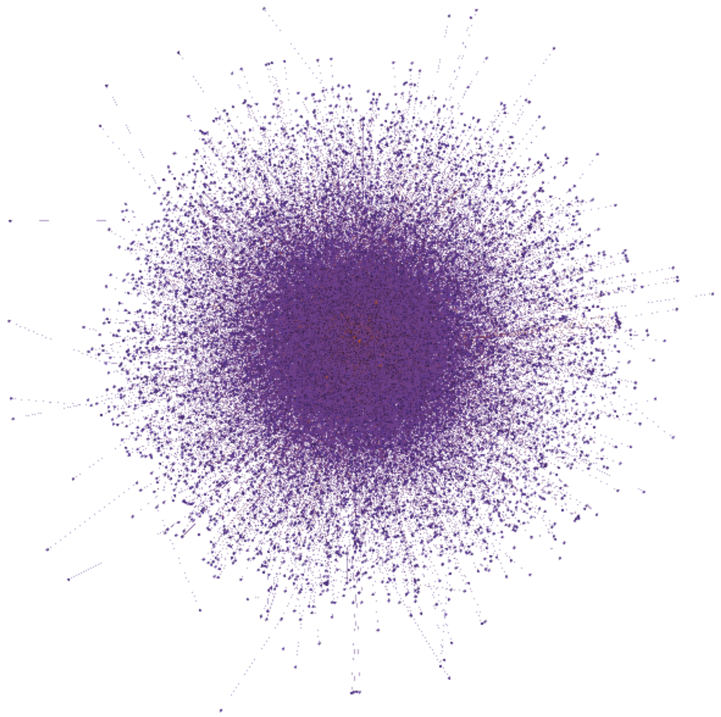
ASSIGNMENT 1

Kushal Hebbar | C13031425 | khebbar@g.clemson.edu

- **Graph library:** I have decided to make use of NetworkX graph library because of the vast amount of documentation and features.
- **Graph visualization tool:** I made use of Gephi visualization tool in order to produce the following graph which consists of 10,876 nodes and 39,994 edges available on Stanford Large Network Dataset Collection (SNAP).



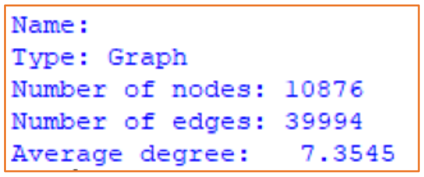
- **Dataset used:** SNAP: Stanford Large Network Dataset Collection. SNAP is a collection of more than 50 large network datasets from tens of thousands of nodes and edges to tens of millions of nodes and edges. It includes social networks, web graphs, road networks, internet networks, citation networks, collaboration networks, and communication networks.

    - **Read graphs using some graph library (atleast 10K nodes):** Used NetworkX to read a graph. The dataset used is Gnutella peer-to-peer file sharing network from August 2002.
      The code is as follows: (Written in python)

```
import networkx as nx

g = nx.read_edgelist('Gnutella.txt', create_using=nx.Graph(),nodetype=int)
print(nx.info(g))

nx.write_graphml(g,"Gnutella.graphml")
```



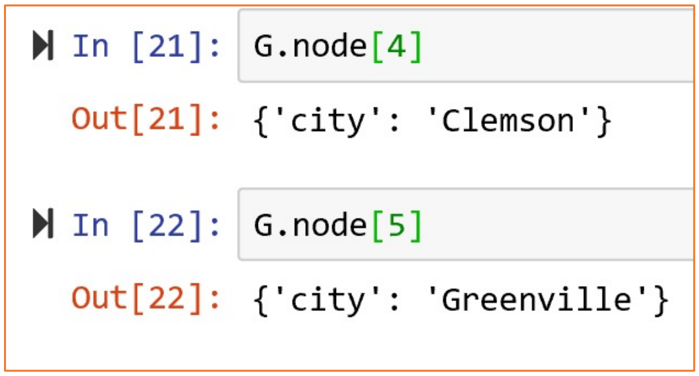
```
Name:
Type: Graph
Number of nodes: 10876
Number of edges: 39994
Average degree:   7.3545
```

The above code first reads the Gnutella peer-to-peer network using read_edgelist and then prints the information as seen above (blue text): it results in the type, number of nodes (10876), number of edges (39994) and Average degree of the graph (7.3545).

The graph is then written into a file in graphml format which can be viewed through a graph visualization tool such as Gephi.

- **Check if you can add attributes to nodes/edges:** Attributes can be added to both nodes and edges using NetworkX and the code is as follows:

```
G.add_nodes_from([1,10])
G.add_edges_from([(1,2), (2,3), (2,4), (3,6), (4,5), (7,9), (8,10), (1,10), (7,10)])
G.add_node(4, city='Clemson')
G.node[5]['city']='Greenville'
```



```
In [21]: G.node[4]
Out[21]: {'city': 'Clemson'}

In [22]: G.node[5]
Out[22]: {'city': 'Greenville'}
```

- **Implement and run one graph algorithm. Choose one from BFS, DFS, connected components, or shortest paths:**

➔ **I have implemented BFS algorithm:**

```
g = nx.read_edgelist('Gnutella.txt', create_using=nx.Graph(),nodetype=int)
bfs = nx.bfs_tree(g,'24')
print(tree.edges())
print(tree.nodes())
```

In the above code we can observe that the source node for the BFS traversal is 24.

- **Compute up to 5 smallest eigenvalues of graph Laplacian with one connected component (LAPACK++, GSL, NumPy, SLEPc, PETSc):**

```
import networkx as nx
import numpy.linalg

g = nx.Graph()
g = nx.read_edgelist('Gnutella.txt', create_using=nx.Graph(),nodetype=int)
L = nx.normalized_laplacian_matrix(g)
e = numpy.linalg.eigvals(L.A)

smallest_eigen_values = numpy.sort(e)
smallest_eigen_values[:5]
```

The above code is used to generate 5 smallest eigen values.

- normalized_laplacian_matrix : Used to generate a normalized Laplacian matrix of graph g
- numpy.linalg.eigvals : Used to compute eigenvalues
- smallest_eigen_values : Contains the sorted list of eigen values
- smallest_eigen_values[:5] : Results in the 5 smallest eigenvalues