CE 272
Traffic Network Equilibrium

Programming Task 1
Due February 3 (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.

Data for the project: The GitHub has a repository of several real-world networks which will be used for the project. Each network has a `networkname_net.tntp` file. The first few lines of this file have some metadata indicating the number of nodes, arcs etc.

```
<NUMBER OF ZONES> 24
<NUMBER OF NODES> 24
<FIRST THRU NODE> 1
<NUMBER OF LINKS> 76
<END OF METADATA>
```

The metadata is followed by the following link attributes (names may sometimes vary across networks).

1. Tail node
2. Head node
3. Capacity
4. Length
5. Free flow time
6. B
7. Power
8. Speed Limit
9. Toll
10. Type

```
~ Init_node Term_node Capacity Length Free_Flow_Time B Power Speed_limit Toll Type;
 1 2 25900.20064 6 6 0.15 4 0 0 1;
 1 3 23403.47319 4 4 0.15 4 0 0 1;
 2 1 25900.20064 6 6 0.15 4 0 0 1;
 2 6 4958.180928 5 5 0.15 4 0 0 1;
 3 1 23403.47319 4 4 0.15 4 0 0 1; ...
```

Task Objectives:

1. Select the `SiouxFalls` network. This network is a simplified version of a city in South Dakota, US and is popular test network in transportation literature (see Figure 1).
2. Do not manipulate the input files. Your code must be designed to handle the data in the current form and should work for the other networks as well.
3. Create a `class` in your code for nodes and for arcs. The `members` of the `node class` should include the node number and the adjacency/downstream list of nodes and arcs (store these as vectors of variable sizes). The `arc class` must contain all of the above attributes.[1]

[1]Defining classes will make it easy for you to implement the algorithms from the course. If you haven't used classes before here is some background material: https://docs.python.org/3/tutorial/classes.html

4. Within the `arc class` use the `int` data type for storing the `Tail node`, `Head node`, and `Type`. For all the remaining attributes use a `double` data type. For the `node class` use the `int` data type for the node number and adjacency lists.

5. Read and store the number of nodes and links from the metadata.

6. Create a class instance for each arc and write a function that reads the rows in the above file and stores the data in the appropriate fields.

7. Create a class instance for each node and using the arc data populate the node number, adjacent node list, and adjacent arc list.

8. Write functions to print the following items.

   (a) Node-node adjacency matrix

   (b) Node-arc incidence matrix

   (c) Adjacency list of the nodes

9. Test your code for `Eastern-Massachusetts`, `Chicago-Sketch`, `Anaheim`, and `Winnipeg`. Do not hard-code anything except the network name. You should be able to change the network name and the code must provide the required outputs.

10. Create graph objects for these networks using NetworkX, which is a Python package for network analysis. Print the node-node adjacency matrix and adjacency list using this package. Verify your answers from your code with this output.

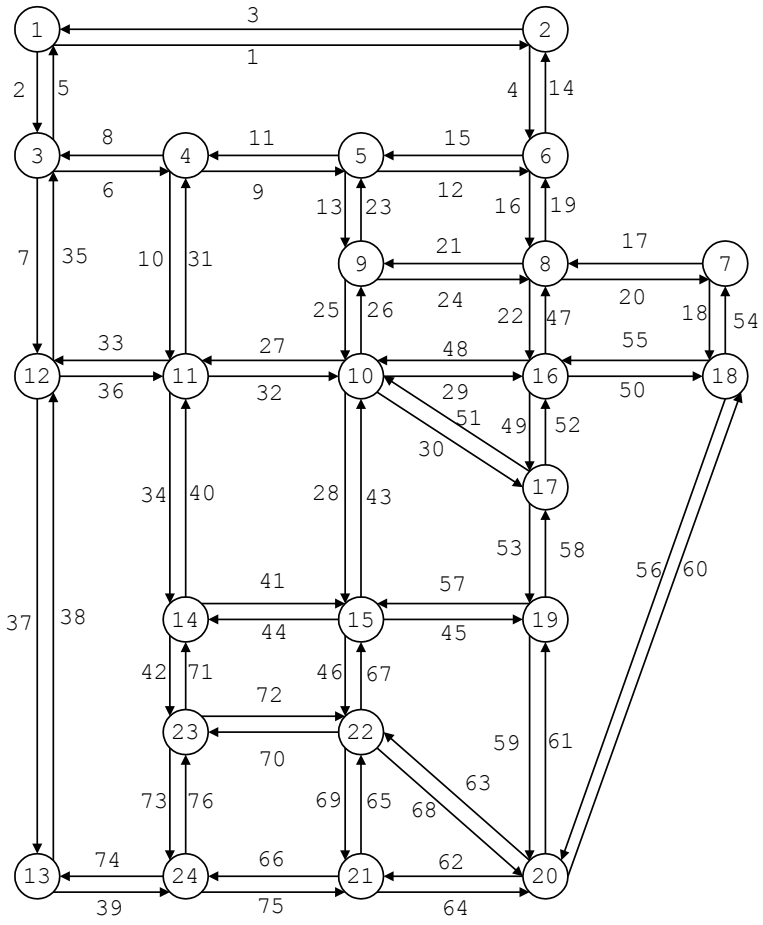

Figure 1: Sioux Falls Test Network