Homework 2

Due 11:59 PM on Monday November 4, 2019

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

- 1. Collaboration: You are **required** to work in groups of 2 students on each assignment. Please indicate the name of your collaborator 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. You may submit just one copy of the solution for the group. Remember not to plagiarize: all solutions must be written by members of the group.
- 2. Partner-finding: You have one week to find your preferred partner yourself and form your own group on CMS. If you have not formed a group on CMS by the **partner-finding deadline on 10.28th 11:59PM**, we will run a partner-matching script to assign groups for you.
- 3. Please only use **python2** for the assignment (allowing multiple versions makes grading substantially harder for a huge class like 5112)
- 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. We will also drop your lowest homework score. An assignment can be at most one day late without penalty via slip days.

1 Graph Shortest Path

1.1 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: that 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`

1.2 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: 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`

2 Logistics

To ensure compatibility with our grading software, please ensure that the provided test files run. You should be able to 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`.
Just like HW1, these tests are **non-exhaustive** meaning *passing these tests alone does not necessarily guarantee a perfect score.*

3 Testing

In addition to the provided tests, we encourage you to write additional ones because in real-life software development, you will eventually have to learn to write tests like this yourself and I recommend reading up on Richard's slides from last year if you want to know more.