[V] <- NULL
86                  If V != S, add V to Priority Queue Q
87              distance[S] <- 0
88
89              while Q IS NOT EMPTY
90                  U <- Extract MIN from Q
91                  for each unvisited neighbour V of U
92                      tempDistance <- distance[U] + edge_weight(U, V)
93                      if tempDistance < distance[V]
94                          distance[V] <- tempDistance
95                          previous[V] <- U
96              return distance[], previous[]
97       *
98       */
99      int[] calculateShortestDistance(){
100
101         for(int i=0;i<vertices;i++){
102             int u=mindistance(distance);
103
104             visited[u]=1;
105             //System.out.println("visited "+u);
106             for(int j=0;j<vertices;j++){
107
108                 if(visited[j]==0 && wMat[u][j]!=0){
109                     if(distance[j]>=(distance[u]+wMat[u][j])){
110                         distance[j]=distance[u]+wMat[u][j];
111                         p[j]=u;
112                     }
113                 }
114             }
115
116
117
118         }
119         return distance;
```

```
120     }
121     int[][] createShortPathGraph(){
122         int sWMat[][]=new int[vertices][vertices];
123         for(int i=0;i<vertices;i++){
124             //System.out.println("parent of "+i+" is "+p[i]);
125             for(int j=0;j<vertices;j++){
126 
127                 if(p[j]==i || p[i]==j){
128                     sWMat[i][j]=wMat[i][j];
129                 }
130             }
131         }
132 
133         return sWMat;
134     }
135 
136 }
137 public class dijkastra {
138 public static void main(String[] args) {
139     Graph g= new Graph(5);
140     int result[]=g.calculateShortestDistance();
141 
142     for (int i : result) {
143         System.out.println(i);
144     }
145     int result2[][]=g.createShortPathGraph();
146     for(int i=0;i<g.vertices;i++){
147         System.out.println("");
148         System.out.print("[");
149         for(int j=0;j<g.vertices;j++){
150             System.out.print(result2[i][j]+",");
151         }
152         System.out.print("]");
153     }
154 }
155 
156 }
157 
```