```
 1  using System;
 2  using System.Collections.Generic;
 3
 4  namespace coursework
 5  {
 6      class Graph
 7      {
 8          List<Tuple<int, string>> vertices = new List<Tuple<int, string>>();
 9          int graphSize;
10          int[,] adjMatrix;
11
12          public Graph(int graphSize)
13          {
14              this.graphSize = graphSize;
15              adjMatrix = new int[graphSize, graphSize];
16          }
17
18          public void AddNode(string value)
19          {
20              vertices.Add(new Tuple<int, string>(vertices.Count, value));
21          }
22
23          public void AddEdge(int indexA, int indexB, int weight, int direction)
24          {
25              adjMatrix[indexA, indexB] = weight;
26              if (direction == 2)
27              {
28                  adjMatrix[indexB, indexA] = weight;
29              }
30          }
31
32          public void RemoveEdge(int indexA, int indexB)
33          {
34              adjMatrix[indexA, indexB] = 0;
35              adjMatrix[indexB, indexA] = 0;
36          }
37
38          public void Display() //displays the adjacency matrix
39          {
40              Console.WriteLine("**********Adjacency Matrix
                  Representation**********");
41              Console.WriteLine("Number of nodes: {0}\n", graphSize);
42              foreach (Tuple<int, string> n in vertices)
43              {
44                  Console.Write("\t{0}", n.Item2);
45              }
46              Console.WriteLine();//newline for the graph display
47              for (int i = 0; i < graphSize; i++)
48              {
49                  Console.Write("{0}\t", vertices[i].Item2);
50                  for (int j = 0; j < graphSize; j++)
51                  {
52                      Console.Write("{0}\t", adjMatrix[i, j]);
53                  }
54                  Console.WriteLine();
55                  Console.WriteLine();
```

```
56              }
57          }
58          /// <summary>
59          /// dijkstras shortest path finding algorithm
60          /// </summary>
61          /// <param name="source"></param>
62          /// <returns></returns>
63          public int[,] dijksta(int source)
64          {
65
66              int current = source; //index of source node
67              int previous = source;
68              int[,] distances = new int[graphSize, 2];
69              PriorityQueue queue = new PriorityQueue(graphSize);
70              List<int> unvisited = new List<int>(graphSize);
71              int count = 0;
72              foreach (Tuple<int, string> n in vertices)
73              {
74                  unvisited.Add(n.Item1);
75                  distances[count, 0] = 9999;
76                  distances[count, 1] = -1;
77                  count++;
78              }
79              distances[source, 0] = 0;
80              distances[source, 1] = source;
81              while (unvisited.Count != 0)
82              {
83                  unvisited.Remove(current);
84                  for (int i = 0; i < graphSize; i++)
85                  {
86                      if (adjMatrix[current, i] != 0 && unvisited.Contains(i))
87                      {
88                          if (distances[i, 0] == 9999)
89                          {
90                              queue.enQueue(vertices[i].Item1, adjMatrix
                        [current, i] + distances[previous, 0]);
91                              distances[i, 0] = adjMatrix[current, i] +
                        distances[previous, 0];
92                              distances[i, 1] = previous;
93
94                          }
95                          else if (distances[i, 0] > adjMatrix[current, i] +
                        distances[previous, 0])
96                          {
97                              distances[i, 0] = adjMatrix[current, i] +
                        distances[previous, 0];
98                              distances[i, 1] = previous;
99                              queue.UpdateQueue(vertices[i].Item1, adjMatrix
                        [current, i]);
100                         }
101                     }
102                 }
103                 if (queue.isEmpty())
104                 {
105                     unvisited.Clear();
106                 }
```

```
107                     current = queue.deQueue();
108                     previous = current;
109                 }
110                 return distances;
111             }
112         }
113     }
114 
```