Data Structures and Algorithms for Information Processing   **Project 4**
Due: Midnight Tuesday, March 28, 2023

Topics: The traveling Salesperson Problem (TSP), Minimum Spanning Trees (MST), Heaps and Graphs

There are three parts to this assignment.

Part 1. Using an approximation algorithm to solve TSP (60%)

Given an undirected graph G = (V, E) we want to find a minimum length cycle that visits each vertex once and then returns to the start vertex. In general, this problem is NP-Hard. In this exercise, we will find a cycle that may not be of minimal length but will be no more than twice the minimum. The solution that we will build will run in polynomial time.

Your task is to implement the following algorithm from "Introduction to Algorithms" by Cormen, Lieserson, Rivest and Stein:

Approx-TSP-Tour(G,c)
1. Select a vertex r ε V[G] to be a root vertex
2. Compute a minimum spanning tree T for G from root r using MST-Prim(G,c,r)
3. Let L be the list of vertices visited in a preorder tree walk of T
4. Return the Hamiltonian cycle H that visits the vertices in the order L

The data file that you will use is found on the course schedule and is named: http://www.andrew.cmu.edu/user/mm6/95-771/CrimeData/CrimeLatLonXY1990.csv. This file contains a list of crime records from 1990 in Pittsburgh. It contains three types of addresses. The first, which we will not use, is the street address of the crime. The second is the latitude and longitude of the crime (we will use these data only in Part 3 of this assignment. The third is the (X,Y) - coordinates of the crime. These (X, Y) - coordinates are from the State Plane Coordinate System. These data may be used to compute distances between vertices in South Western PA using the Pythagorean theorem. In Part 1, we will be using the (X, Y) pairs to compute the distance between each crime. To convert from feet to miles, simply multiply the computed distances by 0.00018939.

Your program will prompt the user for two dates. If the user enters 1/1/90 and 1/1/90 as input (the same date), your program will find an approximate TSP tour that visits the four crime locations of crimes that occurred on that date. There were four crimes on 1/1/90.

If the user enters 1/14/90 and 1/15/90 as input, your program will find an approximate TSP tour that visits the crime locations of those crimes that occurred between 1/14/90 and 1/15/90 inclusive. The root of the minimum spanning tree will always be the first index.

The output of your program will include a list of the crime records processed by your TSP approximation algorithm. It will provide a list of vertices showing a tour generated from the Approx-TSP-Tour algorithm shown above. It will also show the length of the tour. When you read the records into memory, you will assign an index of 0 to the first record that the user wishes to process. That is, the first crime that occurred on the first date of interest, will have an index of 0. For example, if the user enters 1/14/90 and 1/15/90 as input, your solution will be a list of vertices numbered from 0 and ending at 8.

An example execution appears on the next page. The user has entered only two dates (1/14/90 and 1/15/90). The program reads the crime file and selects the nine records for processing. It finds a cycle and displays the cycle and the cycle's length.

Enter start date
1/14/90
Enter end date
1/15/90

Crime records between 1/14/90 and 1/15/90
1340358.516,418063.7574,32887,1 ALPINE ST,AGGRAVATED ASSAULT,1/14/90,250300,40.45891236,-80.00757768
1351183.029,410482.0054,32888,211 BURROWS ST,AGGRAVATED ASSAULT,1/15/90,51000,40.43886004,-79.968006
1346775.118,410574.5466,32888,1600 FIFTH AV,AGGRAVATED ASSAULT,1/15/90,10300,40.43880904,-79.98384521
1346775.118,410574.5466,32888,1600 FIFTH AV,ROBBERY,1/15/90,10300,40.43880904,-79.98384521
1375199.387,415958.6475,32888,8100 FRANKSTOWN AV,AGGRAVATED ASSAULT,1/15/90,130600,40.45551239,-79.88222498
1342709.971,416295.7223,32888,609 E OHIO ST,ROBBERY,1/15/90,230400,40.45422538,-79.99896828
1341087.095,417642.7939,32888,1400 SANDUSKY ST,AGGRAVATED ASSAULT,1/15/90,220600,40.45780829,-80.00492164
1370004.246,420807.4845,32888,6628 BRAINARD ST,AGGRAVATED ASSAULT,1/15/90,120300,40.46847265,-79.90131253
1355089.881,420937.5121,32888,422 FORTY-FOURTH ST,ROBBERY,1/15/90,90200,40.46781983,-79.95491236

Hamiltonian Cycle (not necessarily optimum):
0 6 5 2 1 8 7 4 3 0
Length Of cycle: 16.354300334021968 miles

Note, two equally good programs may produce different results. It is the case that Prim will remove nodes from the heap in order of small to large. But the children of a particular node in Prim's tree may be ordered differently – producing different tours. These tours may be of different length but all will be less than twice the value of the optimal tour. Your answer may not match mine.

Part 2. Finding an optimal solution to TSP (30%)

One way to find an optimal tour is to simply list every possible tour and compute the length of each. Then, simply select the tour of minimum length. In Part 2, your task is to find an optimal tour using this brute force approach. Note that there are |V| ! permutations of the |V| vertices. Each of these is a different Hamiltonian cycle. But half of these are the same cycle travelled in a different direction. In this homework, we are unconcerned with the direction of travel. So, any minimal length cycle will do just fine.

The output of Part 2 will look like this:

Enter start date
1/1/90
Enter end date
1/1/90

Crime records between 1/1/90 and 1/1/90
1348656.471,399538.5342,32874,100 BONIFAY ST,ROBBERY,1/1/90,160600,40.40865518,-79.9760891
1359951.481,410726.1273,32874,320 SCHENLEY RD,ROBBERY,1/1/90,140100,40.44013011,-79.93653583
1357049.25,418429.9175,32874,4779 LIBERTY AV,ROBBERY,1/1/90,80900,40.46107271,-79.94764804
1361745.729,419343.2848,32874,5600 PENN AV,AGGRAVATED ASSAULT,1/1/90,111500,40.46389868,-79.93085611

Hamiltonian Cycle (not necessarily optimum):
0 1 2 3 0
Length Of cycle: 9.972129467002565 miles

Looking at every permutation to find the optimal solution
The best permutation
0 2 3 1 0

Optimal Cycle length = 9.499048743818797 miles

Part 3. Displaying the output to Google Earth (10%)

In Parts 1 and 2, we computed a tour of crime scenes in two ways. The first was a tour that may or may not have been optimal. The second was a minimum length tour. In Part 3, your task is to generate a KML file that contains both tours (the first tour will be shown in blue and the second will be shown in red). This KML file, when loaded into Google Earth, will show both tours. In order for both tours to be visible, you may add a very small bit of latitude and longitude to each vertex in one of the tours. In this way, the lines will not run exactly the same and one will not completely cover the other. Below is a screen shot of my solution over the vertices 1 through 5. The name of the output file will be PGHCrimes.kml. The user interaction is the same as above. That is, your program will prompt the user for two dates (which may be the same) and then the crime data will be processed and the KML will be generated.
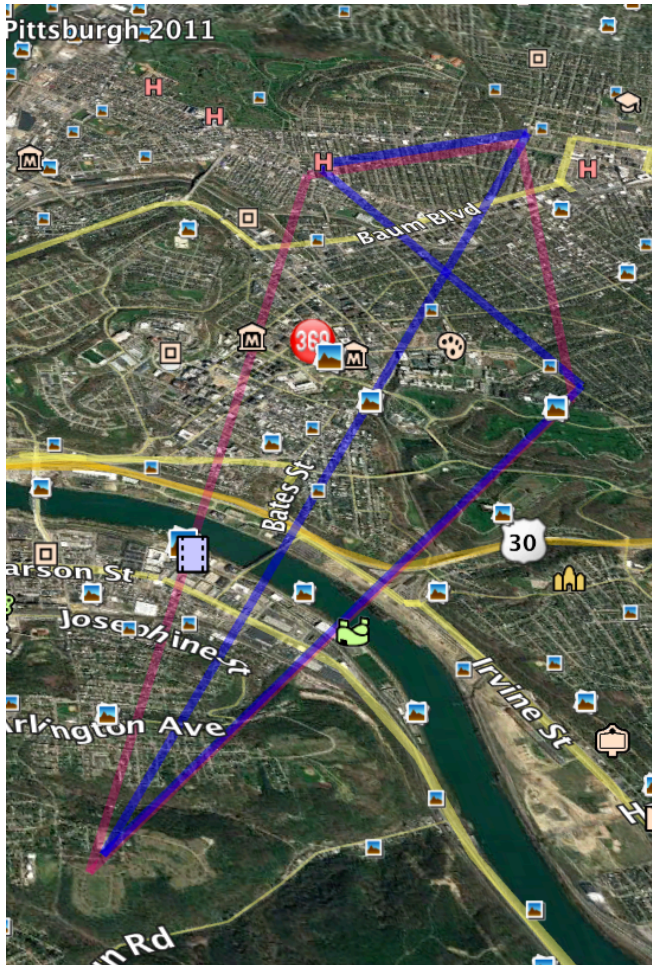
Here is the KML file (PGHCrimes.kml) that was generated by my solution to produce the image above:

```
<?xml version="1.0" encoding="UTF-8" ?>
<kml xmlns="http://earth.google.com/kml/2.2">
<Document>
<name>Pittsburgh TSP</name><description>TSP on Crime</description><Style id="style6">
<LineStyle>
<color>73FF0000</color>
<width>5</width>
</LineStyle>
```

```
</Style>
<Style id="style5">
<LineStyle>
<color>507800F0</color>
<width>5</width>
</LineStyle>
</Style>
<Placemark>
<name>TSP Path</name>
<description>TSP Path</description>
<styleUrl>#style6</styleUrl>
<LineString>
<tessellate>1</tessellate>
<coordinates>
-79.97508909999999,40.409655179999994,0.000000
-79.93553582999999,40.441130109999996,0.000000
-79.94664804,40.46207271,0.000000
-79.92985610999999,40.46489868,0.000000
-79.97508909999999,40.409655179999994,0.000000
</coordinates>
</LineString>
</Placemark>
<Placemark>
<name>Optimal Path</name>
<description>Optimal Path</description>
<styleUrl>#style5</styleUrl>
<LineString>
<tessellate>1</tessellate>
<coordinates>
-79.9760891,40.40865518,0.000000
-79.94764804,40.46107271,0.000000
-79.93085611,40.46389868,0.000000
-79.93653583,40.44013011,0.000000
-79.9760891,40.40865518,0.000000
</coordinates>
</LineString>
</Placemark>
</Document>
</kml>
```

FAQ for Assignment 4

1. Must I document my code? Yes, for full credit, documentation is required.
2. Must it be object oriented? Yes, for full credit, it must be object oriented.
3. Must I use appropriate variable names? Yes, for full credit, you must use appropriate variable names.
4. May I use Java's built in classes? You may not use any built in data structure related classes except for Java's Linked List class and Java's arrays. We have already worked with linked lists in previous assignments. Now, you are allowed to use the Java Linked List classes. The graph will be implemented as a one dimensional array of crime records along with a two-dimensional array of doubles – holding the computed distance between each pair.
5. How should I implement deleteMin() calls from Prim? You may write your own heap or you may simply select the smallest element from a distance array (taking care to mark this deleted value so it won't be selected again). Programs that implement a heap will score a bit higher than those that search a distance array. Those that simply search a distance array will earn a deduction of 4 points.
6. How do I perform a preorder traversal of the tree? Prim only provides parent pointers? Hint: an array of linked lists might be helpful. And be sure to review the slides on representing graphs.

7. Do I need to generate the exact same KML file that your program generated? No. Feel free to be creative. But, your Google Earth screen shots should clearly show the two tours.
8. What do I need to submit? You must submit all three parts in three separate projects. You must also submit three screen shots, one for each part. The last screen shot will be a screen scrape of Google Earth. Place the three projects and the three screenshots into a single directory and zip it with the name <your id>project4.zip.

Project 4 Auto Grader Submission Guidelines

1. Every time your code is run, it should create a *result.txt* file. The first line to be written to this file is the Andrew ID.
2. Your main method should keep asking the user for the next input. At each run, <u>you can print to console whatever you'd like</u>.
   However, the formatting for writing to <u>result.txt</u> needs to be according to below guidelines.
3. A snip of how your results.txt will look:

```
1    AndrewID
2
3    TestCase1
4    Hamiltonian Cycle
5    0 1 2 3 0
6    Length
7    9.972129467002565
8    Optimum path
9    0 2 3 1 0
10   Length
11   9.499048743818797
12
13   TestCase2
14   Hamiltonian Cycle
15   0 6 5 2 1 8 7 4 3 0
16   Length
17   16.354300334021968
18   Optimum path
19   ██ ! ██ ! █ █ █ █
20   Length
21   ██ ███ ████████
22
23   TestCase3
24   Hamiltonian Cycle
25   ██ ! ! █ █ █ █ █ ██ █ █ █ █
26   Length
27   ██ ██ ██ ██ ███
28   Optimum path
29   ██ ! █ ! █ █ ! █ ██ ██ █ █
30   Length
31   █ ██ ! ████ ████
```

4. The first line should be your Andrew ID. When you run your code, you will test it with 3 test cases. The rest of the lines should contain the output for these three test cases:
   a. Test Case 1 (open):
      1/1/90 to 1/1/90
   b. Test Case 2 (partially hidden):
      1/14/90 to 1/15/90
   c. Test Case 3 (hidden):
      1/2/90 to 1/2/90

5. Your result for the Hamiltonian Cycle may be slightly different and is acceptable.
6. Please ensure you strictly follow the formatting instructions and your *result.txt* looks exactly like the screenshot above.