PROJECT 1: NETWORK ANALYSIS
ENG6 - Engineering Problem Solving
Spring 2016
UC Davis

Collaboration Policy:
This project is to be completed individually. You may talk at a high level about the project with up to 2 other students. They are your collaborators. By high level discussions, we mean that you can say things like `We need to use a For Loop to accomplish this task'. Once you start `speaking in Matlab code', you've gone too far. Collaborators may help each other debug their code, but the only hands that should ever touch the keyboard for your project are your own. Transferring even small portions of code to each other is certainly not allowed. You must submit your own solution to the project. At the top of your submission, state the names of all your collaborators. It will be your responsibility to make sure that all your collaborators are listed. Other than contacting the teaching assistants for clarification, you may not seek the assistance of other persons.

Grading Criteria:
Your computer program will be tested with an input file similar in format used in "Abilene.mat" provided with the assignment. The point assignment is indicated on each task. 75% of the points within each task will be assigned for correct execution. The remaining 25% will be divided into clarity, formatting style, and documentation of your code. No late submissions will be accepted, and no resubmissions will be accepted for credit.

Submission Checklist:
You must follow the following submission requirements closely.
- A script with the name **Analyze_Network** must execute your program.
- Your **Analyze_Network** and **ALL the supporting functions** that you have written must be submitted in one "zipped" folder that ***is given an identifiable name containing your initials and surname***. Don't forget to include *Abilene.mat*, *Dijkstra.m* and *listdijkstra.m* functions provided to you within the zipped folder.
- The "zipped" folder must be submitted through SmartSite.

It will be completely your responsibility to make sure that your zipped folder contains your **Analyze_Network.m** script and ***all the supporting functions*** are in the same folder as the script.

NOTE: **IF ANY OF THE SUPPORTING FUNCTIONS USED IN THE SCRIPT ARE NOT PRESENT IN THE FOLDER, YOU WILL AT MOST BE ABLE TO GET 25% OF THE GRADE. In the event that a supporting function is not present, you will be give ONE opportunity to resubmit within 24 hours after receiving a notice that there is a problem with your submission. You will not be allowed to submit corrections to your original submission, just correcting the submission error.**

Project Statement:

In today's world, computers are no more standalone entities but they are connected to several other systems over large geographical locations. Computer and communication networks have made this interconnection possible over the years, internet being its biggest brainchild. It is obvious to wonder how millions of such devices communicate with each other seamlessly over large geographical locations; the answer to that is most networks, including the internet, are arranged hierarchically. This means, as an end user, you first connect to your local Internet Service Provider (ISP) who would in turn connect to a regional ISP and finally they connect to a backbone network. In other words, backbone network is your core network extending over large geographical locations and, as expected, the bandwidth capacities of these backbones are very high compared to other links.

Abilene Network is one such high speed backbone network which spans over several cities across United States connecting universities and corporations where it was primarily used for research, educational and other experimental purposes. It is important to analyze the network to know the node's location (each connection point is called a node), connectivity between the nodes, data traffic demands & trends, etc. using which network operators can perform capacity planning, resource provisioning, routing etc. So in this project, you will be using Abilene network's dataset to perform network analysis to observe traffic trends, visually realize the nodes and their connectivity, calculate network path costs for routing, etc.

<u>The contents and format of the dataset are briefly explained below</u>:

Place the file **Abilene.mat** in your current working directory and type "load Abilene.mat" to load the data. **Abilene.mat** contains *city*, *NodeNumber, coordinates*, *LinkInfo*, *DemandPair, Abilene_All* matrices.

city: It is a character array consisting of all city names in the database. Here each city represents a node in the network and the row index gives the node number of the respective cities.

NodeNumber: Each city has a node number associated with it to keep track of them. Node numbers are given alphabetically to the cities. To make it simpler, you need not even use this NodeNumber cell array for majority of tasks, as 'city' character array has been arranged alphabetically, so its index would give you the node number.

coordinates: In this matrix, the first column gives the node number, the second column denotes the latitude, and the third column is longitude of the respective cities.

LinkInfo: This matrix gives the link connectivity between the nodes of the network. First two columns give us the node numbers of cities between which the link is present. Third column gives the Open Shortest Path First (OSPF) weight (a cost metric) of that link. OSPF gives the best route for packets of data as they pass through a set of connected nodes in the network. Note that, links are bidirectional (which means if there is link connectivity from city A to city B, then there is also link connectivity from city B to city A - traffic will flow in each direction).

DemandPair: Since there are 12 nodes in this dataset, there can be 144 node pairs with traffic demands. Traffic can flow from each node to all other nodes including itself (since each node also acts as a router, there can be internal traffic flow). Hence, 12 x 12 = 144 demand pairs. The row index gives the demand pair number and the values in that row indicate the node numbers of that particular demand pair.

Abilene_All: This is the Abilene traffic demand matrix, for all 144 demand pairs for a period of 4 weeks. There are 144 rows indicating each demand pair and the columns are traffic demands averaged over a 5 minute interval for 4 weeks, so we have 8064 columns *(12 5minute_intervals/hour * 24 hours/day * 7 days/week * 4 weeks = 8064 traffic data/5 minute)*. Each entry is a 5 minute traffic aggregate in bytes per second for that demand pair and hence there are 8064 entries for each of the 144 demand pairs.

Getting Started:

The terms *router*, *nodes*, *cities* might be used interchangeably in this project. **Since there are two routers in Atlanta, do consider them as two separate nodes/cities to avoid ambiguity (Atlanta and Atlanta2)**. The dataset given here too considers them as two separate entities in all respects.

Note that, the nodes mentioned here are backbone routers and not end user nodes; hence, the traffic matrix (TM) can contain traffic 'from' itself 'to' itself (thus the diagonals of TM may be non zero). If they were end user nodes, the diagonals would have been zero since traffic 'from' itself 'to' itself wouldn't been seen in the network. Remember, links are bi directional, so if there exists a link between node A and node B, then traffic can directly flow from node A to node B as well as from node B to node.

Adjacency matrix: You will be constructing an Adjacency Matrix based on *LinkInfo* and it will represent the connectivity between the nodes. It is a N x N matrix where N is the number of nodes and each (N_i,N_j) value determines if there is a link between N_i and N_j nodes. If there is a link, it is denoted by a **1**, else **0** at (N_i,N_j). Also be aware that, if *LinkInfo* shows that there is a link between node A and node B with OSPF weight W, it means that there is an equivalent link from node B to node A as well with the same OSPF weight W. This matrix is symmetric in our case.

OSPF Weight matrix: It is similar to Adjacency matrix but the link connections are represented by OSPF weights **W** instead of just **1**. For instance, if N_i and N_j have a link with OSPF weight **W**, then (N_i,N_j) = **W**, else **0**. This matrix is symmetric in our case.

In addition to the dataset, you are also provided with *dijkstra.m* and *listdijkstra.m* files which you will be using in Task 11 to calculate the *shortest path between any two nodes* using Dijkstra's algorithm. Your only job in this task will be to provide inputs to the *dijkstra.m* function and use the result from the function to display the *path* and *path cost (path cost is the summation of OSPF weights of all the links along that path)*. The **inputs** to *dijkstra.m* are *OSPF weight matrix*, *source node index*, *destination node index* and the function's **output** will be the *shortest path between those nodes*.
In summary, it is not required for you to understand what dijkstra.m does internally but if you are interested on how the algorithm works, watch this video *https://youtu.be/5GT5hYzjNoo*.

<u>Tasks:</u>

Task 1 (2pts): Include the following at the top of your script:

```
%% ENG 6 SQ 2016 Project 1
% First Name, Last Name
% Section Number
% UC Davis Student ID
% Collaborator 1 First Name, Last Name
% Collaborator 2 First Name, Last Name

clear all; % Clear any previous data
close all; %Close all previous figures
clc; % Clears the command window
load Abilene.mat; % Load the Abilene data
```

* <u>Each task is followed by a *sample output*. Display your results in that format.</u>

Task 2 (4pts): Determine how many backbone routers and links are given in the database?

Task 2: In total, 10 backbone routers and 20 links are given in the database.

Task 3 (6pts): Determine how many cities have a name starting with the letter 'S'?

Task 3: In total, 6 cities in the database have names starting with the letter S.

Task 4 (6pts): Determine how many cities have names longer than 8 characters (white space does not count as a character here)?

Task 4: 2 cities in the database have names longer than 8 characters.

Task 5 (8pts): Find out the top 3 cross-city demands (the demand should be between two different nodes; if the demand is between same nodes, then ignore them) which has the highest weekly average over the given 4 week period.

Task 5: The top three demands sorted according to highest weekly average are:
Seattle to Washington
Washington to Seattle
Kansas City to Denver

Task 6 (8pts): Determine on which week the average traffic from Chicago to Seattle was maximum?

Task 6: The average traffic from Chicago to Seattle was maximum during Week 3.

Task 7 (8pts): In *LinkInfo,* if there is a link between node A and node B with OSPF weight W, it means there is an equivalent link between node B and node A with same OSPF weight W since the links are bi-directional. Create a new matrix *BiLinkInfo* using the *LinkInfo* matrix so that the new matrix incorporates the bi-directional links. This new matrix will be a continuation of the matrix *LinkInfo* with the added rows for reverse direction links with the same OSPF weights W (the number of rows in *BiLinkInfo* will be double the number of rows in *LinkInfo*). An example is shown below:

LinkInfo:

Node	Node	Weight
A1	B1	W1
A2	B2	W2
A3	B3	W3

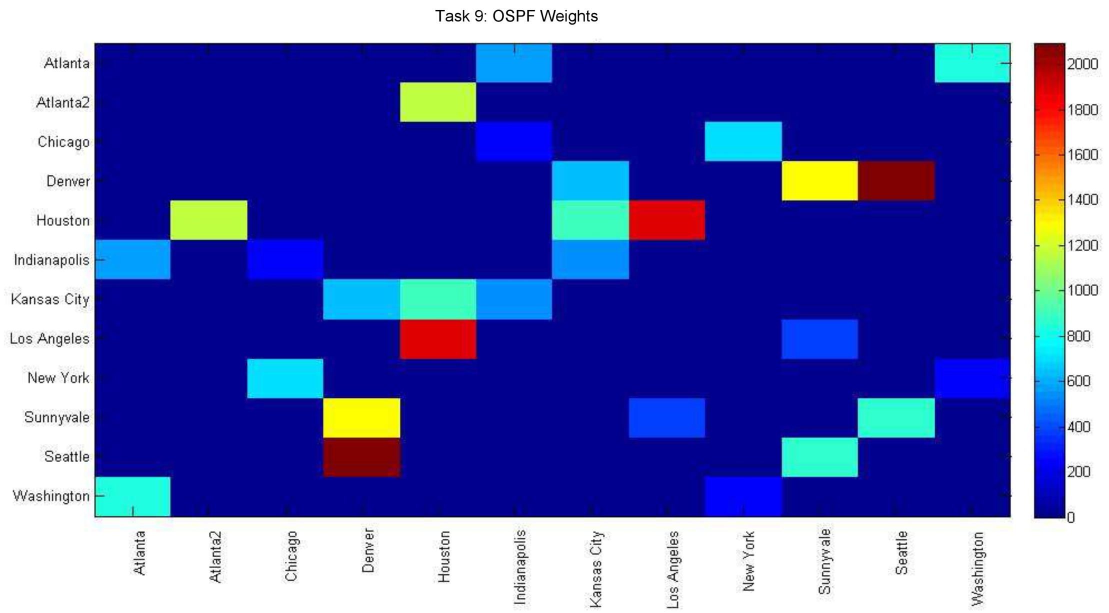
BiLinkInfo:

Node	Node	Weight
A1	B1	W1
A2	B2	W2
A3	B3	W3
B1	A1	W1
B2	A2	W2
B3	A3	W3

Task 8 (8pts): Use *BiLinkInfo* matrix from Task 7 to create the Adjacency matrix[1] *AdjMat* for the given data and display it. Make sure your *AdjMat* reflects the bi-directional link information. Finally, verify that your resulting adjacency matrix is symmetric.

[1]*The adjacency matrix of a simple labeled graph is a matrix with rows and columns labeled by graph vertices with a 1 or 0 in position (V_i,V_j) according to whether V_i and V_j are adjacent or not.*

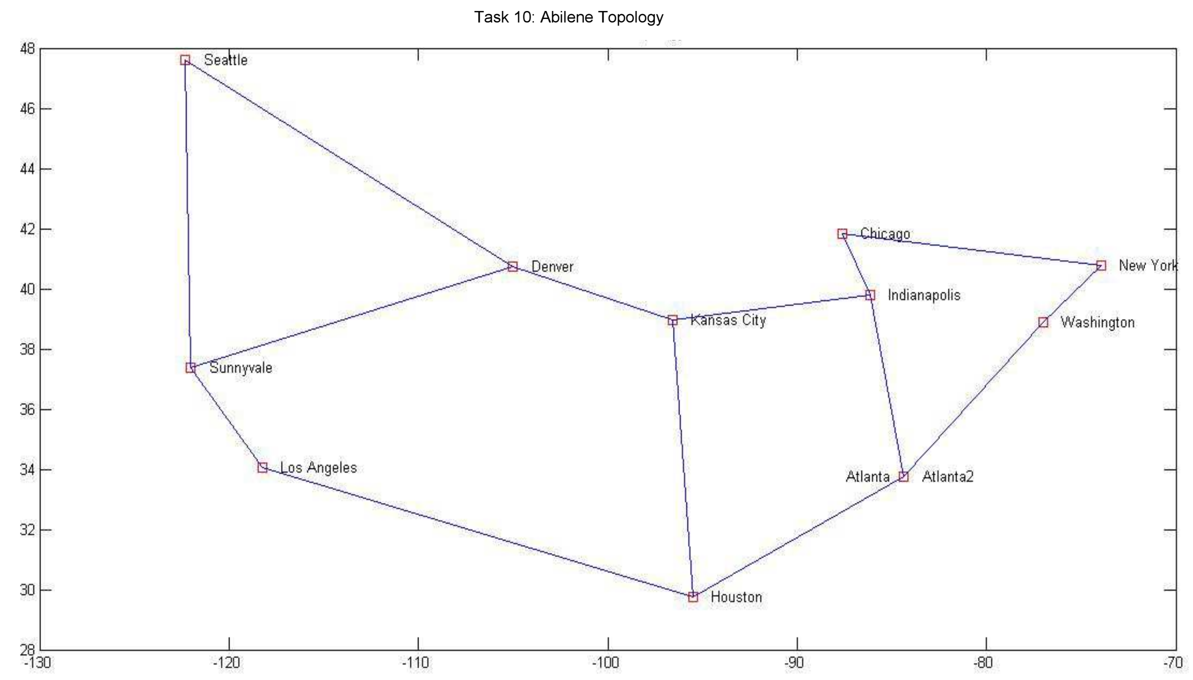
Task 9 (10pts): Using the Adjacency matrix *AdjMat* from Task 8 and OSPF weights from *BiLinkInfo*, create an OSPF weight matrix *OSPFWeight* where, instead of representing the links by 1, denote the links by their respective OSPF weights. Plot the resulting matrix as a scaled image plot (we recommend you use *imagesc* command) with a colorbar of what the color intensities mean. Also, label the axes ticks with the corresponding city names on both axes. An example of how the Figure should look like is shown below.



Note: The x-axis tick label should be rotated 90 degrees[2] so that the names do not overlap.

[2]*Formatting the labels on the x-axis is not trivial. For help figuring out how to do this, refer to the following websites: http://www.mathworks.com/help/matlab/creating_plots/change-tick-marks-and-tick-labels-of-graph-1.html and http://www.mathworks.com/videos/rotate-axes-labels-in-matlab-98218.html.*

Task 10 (10pts): Using the Adjacency matrix *AdjMat* and coordinates, plot the network topology. You are also required to mark the nodes as *red squares* and label each node with the respective city name. *Hint*: First plot the coordinates, hold your plot then label their names and finally plot the topology. There is a special plot function *gplot* for plotting graphs which you can use to plot this topology. The resulting plot might be rotated, so figure out a technique to view it the right way (so that it is consistent with the geographical locations). An example of how the Figure should look like is shown below.



Task 11 (12pts): In this task you'll be using the Dijkstra[3] function (*dijkstra.m* given with the project set) to find the shortest path between any two given nodes. Get the names of two different cities/nodes from user input (inputs could be case insensitive, so do comparisons accordingly) for which you need to find the shortest path. Pass the *OSPFWeight matrix*, *source node index,* and *destination node index* as inputs to the Dijkstra function and it will return the shortest path between the nodes as output. Map this output which is just the city index into city names and display them. You are also required to find the total cost of the shortest path which can be determined by summing the OSPF weights of all the links the shortest path passes through.

[3]If you are curious on how Dijkstra's algorithm works on finding the shortest path, watch this video: https://youtu.be/5GT5hYzjNoo . But it is not a requirement to know the algorithm to solve this task.

Task 11:
Enter the first node: los angeles
Enter the second node: kansas city
The shortest path for given pair of nodes is: Los Angeles > Sunnyvale > Denver > Kansas City.
The cost of the path is: 2300.

Task 12 (10pts): Plot the third day's traffic demand from Sunnyvale to Seattle, Houston to Denver and Indianapolis to Kansas City on a *single figure*. Make sure each of the plots are marked with different colors and include the legend as well. Do you see any trends in the traffic flow from the graph? Deduce what might be the reason for that variation and display your deductions/comments.

Task 13 (8pts): Write a custom function which creates a pop up menu (*hint*: use *menu* function) and displays a list of all cities. Once a city is selected, it should display the co-ordinates of its location and all the cities it is connected to on the command window.

```
Task 13:
Select a city from the Menu..

The coordinates of Chicago are 41.833300 and -87.616700.
Chicago is connected to: Indianapolis, New York.
```