```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Dijkstra {



// Numero de vertices do gráfico
static final int V  = 9;


// funcao de utilidade para encontrar o vertice com a distância mínima,
// a partir do conjunto de vertices ainda não incluidas no
// caminho mais curto

private static int minDistance(int[] dist, boolean[] verticeProcesado)
{
   // Inicializar min valor
   int min = Integer.MAX_VALUE; int min_index=0;

   for (int v = 0; v < V; v++)
     if (verticeProcesado[v] == false && dist[v] <= min) {
         min = dist[v];
         min_index = v;
      }

   return min_index;
}

// funcao de utilidade para imprimir as distancias matriz calculados
private static void printSolution(int[] dist, int n)
{
   System.out.println("" distancia vertice da origem \ n "");
   for (int i = 0; i < V; i++)
      System.out.println(i + " \t\t " + dist[i]);
}

private static void dijkstra(int[][] grafo, int src)
{
     int[] dist = new int[V];
     // dist [i] armazena a distancia mais curta entre src para o vertice i

     boolean[] verticeProcesado = new boolean[V];
     //Este arranjo e verdade se o vértice i já foi processado

     // Inicializar todas as distâncias tão infinito e stpSet [] como falsa
     for (int i = 0; i < V; i++) {
        dist[i] = Integer.MAX_VALUE;
        verticeProcesado[i] = false;
     }
     // A distancia desde a origem ate o mesmo vertice e sempre 0
     dist[src] = 0;

     //Encontrar o caminho mais curto para todos os vértices
     for (int count = 0; count < V-1; count++)
     {

       // Pegue o vertice com a distancia mínima entre o vértice cojunto ainda não processados
       // src na primeira iteração sempre voltava
```

```
61.         int u = minDistance(dist, verticeProcesado);
62.
63.         // ele é marcado como já processados
64.         verticeProcesado[u] = true;
65.
66.         // Atualizacao do valor dist dos vertices adjacentes do vertice escolhido.
67.         for (int v = 0; v < V; v++)
68.
69.           // a dist é atualizado [v] só se for verticeProcesado, não é um
70.           // arco de uavy o peso total da estrada de src para percorrer ou é
71.           // menor do que o valor atual de dist [v]
72.           if (!verticeProcesado[v] && grafo[u][v] > 0 && dist[u] != Integer.MAX_VALUE
73.                                       && dist[u]+grafo[u][v] < dist[v])
74.              dist[v] = dist[u] + grafo[u][v];
75.       }
76.
77.       // a matriz é impresso com distâncias
78.       printSolution(dist, V);
79. }
80.
81. // programa para testar acima função
82. public static void main(String[] args)
83. {
84.    /* Vamos criar o gráfico exemplo discutido acima  */
85.    int[][] graph = {{0, 4, 0, 0, 0, 0, 0, 8, 0},
86.                     {4, 0, 8, 0, 0, 0, 0, 11, 0},
87.                     {0, 8, 0, 7, 0, 4, 0, 0, 2},
88.                     {0, 0, 7, 0, 9, 14, 0, 0, 0},
89.                     {0, 0, 0, 9, 0, 10, 0, 0, 0},
90.                     {0, 0, 4, 0, 10, 0, 2, 0, 0},
91.                     {0, 0, 0, 14, 0, 2, 0, 1, 6},
92.                     {8, 11, 0, 0, 0, 0, 1, 0, 7},
93.                     {0, 0, 2, 0, 0, 0, 6, 7, 0}
94.                    };
95.
96.     dijkstra(graph, 0);
97. }
98. }
```