DJIKSTRA ALGORITHM FOR WEIGHTED GRAPH

```
import java.util.*;
import java.lang.*;
import java.io.*;

class ShortestPath {
    // A utility function to find
the vertex with minimum distance
value,
    // from the set of vertices not
yet included in shortest path tree
    static final int V = 9;
    int minDistance(int dist[],
Boolean sptSet[])
    {
        // Initialize min value
        int min =
Integer.MAX_VALUE, min_index = -1;

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

        return min_index;
    }

    // A utility function to print
the constructed distance array
    void printSolution(int dist[],
int n)
    {
        System.out.println("Vertex
   Distance from Source");
        for (int i = 0; i < V; i++)
            System.out.println(i +
" tt " + dist[i]);
    }

    // Function that implements
Dijkstra's single source shortest
path
    // algorithm for a graph
represented using adjacency matrix
    // representation
    void dijkstra(int graph[][],
int src)
    {
        int dist[] = new int[V]; //
The output array. dist[i] will hold
        // the shortest distance
from src to i
```

```
        // sptSet[i] will true if
vertex i is included in shortest
        // path tree or shortest
distance from src to i is finalized
        Boolean sptSet[] = new
Boolean[V];

        // Initialize all distances
as INFINITE and stpSet[] as false
        for (int i = 0; i < V; i++)
{
            dist[i] =
Integer.MAX_VALUE;
            sptSet[i] = false;
        }

        // Distance of source
vertex from itself is always 0
        dist[src] = 0;

        // Find shortest path for
all vertices
        for (int count = 0; count <
V - 1; count++) {
            // Pick the minimum
distance vertex from the set of
vertices
            // not yet processed. u
is always equal to src in first
            // iteration.
            int u =
minDistance(dist, sptSet);

            // Mark the picked
vertex as processed
            sptSet[u] = true;

            // Update dist value of
the adjacent vertices of the
            // picked vertex.
            for (int v = 0; v < V;
v++)


                // Update dist[v]
only if is not in sptSet, there is
an
                // edge from u to
v, and total weight of path from
src to
                // v through u is
smaller than current value of
dist[v]
                if (!sptSet[v] &&
graph[u][v] != 0 &&
```

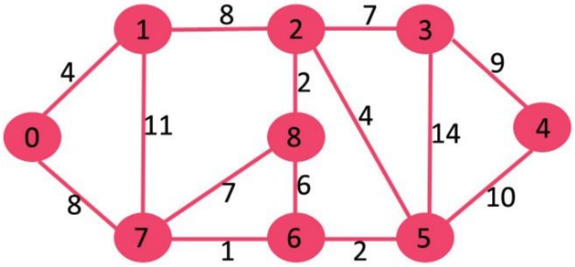
DJIKSTRA ALGORITHM FOR WEIGHTED GRAPH

```
                   dist[u] !=
Integer.MAX_VALUE && dist[u] +
graph[u][v] < dist[v])
                    dist[v] =
dist[u] + graph[u][v];
        }

        // print the constructed
distance array
        printSolution(dist, V);
    }

    // Driver method
    public static void
main(String[] args)
    {
        /* Let us create the
example graph discussed above */
        int graph[][] = new int[][]
{
{ 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

{ 0, 0, 2, 0, 0, 0, 6, 7, 0 }
};
ShortestPath t = new
ShortestPath();
    t.dijkstra(graph, 0);
    }
}
```

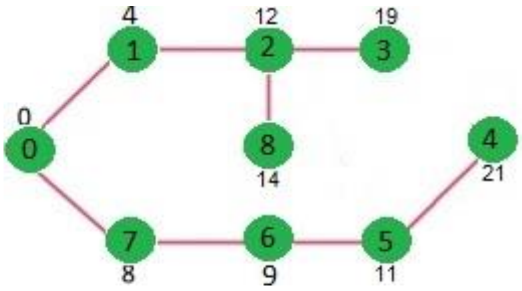
OUTPUT:

```
Vertex   Distance from Source

0                  0

1                  4

2                  12

3                  19

4                  21

5                  11

6                  9

7                  8

8                  14
```

EXAMPLE:



Shortest Path Tree (SPT)



4
1
12
2
19
3
0
0
8
14
4
21
7
8
6
9
5
11