```
"""
Brian Tremaine  April 3, 2019
Python implementation of Dijkstra's shortest path algorithm
TremaineConsultingGroup.com

pseudo code:
function Dijkstra(Graph, source):
       create list of all vertices Q
       create list of all visited vertices, S

       for each vertex v in Graph:
           dist[v] ← INFINITY
           prev[v] ← UNDEFINED
           add v to Q
      dist[source] ← 0

      while Q is not empty:
          u ← vertex in Q with min dist[u]
          S ← add u to S
          remove u from Q

          for each neighbor v of u:       // only v that are still in Q
              alt ← dist[u] + w(u, v)
              if alt < dist[v]:
                  dist[v] ← alt
                  prev[v] ← u

      return dist[], prev[]
  end function
"""
import numpy as np

def spf(edges, start, end):
    """ 'edges' is a list of edge pairs with connecting nodes and cost
        'start' and 'end' are the respective start & end nodes.
    """

    # create a graph dict from edges
    Graph = graph(edges)

    INFINITY = 999999
    unique_v = unique(edges)

    # initialize dict
    dist = {}
    prev = {}
    for i in unique_v:
        dist.update({i:INFINITY})
        prev.update({i:-INFINITY})

    dist[start] = 0
    prev[start] = 0

    Q= []  # list of unvisited nodes
    S= []  # list of visited nodes
    for v in range(len(unique_v)):
        Q.append(unique_v[v])

    while Q:
       # find vertex in Q with minimum dist and remove from list
       u = minVertex(Q, dist)  # u is str label
       Q.remove(u)
       S.append(u)

       # search neighbors of u in graph, updating dist and prev
       neighbors = Graph[u]         # returns dict of neighbors
       for v in neighbors.keys():
```

```
            # cost function update
            if dist[v] > dist[u] + neighbors[v]:
               dist[v] = dist[u] + neighbors[v]
               prev[v] = u

    # update output return variables
    path= prev
    length = dist[end]

    return path, length, dist

def unique(list):
    v = []
    for i in range(len(list)):
        v.append(list[i][0])
        v.append(list[i][1])
        values = np.unique(v)

    return values.tolist()

def minVertex(Q, dst):
    # return vertex str in Q with minimum distance
    temp = {}
    for j in Q:
        temp.update({j:dst[j]})

    return min(temp, key=temp.get)

def graph(edges):
    # from a list of edges create a graph dictionary
    g = {}

    for i in range(len(edges)):
        if edges[i][0] in g:
           u = g[ edges[i][0] ]
           u.update({edges[i][1]:edges[i][2]})
           g.update({edges[i][0]:u})
        else:
           u = {edges[i][1]:edges[i][2]}
           g.update({edges[i][0]:u})

    return g

# ============================================================================
if __name__ == '__main__':

    # initialize the list of edges as a node pair and a cost
    """ node1, node2, length """

    # ========================================================================
    # ========================================================================
    # graph for specific problem goes next, list of edges

    # Example 1: undirected graph with edges repeated to fill
    # directed graph with matching weights.
    Edges= \
    [['A', 'C', 9],
     ['A', 'B', 14],
     ['A', 'D', 7],
     ['B', 'C', 14],
     ['B', 'F', 9],
     ['C', 'D', 10],
     ['D', 'E', 15],
     ['F', 'E', 6], # ...
     ['C', 'A', 9],
     ['B', 'A', 14],
     ['D', 'A', 7],
```

```
        ['C', 'B', 14],
        ['F', 'B', 9],
        ['D', 'C', 10],
        ['E', 'D', 15],
        ['E', 'F', 6]]


    # Example 2: directed graph with unequal weights between
    # any two nodes

    # If the Example 2 is uncommented it we will run instead of
    # Example 1. Be sure to update 'start' and 'end'

    Edges=\
    [['s','a',2],
     ['s','b',1],
     ['a','s',3],
     ['a','b',4],
     ['a','c',8],
     ['b','s',4],
     ['b','a',2],
     ['b','d',2],
     ['c','a',2],
     ['c','d',7],
     ['c','t',4],
     ['d','b',1],
     ['d','c',11],
     ['d','t',5],
     ['t','c',3],
     ['t','d',5]]

    # ================================================================================

    # call Dijkstra's spf algorithm
    start = 's'
    end = 't'
    path, length, weights = spf(Edges, start, end)

    # results
    #print(graph(Edges))

    print()
    print('path:', start, 'to', end, path)
    print('length:', length)
    print('weights', weights)

```