SFWRENG 3XB3

L1 – Graph Lab

McMaster University
ENGINEERING
Computing & Software

GENERAL INFORMATION

- Author: Dr. Sébastien Mosser mossers@mcmaster.ca

- Start date: 10/09/2022
- Delivery date: 02/10/2022
- Weigh in final grade: 20%

LEARNING OBJECTIVES

The first week is guided in a step-by-step way. Then, you'll be more autonomous. At the end of this three weeks assignment, students should:

- **Know and understand:**
  - How graph data structures can be implemented
  - How graphs are manipulated to support engineering problems
- **Be able to:**
  - Implement a non-trivial graph data structure from scratch
  - Implement graph algorithms on top of this data structure
  - Design and implement benchmarks to compare algorithms *w.r.t.* different KPIs

PREAMBLE

You are not allowed to modify the Pipenv file (except for the python version, among 3.8, 3.9 and 3.10). The only libraries that you can uses are the one available in python core library, and matplotlib for graphing your results. Typically no NetworkX or Panda

PROJECT PREPARATION

1. Find a teammate in your lab section.
   a. If your lab section contains an odd number of students, then one (and only one) group can be composed of three students. It'll require manual action on Avenue, please check the situation with your TA.
   b. We keep you separated "by lab section" so that we can ensure that you'll not be graded by your TA. You can consider that your TA is only here to help and support you during the lab and will never be in a position of grading your work. Don't hide anything!
2. One member has to fork the "L1 – Graph Lab" project on the GitLab interface:
   a. URL: https://gitlab.cas.mcmaster.ca/sfwreng-3xb3-fall-22/l1-graph-lab
   b. Make sure your project is "private"
   c. Add your teammate as "maintainer" of the project
   d. Add your TA and the instructor as a "reporter"
3. **Declare your group on Avenue by the end of your first lab**
   a. Check that you have access to the submission interface
4. Clone the project on your local computer

5. You are ready to start this lab!
    a. If you have questions outside of labs, shoot them in your Lab section channel on Teams. Your TA and the instructor are monitoring these channels to support you.

WEEK #1: MANIPULATING

This week's objective is to be familiar with the dataset (it represents a graph as a set of CSVs files), as well as to start producing some business value on top of this graph by computing itinerary.

LOADING AND ANALYZING THE GRAPH

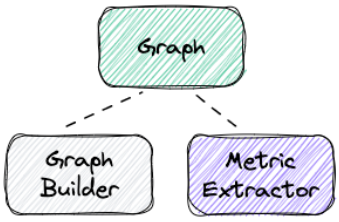

Figure 1. Initial module

1. **Analyse the files** available in the dataset, to identify where the nodes and edges are stored.
    a. Which kind of graph is represented by this file?
2. **Implement an object-oriented Python library** to encapsulate the relevant graph data structure, according to the following requirements:
    a. The CSV file format might change in the future, so the graph class must be totally independent of the CSV file, so you should consider separating responsibilities between "being a graph" and "building a graph".
3. **Quantify the graph** you are working with:
    a. How many nodes? How many edges? What is the average degree of the nodes?
    b. We might have to add other "metrics" in the future, so considering carefully how the metrics extraction should be designed to support the inclusion of metrics in an "open/closed"[1] way.
4. **Start your report** by opening the `graph_lab.ipynb` file with Jupyter (see warmup labs for technical details)
    a. Edit your names in the header, as well as the URL to clone your repository
    b. Use your notebook to:
        i. Load the graph form the CSV files using your library
        ii. Compute the metrics
        iii. Draw the distribution of node's degree (x-axis is the degree, and y-axis is the number of nodes with such degree).

SHORTEST PATHS

For example, to go from *Piccadilly Circus* to *St. Paul's*, one has three options in five stops.

A. You can take the blue line for three stops and connect to the red one at Holborn, then reaching St. Paul,s in two stops.
B. Or you can take the blue line for one stop to Leicester Square, then the black one for one stop to Tottenham Court Road, and then connect to the red one for three stops.
C. Or going to Oxford Circus and then straight to St Paul's

Figure 2. Subset of London's subway network

[1] https://en.wikipedia.org/wiki/Open%E2%80%93closed_principle

1. **Design the itinerary structure**:
    a. Define a class to model an Itinerary between two stations.
    b. How to rank paths of same lengths?
        i. Consider the number of connections (the less the better), as well as the travel time. One might prefer a slightly longer travel if it avoids changing seats three times.
2. **Provide an implementation of Dijkstra's algorithm** to find the shortest path between two stations.
3. **Provide an implementation of the A* algorithm**.
    a. Both algorithms use a priority queue… Think modular and DRY[2]
    b. A* relies on heuristics to accelerate the computation.
        i. An easy heuristic is to compute the physical distance between two nodes (we know the latitude and longitude of each station), assuming that if two stations are close to each other, they'll also be close in the subway network.
        ii. But you can imagine other heuristics, fitting your definition of "shortest"
4. **Write a benchmark** to compare your itinerary finding implementations
    a. What are the right KPIs to measure?
    b. What are the right cases (itineraries) to consider for this benchmark?
    c. Implement the benchmark, and measure both execution time and KPIs
5. **Update your report**
    a. Provide a class diagram of your implementation and insert it into the notebook.
    b. Justify your design choices
    c. Present and analyse the results of your benchmark.
    d. Describe how you've split the work and who did what.
6. **Place a tag** labelled `week1` in your git repository and push it.

WEEK #2: PLANNING

This week, we'll polish your work from last week, and add more business value to the project.

1. If you haven't started **testing your code**, it's time to start!
    a. Identify relevant test cases, and provide a test suite associated to your library

SUBWAY PATROL PLANNING

Subway officers are patrolling the network, to ensure the security of travellers and to provide help to passengers when needed. The patrol manager identifies a subset of nodes that are worth patrolling (e.g., soccer game scheduled, concert, touristic area).

1. Your job is to add into your library a way to compute the path the patrol officer must follow to cover all these stations in an efficient way.
    a. *Hint: This is a variation of the "Traveler Salesman's Problem"*

URBANISM PLANNING

London's network is organized in different zone. We are interested into understanding how these zones are interconnected to each other's, as well as identifying "transportation islands".

[2] Don't Repeat Yourself

Consider the network depicted in in Fig 3. Zone 1 contains stations {1,2,3,4}, and zone 2 contains stations {5,6,7,8,9}:

- Zone 2 contains two isolated transportation islands: {5,6} and {7,8,9}.
- To travel from one island to another one, it is necessary to go through another zone: even if station 6 and 7 are geographically close to each other, travelling from 6 to 7 requires crossing stations {5, 1, 4} or {5, 1, 2, 3}.
- The connection {5, 1, 4, 3} also exists, but is subsumed by {5, 1, 4}, and thus does not count.

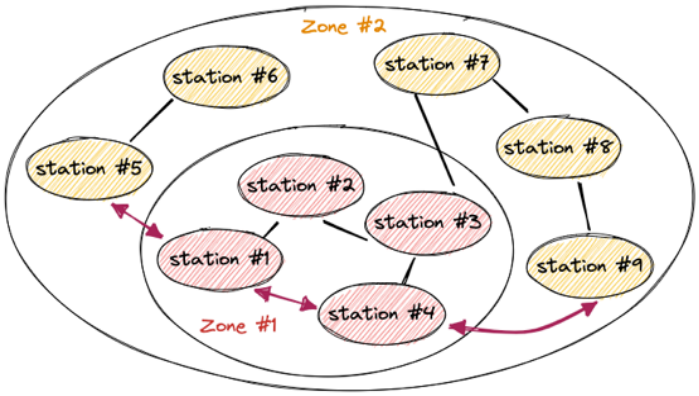

Figure 3. Example network

1. Your job is to add into your library how to identify these islands for each zone of the network, as well as how they are connected to each other's.
    a. *Hint: the keyword you're looking for is "connected components". You're welcome.*

REPORTING & ANALIZING

1. **Extend your report**:
    a. Provide an updated version of your class diagram, making explicit how you've modified your design to introduce the new features. Justify these changes and how they are compatible with the SOLID[3] principles.
    b. Provide executable example of the two previous features in your code.
    c. Analyse the performance of your algorithms, from both theoretical and empirical points of views.
2. **Place a tag** labelled `week2` in your git repository and push it.

WEEK #3: GENERATING

Last week, final leg of this three-weeks sprint! We will now polish what was done and introduce a last feature.

1. Re-read your report and code with a calm head. It'll give you another perspective on your work after two weeks. Polish your test cases, benchmark code, and visualization.

[3] https://en.wikipedia.org/wiki/SOLID

GENERATING RANDOM GRAPHS

One of the main problems of our work to date is that we are dependent on the London subway case. If we want to test our work, it is always on London's data, meaning that our tests and analyses can be biased by London's peculiarities.

1. Enhance your library to provide the concept of "Graph Generator"
2. Provide a naïve generator which, given a number of node and a degree value, create such a graph randomly, with all stations are uniformly connected to "degree"s other's.
    a. Leverage the names of the existing stations to create "reasonable" names in your generated networks.
3. Enhance your generator to support other form of distribution for the node's degrees (e.g., normal distribution instead of uniform).

GENERATING REALISTIC TRANSPORTATION GRAPHS

Our previous graphs are too random to be realistic. A transportation network is organized as zones, and each zone as some particularities.

- Identify which characteristics are important in a transportation graph to guide its generation
    - *Hint: you can leverage your work on the paths to take into account metrics such as centrality for example, as well as graph diameter.*
- Propose a new generator, that take these characteristics into account
- Experiment with your generator to validate its time complexity, as well as the realism of the generated graphs

REPORTING & ANALIZING

1. **Extend your report**:
    a. Provide an updated version of your class diagram, making explicit how you've modified your design to introduce the new features. Justify these changes and how they are compatible with the SOLID principles.
    b. Provide executable example of the two previous features in your code.
    c. Analyse the performance of your algorithms, from both theoretical and empirical points of views.
2. **Place a tag** labelled `week3` in your git repository and push it.

FINALIZING

SELF REFLECTION

Self-reflection is an essential part of *experiential learning*. It is the occasion to take a step back on your work and look at it according to four directions: *(i)* backward, *(ii)* inward, *(iii)* outward and *(iv)* forward. Each member of the team must answer to the following questions in the final report:

Dimension	Question
Backward	Have you done a similar kind of work in the past?
Inward	How do you feel about this piece of work? What parts do you particularly like? Dislike? Why?
Outward	What is the one thing you particularly want people to notice about your work?
Forward	What would you change if you had the chance to do this project over again?

DELIVERING THE ASSIGNMENT

- Ensure that your tests can be executed using the following command:

  - `pipenv run python -m coverage run -m pytest -v test_*.py`
- Ensure that your coverage report can be executed using the following command:
  - `pipenv run python -m coverage report`
- Check what flake8 reports about your code
  - `pipenv run python -m flake8 *.py`
- Check that it is possible to open your notebook with the following command:
  - `pipenv run python -m jupyter lab graph_lab.ipynb`
- Print/Export your Jupyter notebook (after execution) as a PDF file.
- Submit this file to Avenue
  - You can submit how many time as you want, only the last one is kept.
- An automatic script will collect your repository on the deadline.
  - **If you deliver after the deadline, it's too late.**

EVALUATION CRITERIA (MAX SCORE: 100)

It is your responsibility to ensure that your code and report are delivered in a way conform to this document. In particular, your assignment will not be graded (i.e., will obtain the grade 0/100) if:

- It is delivered too late;
- Your Gitlab repository is not private;
- The TAs or instructor does not have access to the repository;
- Your report not delivered as a PDF on Avenue;
- Your code (tests and notebook) does not execute.

This list is not exhaustive, and the instructor and grading teams reserves the right to refuse deliveries that would not meet the excellence criteria expected from McMaster's students.

The following grading scheme will be used to evaluate your work.

Dimension	Criteria		#Points
Syntactical (/15)	Weekly tags available		5
	Self-reflection questions answered		5
	Presence of tests for the main features		5
Soft. Quality (/25)	Code design & Respect of SOLID principle		15
	Readability & Maintainability		10
Features (/30)	Week #1	Graph builder	2.5
		Metrics extractor	2.5
		Path Finder	5
	Week #2	Subway Patrol	5
		Islands	5
	Week #3	Random Graph Generator	5
		Domain Graph Generator	5
Report (/30)	Quality of benchmarks		10
	Quality of analysis		20

It is important to note that delivering according to the requirements (syntactical plus features dimensions in the table) only brings you 45% at max.
The remaining 55%are based on your capacity to engineer code and analyse it.

END OF ASSIGNMENT.