Homework 6 Programming

Due Thursday, May 3rd at 11:59pm. There is *no* late homework on this assignment. It must be submitted by 11:59pm on May 3rd to be counted.

Please remember that to submit the assignment you will need to the Education drop down menu and select assignment complete. Also remember that you should submit a txt file on canvas that indicates your codio username so that we can cross check our grade data entry.

Traveling Salesman Problem - 70 points total

In this problem you will find solutions to the traveling salesman problem (TSP) and display the tours using the provided GUI. The graph implementation is identical to the one used for Programming problem 2 on Homework 5, except that vertices are identified by Integers instead.

Compile all Java sources and run the program. Display it this point, click on the "Virtual Desktop" button along the codio tool bar at the top of your screen. This should bring up a view of the computer's ac eared with your running program that displa

Clicking "Generate R arest Neighbor" or "Brute Force" will throw a null pointer exception on t to implement these methods in Graph.java.

Generating a Random Complete Graph - 7 points

In Graph.java, implement the method `generateRandomVertices(int n)`. This method should add vertices and edges to the Graph instance. After running the method, the instance should represent a complete graph with nvertices with the IDs/names 0,1,...,n-1 at random (x,y) coordinates. x and y can range between 0 and 100. Use java.util.Random to an external site. to create random numbers. If the method is implemented correctly, clicking on "Generate Random Graph" in the GUI will display the graph.

Nearest Neighbor - 28 points

Implement the method `nearestNeighborTsp()`, which should use the nearest neighbor algorithm to find an estimate of the shortest simple cycle that visits each vertex. The method should return a list of Edges on the cycle. Make sure to add the correct distance to each edge. If the method is implemented correctly, clicking on "Nearest Neighbor" in the GUI will display the nearest neighbor tour as an overlay. The GUI will also display the sum of edge costs, as well as the time required to compute the tour. This should not take longer than a few milliseconds for a graph of size 8.

Brute Force - 35 points

Brute Force - Implement the method `bruteForceTsp()`, which should compute the actual shortest simple cycle visiting each vertex. The method returns this cycle in the same format as `nearestNeighborTsp()`. You can run the method by clicking the "Brute Force" button, but it is not advisable to try this for graphs with n>8.

Hint: Because the graph is complete, the set of possible TSP cycles corresponds to all possible permutations of the vertices. In our graph representation, the vertices are identified by Integers. Try the following:

First, enumerate all possible permutations of the integers 0 to n-1. You should use additional methods to do this. For each permutation, measure the total cost of the corresponding cycle (sum of edge costs). Select the permutation with the lowest total cost.

Readme

In your `STUDENTREADME.md`, compare and comment on the actual time required by the Nearest Neighbor method and the Brute Force method for random graphs with graphs from 2 to 8 vertices. Run the algorithms on at least 3 different graphs for each n and average the runtime results.