Lab Assignment 4: Geomatics and the Travelling Sales[person] Problem

According to the ISO/TC 211, geomatics is the "discipline concerned with the collection, distribution, storage, analysis, processing, [and] presentation of geographic data or geographic information." Geomatics is associated with the travelling salesman problem (TSP), a fundamental computing problem about which there is an award-winning feature film. In this lab assignment, a University of Alberta student completes a Python program to analyze, process, and present entries, stored in a binary data file, of the TSPLIB, a database collected and distributed by the University of Heidelberg.

Version 0: Get Started

Unzip `V0GetStarted.zip` into your Working Directory. Double-click on the `tspTest_v0.txt` file, a text file, to open and review it in Spyder. Open the `tspAbout.txt` file externally, e.g., right-click on it and select `Open externally`. Close `tspAbout`. Double-click on the `tspAnalyze.py` file, a code file, to open it in Spyder. There is also a `tspData.mat` file, a *binary* file (non-text file) discussed below.

Review the `tspAnalyze.py` file in the Editor. The program consists of one long script that nests deeply twice: repetition within selection inside repetition; and selection within selection inside repetition. The code, related to menu selection, exhibits duplication. With an `if` statement, there is code for print and plot options ("`choice == 1`" and "`choice == 3`") but no code for a limit option ("`choice == 2`"). Make a copy of the `tspAnalyze.py` file. Call it `tspAnalyze_v0.py` and keep it for reference.

When a code file is opened in the Editor, Spyder looks for syntax errors and reports them in the margin. The given program has no syntax errors! Select Run `>>` Run to run it. When prompted, enter input as specified in `tspTest_v0`. With these test cases, there are no runtime errors! When the program ends, in the Variable Explorer pane, next to Plots, double-click on the `tsp` variable to explore its structure, a `list` of tuples. Double-click on an index to review one record, a `tuple`, in the table-like database. In the Console, enter `print(tsp[7])` to display a record. Enter `print(tsp[0])` for headings.

Click the three-bars button in the top-right corner of the Console and select `Undock`. Resize the Console to accommodate wide lines of text. Compare the Console contents to `tspTest_v0`, a diary of Console side effects for (any version of) the program when the Version 0 test cases are entered correctly.

Version 1: Refactor and Plot

Unzip the two files, `tspTest_v1.txt` and `tspPlot_v1.png`, in the `V1Refactor&Plot.zip` file into your Working Directory. When you complete Version 1 sufficiently and test it as specified, the iPython Console side effects of your `tspAnalyze` program should match `tspTest_v1`. Once you finish, there should also be an output file, `tspPlot.png`, that resembles `tspPlot_v1.png`, when you open and compare both images externally. Compare the `tspTest_v0` and `tspTest_v1` files carefully.

Convert the `tspAnalyze` program from a script to a modular form having five functions, `main`, `menu`, `tspPrint`, `tspPlot`, and `plotEuc2D`. Invoke the `main` function, having no arguments, at the end. This conversion, called refactoring, will not impact functional attributes of the program (notwithstanding variables available to explore after the program ends), as evidenced partly by unchanged side effects for Version 0 test cases. Refactor and test, as follows, before completing and testing the `plot` option.

Start by defining the `main` function to have all code below the `import` statements. Fix syntax errors, like bad indentation, and runtime errors. Failing to invoke `main` is a logical error. Copy the menu-selection code, a sequence of `print` statements followed by a related `while` loop, into a `menu` function, having no input arguments but returning one output argument, `choice`. Replace all instances of the menu-selection code in the `main` function with invocations of the new `menu` function. Test for errors.

Copy the "`choice == 1`" action from the `main` function into a `tspPrint` function, which returns no output argument but requires one input argument, `tsp`. Replace the "`choice == 1`" action in the `main` function with an appropriate `tspPrint` invocation. Similarly, move the "`choice == 3`" action from the `main` function into a `tspPlot` function, invoking the latter in the former. Test for errors. Check the program, once it satisfies all the Version 0 test cases, with the Version 1 test cases.

When side effects are incorrect, there are logical errors. Complete the `tspPlot` function so that a plot is created, using `matplotlib.pyplot` (`import` it), according to the story of a given test case. Create a variable `tsp1` in `tspPlot`, before the `if` statement, equal to `tsp[num]`. In the `edge = 'EUC_2D'` action, replace the `print` statement with `plotEuc2D(tsp1[10], tsp1[2], tsp1[0])`. Define a function, called `plotEuc2D`, with three input arguments, called `coord`, `comment`, and `name`.

For an entry of the `tsp` database that has `'EUC_2D'` node coordinates, the `tspPlot` function invokes the `plotEuc2D` function to make and show a plot, including annotations. Plot the second column (y-axis) of the `coord` argument, a NumPy array, vs. the first column (x-axis). Use solid lines (blue) with dot markers. After extracting into lists coordinates of the last and first points, plot a red line to link them. Include `plt.savefig('tspPlot.png')` to output the plot as an image file, `tspPlot.png`.

Submit your Version 1 solution by the Version 1 deadline. Submit `tspAnalyze` only, after completing the initial comment header. Before submission, test the code in a folder where all other relevant files are as given. To match `tspTest_v1` exactly, review the text printed in the Console and refine your code, e.g., `plotEuc2D`. Consistent with Version 1 requirements, other test cases are possible.

Version 2: Limit Dimension

Unzip the files, `tspTest_v2.txt` and `tspPlot_v2.png`, in the `V2LimitDimension.zip` file into your Working Directory. When you complete Version 2 sufficiently and test it as specified, the Console side effects (text input/output) of your `tspAnalyze` program will match `tspTest_v2`. There will also be an output file, `tspPlot.png`, that will resemble `tspPlot_v2.png`, when you open and compare both images externally. Given test cases are a very small subset of the possible test cases.

Review your Version 1 program and its requirements. Write suitable comment headers, in your own words, for the non-`main` functions, i.e., `menu`, `tspPrint`, `tspPlot`, and `plotEuc2D`. Each comment header must summarize the function's purpose, any input (formal parameter) arguments, any output (returned) arguments, and side effects (such as Console input and output, as well as plots).

To complete the limit option of the program, define a function, `tspLimit`, with one input/output argument, `tsp`, a list passed by reference. In the `main` function, invoke `tspLimit` appropriately in the action of a "`choice == 2`", an `elif` block added to the `if` statement. When invoked, a completed `tspLimit` may modify the `main` function's database copy, `tsp`, leaving it with fewer records.

In the `tspLimit` function, compute the minimum and maximum dimension (number of cities) of all records in the `tsp` database. Second, print the min and max dimension to the Console. Third, prompt the user to input a limit value, following the `tspTest_v2` storyboard. Finally, delete records with a `dimension` field more than the limit. Therefore, `tspLimit` may modify the `tsp` database.

The `menu` function uses a `while` loop to error-check user input. An integer less than a valid minimum or greater than a valid maximum prompts the user for the input again. Modify `tspLimit` and `tspPlot` to error-check user input the same way. An integer less than the minimum or greater than the maximum dimension is an invalid limit value. As the table header, `tsp[0]` is not a valid record for plotting.

Consider refactoring a completed `tspLimit` function. Move code that computes the minimum and maximum dimension into a new function, `tspMinMax`, having one input argument, `tsp`, and returning two output arguments, `minVal` and `maxVal`. You need not do this if you believe you can successfully appeal a decision by a teaching assistant who decides your code is less readable otherwise. 😉

Write suitable comment headers, in your own words, for the `tspLimit` function and any other new function, like `tspMinMax`. Review the initial comment header of the program, especially reported percentages. Submit your Version 2 solution by the Version 2 deadline. Submit `tspAnalyze` only. Before submission, test the code in a folder where all other relevant files are as given.

Revision History

This document and related files were created using MATLAB, in 2021, and edited using Python, in 2022, by Dileepan Joseph, with contributions from Edward Tiong and Wing Hoy. They were edited again, in 2024, by Joseph with contributions from Antonio Andara Lara. Two ENCMP 100 Programming Contest entries, Distance Matters and Open Street Maps Path Generator, motivated the lab assignment.