```
/*_____________________________________________________________________
Assignment no :-3
Title:- Implement Greedy search algorithm for Dijkstra's Minimal Spanning Tree.
Name:-Pravin Jain      Roll No:-74     Batch:-T4       Subject:-AI
_____________________________________________________________________*/
class Dijkstra{
   // 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 printShortestPaths(int dist[], int src)
   {
      System.out.println("" +
            "|-----------|----------------------|" +
            "\n| Vertex \t"+"|"+"\tDistance from Source|\n" +
            "|-----------|----------------------|");
      for (int i = 0; i < V; i++){
```

```
            System.out.println("|   "+src+"->"+i + " \t| \t\t\t " + dist[i]+"\t\t\t|");
        }
        System.out.println("|-----------|----------------------|");
    }

    // Function that implements Dijkstra's single source the shortest
    // path algorithm for a graph represented using adjacency matrix representation

    void computeShortestPath(int graph[][], int src)
    {
        int dist[] = new int[V]; // The output array. Here, dist[i] will hold the shortest distance
from src to vertex i

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

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

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

        // Find the 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, shortestPathTreeSet);
```

```
        // Mark the picked vertex as processed
        shortestPathTreeSet[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 shortestPathTreeSet, 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]
            //d(u) + c(u,v) < d(v)
            if (!shortestPathTreeSet[v] && graph[u][v] != 0 && dist[u] !=
Integer.MAX_VALUE && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
      }

      // print the constructed distance array
      printShortestPaths(dist,src);
   }

   // Driver method
   public static void main(String[] args)
   {
      //Creating a graph by the means of adjacency matrix
      int graph[][] = new int[][]{
            //0  1  2  3  4  5  6  7  8
            { 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 }
        };
        System.out.println("\nAdjacency Matrix :- ");
        System.out.print("\t");
        for(int k=0;k<graph.length;k++){
           System.out.print(k+"\t");
        }
        System.out.println("\n\t|---------------------------------|");
        for(int x=0;x<graph.length;x++){
           System.out.print(x+"\t|");
           for(int y=0;y<graph[x].length;y++){
              System.out.print(graph[x][y]+"\t");
           }System.out.print("|");System.out.println("\n\t|---------------------------------|");
        }

        System.out.println("\n\nShortest Paths from Source to ith vertex :- ");
        Dijkstra t = new Dijkstra();
        t.computeShortestPath(graph, 0);
    }
}

/*_____
_______
Output:-
Adjacency Matrix :-
        0       1       2       3       4       5       6       7       8
        |---------------------------------|
0       |0      4       0       0       0       0       0       8       0       |
```

|---|---|---|---|---|---|---|---|---|---|
1	4	0	8	0	0	0	0	11	0
2	0	8	0	7	0	4	0	0	2
3	0	0	7	0	9	14	0	0	0
4	0	0	0	9	0	10	0	0	0
5	0	0	4	14	10	0	2	0	0
6	0	0	0	0	0	2	0	1	6
7	8	11	0	0	0	0	1	0	7
8	0	0	2	0	0	0	6	7	0

Shortest Paths from Source to ith vertex :-

Vertex	Distance from Source
0->0	0
0->1	4
0->2	12
0->3	19
0->4	21
0->5	11
0->6	9

```
|  0->7       |                    8                    |
|  0->8       |                    14                    |
|-----------|-----------------------|
```

______*/