static final int V = 9;

static int minDistance(int dist[], Boolean sptSet[]) {

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;

}

static void printSolution(int dist[]) {

System.out.println("Vertex \t\t Distance from Source");

for (int i = 0; i < V; i++) {

System.out.println(i + " \t\t " + dist[i]);

}

}

static void dijkstra(int graph[][], int src) {

int dist[] = new int[V]; // 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 && dist[u] != Integer.MAX\_VALUE && dist[u] + graph[u][v] < dist[v]) {

dist[v] = dist[u] + graph[u][v];

}

}

}

printSolution(dist);

}