IS590PR. Fall2017. Assignment 6 – Network Graphs & OO design

Work in teams of 4 or 5 students to design and then implement an object-oriented solution to the following scenario. There's enough work here to do that you'll need to cooperate, plan, and divide up the effort to combine into a complete and working whole. This is a great chance for everyone to develop and learn from each other. Notice there's some planning, mapping and encoding work that's not programming. If you procrastinate, your team is not likely to succeed.

Imagine we're developing a custom navigation app that will help guide a self-driving van between various campus buildings to automate campus mail delivery. Your part of the system is only concerned with determining a "best" route for the van to take, given the knowledge your program has about (some of) the campus buildings and the roads around them. At this point we're designing and testing a prototype, so it thankfully doesn't need the entire campus map and roads yet.

General Requirements:

- You are not allowed to be in the same team as anyone else who was in your group for Assignment 2. I have assigned you to a team, and in class we'll discuss getting started.
- A brief, written report must accompany your program solution. In it, you must describe how you split up the work and which team members contributed significantly to which parts and which team roles they had.
- You must design at least one new OO class in your program, with some methods designed for clarity, testability, and reusability.
- You should avoid accessing any module-level variables globally, though you may access instance variables or class variables from inside the class methods without passing them as parameters. But use true parameters and return values as much as you can for better function design.
- Make use of the Python NetworkX library. You can either subclass one of the NetworkX classes or just use it as a data structure within your class. DiGraph is most likely what you'll need.
- Do not overlook the code quality factors we've been discussing so far, including clear structure and symbol names, documentation, and even some unit testing (good Doctests can be sufficient).

Scenario Requirements:

- Pick a section of campus just a few blocks in size so this is manageable. The building list & maps at http://illinois.edu/map/view should help you constructing your graph.
- The program needs to know about at least 8 differently named or numbered streets around campus. Note that you'll have to code far more road segments (edges) than that into the graph. Road segments will benefit from a roughly approximated distance stored as the "distance" attribute of the edge. That allows using weighted shortest path algorithms.

- ☐ The program needs to know about at least 20 different major campus buildings, which you'll store as graph nodes. For each building you include, it should know the name, the official mailing address, a unique campus mail code, and a point of road access.
    - o For example, the iSchool or "School of Information Sciences" building is located at 501 E. Daniel Street, Champaign. Its campus mail code is MC-493. It technically has paved driving access to three streets (Daniel, S. 5th, and S. 6th), but you may simplify by choosing a single spot on your network where a building's primary road access starts.
    - o There are some buildings with multiple Mail Codes. You may simplify by just choosing one of the codes located there. All the Mail Codes are listed here https://union.illinois.edu/shop/campus-mail/campus-mail-codes
- ☐ Among the buildings, you must include either the Foreign Languages Building or Lincoln Hall. This is to make sure your program and graph support *one-way streets* properly.
- ☐ Use network nodes to represent intersections joining the streets you've selected and spots where the building driveways connect to streets.
- ☐ When the program runs, it must:
    - o Alphabetically list all the buildings it knows of (showing mail codes and names).
    - o Ask the user to enter any starting and ending mail codes from that list.
    - o Then it will calculate a *shortest path* and print simple turn-by-turn navigation instructions. (See example below)
    - o Then it should allow the user to quit or enter another navigation query.
- ☐ The shortest path calculations must be derived from the graph, not pre-determined. For example, if a construction project closes a street segment, we should be able to temporarily either remove that edge or temporarily change its "distance" to a very high number so that routes normally through it will get re-routed. The algorithm(s) you need are already part of NetworkX.

Example. A portion of the program session could look like this:

```
Enter starting and ending mail codes: 493 454

Travel from School of Information Science to Lincoln Hall:
     Starting on Daniel Street, turn East
     At S. 6th Street, turn South
     At E. Armory Avenue, turn East
     At S. Wright Street, turn North
     Proceed until you arrive at Lincoln Hall
```

NON-Requirements: You are not expected to generate any visual maps or graphics. Do not interface with any external mapping databases, GIS tools, or other navigation-related systems. Although that would also be cool and useful, the purpose here is to practice team OO class design and to use a flexible network graph data structure to construct and solve simple path problems. It is not necessary to encode GPS coordinates into the nodes, though you may.