```
 1 #!/usr/bin/python3
 2
 3
 4 from CS312Graph import *
 5 import time
 6
 7
 8 class NetworkRoutingSolver:
 9     def __init__( self, display ):
10         pass
11
12
13
14     def initializeNetwork( self, network ):
15         assert( type(network) == CS312Graph )
16         self.network = network
17
18
19
20     def getShortestPath( self, destIndex ):
21         """
22         Find the shortest path from the source node to the
   destination node
23         Dijkstra's algorithm has already been run
24         We are only finding the specific shortest path and
   cost for each node
25
26         :param destIndex: The index of the destination node
    we are interested in finding the shortest path of
27         :return: Returns a dictionary with the cost and the
    path as keys
28         """
29         self.dest = destIndex
30
31         # TODO: RETURN THE SHORTEST PATH FOR destIndex
32         #       INSTEAD OF THE DUMMY SET OF EDGES BELOW
33         #       IT'S JUST AN EXAMPLE OF THE FORMAT YOU'LL
34         #       NEED TO USE
35
36         path_edges = []
37         total_length = 0
38
39         node = self.network.nodes[self.source]
40         edges_left = 3
41
42         while edges_left > 0:
43             edge = node.neighbors[2]
44             path_edges.append( (edge.src.loc, edge.dest.loc
   , '{:.0f}'.format(edge.length)) )
45             total_length += edge.length
```

```
46
47              node = edge.dest
48              edges_left -= 1
49
50          return {'cost':total_length, 'path':path_edges}
51
52
53
54      def computeShortestPaths( self, srcIndex, use_heap=
    False ):
55          """
56          Compute the shortest path from the source node to
    every other node in the graph
57
58
59          :param srcIndex: Specifies the source node used in
    Djikstra's
60          :param use_heap: Determines whether to use heap or
    array priority queue
61          :return: Time it took to compute Dijkstra's
62          """
63          self.source = srcIndex
64
65          t1 = time.time()
66          # TODO: RUN DIJKSTRA'S TO DETERMINE SHORTEST PATHS.
67          #       ALSO, STORE THE RESULTS FOR THE SUBSEQUENT
68          #       CALL TO getShortestPath(dest_index)
69
70          t2 = time.time()
71
72          return (t2-t1)
73
74      """
75      We will need a function that can perform dijkstra's
    algorithm
76      Function dijkstra G, s
77          • G = V, E is the weighted directed graph
78          • s is the starting node
79          • Returns the shortest distance from s to every
80          other vertex
81
82          • Foreach u ∈ V:
83          • u. dist = ∞
84          • u. prev = None
85          • s. dist = 0
86          • H = priorityqueue V # distances as keys
87          • While H is not empty:
88              • u = H. getnext() # gets the smallest item from
    the
89              priority queue, deleting it from the queue
```

```
90              • Foreach e = u, v ∈ E and v ∈ Q (not visited
    ):
91                  • If v. dist > u. dist + e. length
92                  • v. dist = u. dist + e. length
93                  • v. prev = u
94                  • H. updatekey(v, v. dist)
95  """
96
97  class PriorityQueue:
98      """
99      Implement a priority queue that either uses an array
    or a heap
100
101     getnext: Gets the next item with the smallest key
102         Runs V times
103 •   updatekey (and insert): Updates the key of a desired
    vertex
104 •       Runs V + E times
105     """
106     def __init__(self, use_heap=False):
107         pass
108
```