Shortest Distance Function

June 19, 2022

[51]:
```
def ShortestDistance (timeOfDay, Day, start, end, fileName='EdgesWeights.txt',␣
 ↪accessMode='w+'):

    import numpy as np

    def trafficFunction(trafficCond, Time):
        if trafficCond == 'G':
            traffic = 1.1*Time
        elif trafficCond == 'O':
            traffic = 1.25*Time
        elif trafficCond == 'R':
            traffic = 2*Time
        elif trafficCond == 'B':
            traffic = 3*Time
        elif trafficCond == 'NA':
            traffic = 0
        return traffic
# These are test numbers to see if the function works.
# I will study the google maps times later to get more accurate functions

    def Noah(start, end, fileName):

        import os

        file2 = open('input.txt', "w")
        file2.write(start + " " + end)
        file2.close()

        text = './bin/project {0} < input.txt'.format(fileName)
        p = os.popen(text)
        print(p.read())
        p.close()
```

```python
distances = [('1A', 377), ('1B', 377), ('2B', 528), ('2D', 528), ('3B',␣
↪1056), ('3E', 1056), ('4A', 367), ('5D', 528), ('5M', 528), ('6C', 430),␣
↪('6F', 430), ('7F', 400), ('7G', 400), ('8G', 302), ('8H', 302), ('9H',␣
↪528), ('9I', 528), ('10I', 528), ('10J', 528), ('11I', 300), ('11K', 300),␣
↪('12J', 318), ('12L', 318), ('13K', 350), ('13D', 350), ('14K', 528),␣
↪('15L', 400), ('15M', 400), ('16M', 2112), ('16N', 2112), ('17E', 528),␣
↪('17N', 528), ('18N', 528), ('18O', 528), ('19D', 528), ('19F', 528),␣
↪('20K', 528)]
input1 = [('1', 'A', 'B'), ('2', 'B', 'D'), ('3', 'B', 'E'), ('4', 'A',␣
↪'C'), ('5', 'D', 'M'), ('6', 'C', 'F'), ('7', 'F', 'G'), ('8', 'G', 'H'),␣
↪('9', 'H', 'I'), ('10', 'I', 'J'), ('11', 'I', 'K'), ('12', 'J', 'L'),␣
↪('13', 'D', 'K'), ('14', 'K', 'L'), ('15', 'L', 'M'), ('16', 'M', 'N'),␣
↪('17', 'E', 'N'), ('18', 'N', 'O'), ('19', 'D', 'F'), ('20', 'G', 'K')]
input2 = [('A', '1', '4'), ('B', '1', '2', '3'), ('C', '4', '6'), ('D',␣
↪'2', '5', '13', '19'), ('E', '3', '17'), ('F', '6', '7'), ('G', '7', '8'),␣
↪('H', '8', '9'), ('I', '9', '10', '11'), ('J', '10', '12'), ('K', '11',␣
↪'13', '14', '20'), ('L', '12', '14', '15'), ('M', '5', '15', '16'), ('N',␣
↪'16', '17', '18'), ('O', '18')]

trafficFile = 'HodgesNeyland616.txt' #change as we record more traffic data
trafficConditions = np.loadtxt(trafficFile, dtype=str)

length = len(distances)
file = open(fileName, accessMode)
file.write("%s\n" %(len(input2)))
minute = timeOfDay % 100
index2 = int (minute / 15 + ( (timeOfDay - minute) - 600 ) / 25 * 40 )
for i in range(length):
    edgen = distances[i][0]
    j = 0
    while edgen[j].isnumeric():
        edge = edgen[:j+1]
        j+=1
    k = 1
    node1 = edgen[k:] #from
    if not(node1.isalpha()):
        k+=1
        node1 = edgen[k:]
    index1 = int(edge) - 1
    node2 = input1[index1][1] #to
    if node2 == node1:
        node2 = input1[index1][2]
    dist = distances[i][1]
    index2 += 1
    thisCond = trafficConditions[index2][2]
    Time = dist / 44
        # speed limit in Knoxville ~ 30 mph is 44 ft/s
        traffic = trafficFunction(thisCond, Time)
        thisTravelTime = '{0}'.format(Time + traffic)
        file.write("%s %s %s\n" %(node1, node2, thisTravelTime))
        ## need to change function when we have more than one day.

    print(file.read())

    start = input("Starting node: ")
    end = input("Ending node: ")

    Noah(start, end, fileName)
```

[144]:
```
def shortestdistance ():

#inputing the variables for the function

    timeOfDay = input('Enter the time of day in military time: ')

    #error check the timeOfDay input. It must be an int.
    while not timeOfDay.isnumeric():
        timeOfDay = input('Invalid input for time of day. Please enter time of␣
 ↪day as integer in military time.')

    #type casting timeOfDay from string to int to use in the function.
    timeOfDay = int(timeOfDay)


    Day = input('Enter the day: ')
    Day = Day.upper()
    Day = Day[:3]

    #error check the Day input. It must be one of the seven days of the week,␣
 ↪and it must be atleast 3 letters.
    while Day != 'SUN' and Day!='MON' and Day!='TUE' and Day!='WED' and Day!
 ↪='THU'and Day!='FRI' and Day!='SAT' and Day!= 'SUN':
        Day = input('Invalid input for day. Please enter a day of the week: ')
    while len(Day) < 3:
        Day = input('Invalid input for day. Please enter atleast 3 letters: ')

    start = input('Enter the starting node: ')
    start = start.upper()

    #error check the starting node input. It must be a letter.
    while not start.isalpha():
        start = input('Invalid starting node. Please enter starting node as a␣
 ↪char or string of letters: ')
```

```
    end = input('Enter the ending node: ')
    end = end.upper()

    #error check the ending node input. It must be a letter.
    while not end.isalpha():
        end = input('Invalid ending node. Please enter ending node as a char or␣
 ↪string of letters: ')

    fileName = './trafficFiles/{0}{1}.txt'.format(Day, timeOfDay)

    accessMode='w+'

    import numpy as np


# This function returns the travel time between 2 nodes, given the time if the␣
 ↪driver
# was going the speed limit and the traffic condition. Each traffic condition␣
 ↪will affect
# the travel time between the 2 nodes.
    def trafficTime(trafficCond, PTime):

        if trafficCond == 'G':
            traffic = 1.1*PTime

        elif trafficCond == 'O':
            traffic = 1.25*PTime

        elif trafficCond == 'R':
            traffic = 2*PTime

        elif trafficCond == 'B':
            traffic = 3*PTime

        elif trafficCond == 'NA':
            traffic = 0

        return traffic
    # These are test numbers to see if the function works.
    # I will study the google maps times later to get more accurate functions



# This function will run Noah's program given the start and end nodes and then␣
 ↪store the
# results in the given filename.
```

```
    def PathTime(start, end, fileName):

        import os

        file2 = open('input.txt', "w")
        file2.write(start + " " + end)
        file2.close()

        text = './bin/project {0} < ./input.txt'.format(fileName)
        p = os.popen(text)
        print(p.read())
        p.close()


# the distances vector has the form (edge, distance in feet). It will be used␣
 ↪to determine the PTime for the trafficFunction
    distances = [('1A', 377), ('1B', 377), ('2B', 528), ('2D', 528), ('3B',␣
 ↪1056), ('3E', 1056), ('4A', 367), ('5D', 528), ('5M', 528), ('6C', 430),␣
 ↪('6F', 430), ('7F', 400), ('7G', 400), ('8G', 302), ('8H', 302), ('9H',␣
 ↪528), ('9I', 528), ('10I', 528), ('10J', 528),
                   ('11I', 300), ('11K', 300), ('12J', 318), ('12L', 318),␣
 ↪('13K', 350), ('13D', 350), ('14K', 528), ('15L', 400), ('15M', 400),␣
 ↪('16M', 2112), ('16N', 2112), ('17E', 528), ('17N', 528), ('18N', 528),␣
 ↪('18O', 528), ('19D', 528), ('19F', 528), ('20K', 528)]

# input1 is a vector that lists each edge with its nodes. It will be used to␣
 ↪determine the second node when writing the weights file.
    input1 = [('1', 'A', 'B'), ('2', 'B', 'D'), ('3', 'B', 'E'), ('4', 'A',␣
 ↪'C'), ('5', 'D', 'M'), ('6', 'C', 'F'), ('7', 'F', 'G'), ('8', 'G', 'H'),␣
 ↪('9', 'H', 'I'), ('10', 'I', 'J'),
                ('11', 'I', 'K'), ('12', 'J', 'L'), ('13', 'D', 'K'), ('14', 'K',␣
 ↪'L'), ('15', 'L', 'M'), ('16', 'M', 'N'), ('17', 'E', 'N'), ('18', 'N',␣
 ↪'O'), ('19', 'D', 'F'), ('20', 'G', 'K')]

# input2 is a vector that lists each node with its possible edges. It will be␣
 ↪used to determine the number of nodes (the length). This will be printed in␣
 ↪the txt file.
    input2 = [('A', '1', '4'), ('B', '1', '2', '3'), ('C', '4', '6'), ('D',␣
 ↪'2', '5', '13', '19'), ('E', '3', '17'), ('F', '6', '7'), ('G', '7', '8'),␣
 ↪('H', '8', '9'), ('I', '9', '10', '11'),
                ('J', '10', '12'), ('K', '11', '13', '14', '20'), ('L', '12',␣
 ↪'14', '15'), ('M', '5', '15', '16'), ('N', '16', '17', '18'), ('O', '18')]

# Reading the file with the form HodgeNeylandXY.txt that has the traffic␣
 ↪condition for each edge from time X to time Y.
```

```
    trafficFile = 'HodgesNeyland616.txt' #NOTE: change as we record more␣
↪traffic data and change the traffic file

    #converting trafficFile into a vector.
    trafficConditions = np.loadtxt(trafficFile, dtype=str)

    # length is for the for loop below. This is how many data points we will␣
↪enter.
    length = len(distances)

    # creating new file to write the data in
    file = open(fileName, accessMode)

    # len(input2) is the number of nodes.
    file.write("%s\n" %(len(input2)))

    # minute can be 0, 15, 30, 45
    minute = timeOfDay % 100

    # index2 will be the index of the trafficConditions vector that we are␣
↪referring to.
    # 600 is 0, 615 is 40, ...
    index2 = int (160*(timeOfDay-minute-600)/100 + 40*(minute/15))


    for i in range(length):
        # edgen is the edge name. It will have the form '2B' or '11I'.
        edgen = distances[i][0]

        # Defining edge, which will be '2' or '11'. We start by saying only the␣
↪first element in the string
        # is the edge, but we keep adding numbers to our edge name until we␣
↪find an element that is a letter,
        # taking care of cases where the edge is more than 1 digit.
        j = 0
        while edgen[j].isnumeric():
            edge = edgen[:j+1]
            j+=1

        # Defining node1, the from node, which will be the rest of the edgen␣
↪vector that was not used for edge.
        node1 = edgen[j:]
        # index1 will be used to access the input1 list. index1 is edge - 1␣
↪because input1 is 0-indexed.
        index1 = int(edge) - 1

        # node2 will either be the 2nd or 3rd element in the list accessed by␣
↪input1[index1]. We assign node2
        # to the 2nd element first. The if statement checks if node2 is the␣
↪same as node1. If they are, then we
        # assign node2 to the 3rd element.
        node2 = input1[index1][1] #to
        if node2 == node1:
            node2 = input1[index1][2]

        # dist accesses the specific distance at the edge we are working with.␣
↪It will be the distance we
        # use to calculate the PTime for the trafficTime function.
        dist = distances[i][1]


        # defining thisCond, which will use index2 to access the traffic␣
↪condition at this edge and this time.
        thisCond = trafficConditions[index2][2]

        # PTime will be the distance divided by the speed limit in Knoxville␣
↪(25 mph = 36.666666666667 ft/s )
        PTime = dist / 36.66666666666666667

        # traffic will be the extra time spent driving due to the traffic␣
↪conditions.
        traffic = trafficTime(thisCond, PTime)

        # incrementing index2 by 1 because trafficCond goes in the same order␣
↪as the distances list.
        index2 += 1

        # defining travel time and writing on the file.
        thisTravelTime = '{0}'.format(PTime + traffic)
        file.write("%s %s %s\n" %(node1, node2, thisTravelTime))

        ## need to change function when we have more than one day. ##

    # reads the text file written in the for loop, sends into Noah's program.␣
↪Noah's program will print
    # the fastest route.
    PathTime(start, end, fileName)
```