CE 272
Traffic Network Equilibrium

Programming Task 2
Due February 20 (5 points)

General Instructions: This task is the first of four tasks in the computer programming project and involves data processing of transportation networks. Each task in this project builds up on the previous task. Hence, try to have a fully working version before proceeding to the next task. Label your variables meaningfully and provide comments. You can make modifications and optimize your code before you submit your final project. Share a zipped version of .py files with the input files or a link to your Colab on Teams.

Task Objectives:

1. To the `node class` that you created in the last task, add three new members to store shortest path related information: a distance label, a predecessor node, a predecessor arc.

2. For the `SiouxFalls` network, using the free-flow-time values in the `arc class`, write a `label correcting` and a `label setting` function that takes an origin node as the argument and computes the shortest path labels and predecessors for all the other nodes in the network.

3. Compare your results using NetworkX's shortest paths functions. Report the average run times of your code and NetworkX's functions. Estimate the average by varying the origins from 1 to 24, and divide the total run-time by 24.

4. In the `SiouxFalls` network, the zone centroids are actual nodes, i.e., junctions/intersections but for most other networks, centroids are represented using artificial nodes. The field `FIRST THRU NODE` in the meta data can used to determine the set of real and artificial nodes. For instance, in the `Anaheim` network, the `NUMBER OF ZONES` is 38 and `FIRST THRU NODE` is 39. This indicates that nodes 1 to 38 are artificial zone centroids and nodes 39 to 416 are actual junctions. Store this meta data information in your code.

5. The shortest path between an OD pair might pass through other zone centroids, which is an issue if those centroids are not actual nodes. Modify your `label correcting` and `label setting` functions such that the shortest path between any OD pair only passes through nodes greater than or equal to the `FIRST THRU NODE`.

6. Also test your code on `Eastern-Massachusetts`, and `Chicago-Sketch`, `Anaheim` networks using the free-flow-times for each network.