Homework 1

Due 11:59 PM on Wednesday March 3rd, 2021

In this assignment, you will implement Dijkstra's Shortest-Path Algorithm.
Key instructions:

- 1. Please indicate your name at the top of each assignment(in comments for each piece of code you submit) and cite any references you used (including articles, books, code, websites, and personal communications). If you're not sure whether to cite a source, err on the side of caution and cite it. Remember not to plagiarize: all solutions must be written by yourself.
- 2. Please use **Python 3** for this assignment, the grading of this assignment will be done in a Python 3.7.5 environment.
- 4. Please do NOT use any additional imports, only write your code where you see `TODO: YOUR CODE HERE`, and change your return value accordingly.
- 5. Please modify and submit the following files:
  - `graph_adjacency_list.py`
  - `graph_edge_list.py`
  - `shortest_path.py`
- 6. Reminder on Late Policy: Each student has a total of one slip day that may be used without penalty for homework.

Graph Shortest Path

Data Structure

We want to implement Dijkstra's Algorithm as presented in lecture to find the shortest path in a graph. Before doing so, you'll first need to implement a directed graph in two different ways:

- An **adjacency list**, where a graph is represented as a map, where the keys are nodes in the graph and the values are a list of all the nodes adjacent to the key node.
- An **edge list**, where a graph is represented as a list of tuples, where each tuple represents an edge between two nodes.

NOTE: these graph representations are intentionally different than the graph representation that was presented in lecture. The goal here is to explore alternate graph implementations, as well as show how a consistent API will allow the same shortest path algorithm to work despite different underlying graph representations.

NOTE: for both these graphs, implementations for both the constructor and a `has_edge` method are given. **Please DO NOT edit those given implementations**; they are there to help you understand the data structure we intend for you to implement.

NOTE: The two graphs should be ***directed*** graphs

Todos

- implement **add_edge** in both `graph_adjacency_list.py` and `graph_edge_list.py`
- implement **get_neighbors** in both `graph_adjacency_list.py` and `graph_edge_list.py`

Dijkstra's Shortest-Path Algorithm

Next, you'll implement Dijkstra's Shortest-Path Algorithm, which should work regardless of which of the two types of graphs is given as input. The `shortest_path` function asks you to return a tuple that looks like `(['start_node', ..., 'target_node'], 'length')` where the first part is the shortest path from the `'source_node'` to the `'target_node'` and the second part is the `'length'` of said path. We recommended you to implement this in two stages:

- First, implement `shortest_path` while only worrying about the length of the path. For this stage, just return something like `([], 'length')` so that the output passes our automated tester's type checks but allows you to focus on making the `'length'` right.
- Second, augment your implementation to track the nodes in the shortest path so that you can output the path itself along with its length.

Each element of the output will be graded separately, so giving an output of `([], 'length')` on a given input will receive partial credit (assuming 'length' is correct for the given input).

NOTE: As mentioned in class, the most efficient implementation of Dijkstra's Algorithm utilizes a priority queue to manage the data. HOWEVER, **use of a priority queue is not a requirement for this assignment**. We will accept any reasonable runtime (i.e. we will test the algorithm on some large graphs to make sure the algorithm completes in a reasonable time) and will not be picky about implementation details if the algorithm returns the correct output.

NOTE: You can assume the input graph is connected, that all the graph's edges have positive edge weights, that the `source_node` and `target_node` are both nodes in the graph, and that at least one path from the `source_node` to the `target_node` exists.

Todos

- implement **shortest_path** in `shortest_path.py`

Logistics

To ensure compatibility with our grading software, please ensure that the provided test files run. You should be able to at least run the following without errors:

- `python graph_test.py`
- `python shortest_path_test.py`

NOTE: `shortest_path_test` relies on your graphs working, so ensure that `graph_test` passes before moving on to `shortest_path_test`.

Testing

The provided tests are **non-exhaustive** meaning *passing these tests alone does not necessarily guarantee a perfect score*. In addition to the provided tests, we encourage you to write additional ones. The grading of this assignment will be done in a Python 3.7.5 environment.