Homework 6: Graph Implementation

COS 226 – Spring 2024

Assigned: 01 Apr 2024 Due: 22 April 2024

This *final* assignment will give you practice implementing graphs in Python.

Graph implementations

1. Create a generic `Graph` class that has all the methods described for the graph ADT in your book. This is an abstract class, meaning that it is not meant to be instantiated itself, but rather will serve as a base class for the other implementations in the other parts of this assignment. That means that most of the methods will exist in a skeletal form that will be actually implemented differently in the other classes. For example, one of the methods that will differ for each implementation is `vertices`, a method to return all the vertices in the graph. You might implement this in the `Graph` abstract class like this:

   ```
   1  class Graph:
   2      # other methods, etc.
   3      def vertices(self):
   4          print(f'Calling dummy method "vertices()" of Graph instance {self}.')
   5          return None
   6      # other methods...
   ```

   Note that you will likely find it useful to create other classes in support of `Graph`, such as ones for vertices and edges. These can be internal to `Graph`, as we did for trees, or defined separately, as you see fit.

   Note: Given the lateness in the semester, *you do* ***not*** *have to provide a* `position` *class* for graphs!

2. Implement an abstract class `Digraph` for directed graphs. You can base this on `Graph`; you'll likely want to have a different edge class for these graphs in order to record origin and destination of edges.

3. Implement subclasses of `Graph` and `Digraph` implemented as edge lists. You can either implement separate classes, such as `GraphEdge` that inherits from `Graph` and `DigraphEdge` that inherits from `Digraph`, or you can create a base class `GraphEdge`, with a subclass `DigraphEdge` of *that*—possibly also inheriting from `Digraph`. As part of the comments for the code (or as a Markdown cell in a Jupyter notebook, if that's what you're using), explain your choice.

   Regardless, these classes will need to actually implement all the methods in the graph ADT in the book (that is, unless a parent class already implements them correctly).

   Provide output to show examples of each of the ADT methods working using these classes.

4. Do the same thing as in question 3, but using an adjacency list to implement the graphs.

5. Do the same thing as in question 3, but using an adjacency matrix to implement the graphs.

Graph algorithms:

6. Implement a depth-first traversal method, `DFT`, of your `Graph` class. Provide output to show that it works on all of the subclasses/implementations you have created.

7. Implement a transitive closure algorithm for *one* of your graph implementations. Provide output to show that it works.

8. Implement Dijkstra's shortest-path algorithm for *one* of your graph implementations. Provide output to show that it works.

9. Implement a minimum spanning tree algorithm for *one* of your graph implementations; I don't care which MST algorithm you choose. Provide output to show that it works.

Turn in:

Turn in a zip file named `lastname.zip`, where `lastname` is your last name, containing:

- Your code, as one or more Python files. One of your files should be named `lastname.py`, where `lastname` is your last name. This should be the file that, if I choose to run your code, I can import and it will run to show off your program.
  - Your code **must** be well-commented and well-structured! See the assignment rubric in the assignment on Brightspace to see what is expected.
- Output file(s) showing that your code works.
- Any data files used to produce the output.