Description:

* You call main.py and pass it the input file (tsp\_example\_0.txt).
* It will create the output file name with the .tour, then read in the input file into a single array of characters.
* It will break up the array by the instructions given (first is n cities, rows of ) into a matrix of city vertices (discarding city identifiers as this will be the index in the list).
* It runs the list of city vertices through math to get a matrix of weights between each city.
* Start recording time > call tsp function (returns) > end recording time > outputs to file
* In tsp function:
  + Catch graph (matrix of weights)
  + Loop n-1 (n = number of cities) times, setting min found weight and vertex to infinity
  + Nested loop n times check if current n is in last walked to graph from weight matrix. If it is, then skip until n is not in walk.
  + Check if n of graph is less than min weight recorded. If it is, record that weight and n
  + Push min n found to walk and add min weight to tour length (val)
  + After loops: manually append walk back to source (0) and add length to val
  + Format list to be printed easily
  + Return list: [tour length, city first visited, next city, next city, … ]

The tsp function has a complexity of O(n^2) since it only has a nested for loop.

tsp-verifier.py confirmed all cities were visited.

I thought about how an MST could be augmented for TSP, it would need to keep track of the visited cities, choose the smallest path from the current city, and add a return to the source city.

def tsp(graph):

val = 0 # tour length

walk = [0] # cities visited # start at 0

for i in graph.length()-1:

minW = float('infinity') # min weight # set to max

minV = float('infinity') # min vertex # set to max

for ii in graph.length():

if(ii in walk): # check if current ii is a city already walked to

Continue # if is, skip

elif((graph[walk[-1]][ii] != 0) and (graph[walk[-1]][ii] < minW)):

# check if next city is not its self, and its weight is less than min

minW = graph[walk[-1]][ii] # record new min’s

minV = ii

walk.append(minV) # add new visited city to walk

val += minW # add tour length

val += graph[walk[-1]][0] # manually add return to 0

# Format walk to be pretty

return walk

def main(fileIn):

fileOut = fileIn+".tour" # get fileOut name

arr = [] # master array of file ints

Open file as read

arr.append(each char from file individually)

arr.pop(0) # get number of cities

Format arr into 2d arr of city x,y

Loop through formatted arr with math, add to graph[]:

round(math.sqrt(pow((x1-x2),2) + pow((y1-y2),2))))

Start time recording

tour = tsp(graph) # call tsp and get returned arr

End time recording

Open output file as w

Print tour to file

| Tour | Time | Ratio | My Length |
| --- | --- | --- | --- |
| 0 | < 1 second | 1.0 | 14 |
| 1 | < 1 second | 1.4 | 150393 |
| 2 | < 1 second | 1.3 | 3256 |
| 3 | < 1 second | 1.1 | 5926 |
| 4 | < 1 second | 1.3 | 9503 |
| 5 | 20 seconds | 1.2 | 28685 |
| 6 - custom test | < 1 second | 1.0 | 21 |