

Assignment 1 - Creating and Manipulating Graphs

Eight employees at a small company were asked to choose 3 movies that they would most enjoy watching for the upcoming company movie night. These choices are stored in the file `Employee_Movie_Choices.txt`.

A second file, `Employee_Relationships.txt`, has data on the relationships between different coworkers.

The relationship score has value of -100 (Enemies) to +100 (Best Friends). A value of zero means the two employees haven't interacted or are indifferent.

Both files are tab delimited.

In [1]:

```

import networkx as nx
import pandas as pd
import numpy as np
from networkx.algorithms import bipartite

# This is the set of employees
employees = set(['Pablo',
                'Lee',
                'Georgia',
                'Vincent',
                'Andy',
                'Frida',
                'Joan',
                'Claude'])

# This is the set of movies
movies = set(['The Shawshank Redemption',
              'Forrest Gump',
              'The Matrix',
              'Anaconda',
              'The Social Network',
              'The Godfather',
              'Monty Python and the Holy Grail',
              'Snakes on a Plane',
              'Kung Fu Panda',
              'The Dark Knight',
              'Mean Girls'])

# you can use the following function to plot graphs
# make sure to comment it out before submitting to the autograder
def plot_graph(G, weight_name=None):
    """
    G: a networkx G
    weight_name: name of the attribute for plotting edge weights (if G is weighted)
    """
    %matplotlib notebook
    import matplotlib.pyplot as plt

    plt.figure()
    pos = nx.spring_layout(G)
    edges = G.edges()
    weights = None

    if weight_name:
        weights = [int(G[u][v][weight_name]) for u,v in edges]
        labels = nx.get_edge_attributes(G,weight_name)
        nx.draw_networkx_edge_labels(G,pos,edge_labels=labels)
        nx.draw_networkx(G, pos, edges=edges, width=weights);
    else:
        nx.draw_networkx(G, pos, edges=edges);

```

Question 1

Using NetworkX, load in the bipartite graph from `Employee_Movie_Choices.txt` and return that graph.

This function should return a networkx graph with 19 nodes and 24 edges

In [2]:

```
def answer_one():  
    return nx.read_edgelist('Employee_Movie_Choices.txt', delimiter="\t")  
  
answer_one()
```

Out[2]:

<networkx.classes.graph.Graph at 0x7f05c81344a8>

Question 2

Using the graph from the previous question, add nodes attributes named 'type' where movies have the value 'movie' and employees have the value 'employee' and return that graph.

This function should return a networkx graph with node attributes {'type': 'movie'} or {'type': 'employee'}

In [3]:

```
def answer_two():  
    # Your Code Here  
    G = answer_one()  
    for node in G.nodes():  
        if node in employees:  
            G.add_node(node, type="employee")  
        elif node in movies:  
            G.add_node(node, type="movie")  
    return G  
  
answer_two()
```

Out[3]:

<networkx.classes.graph.Graph at 0x7f05b1807b38>

Question 3

Find a weighted projection of the graph from `answer_two` which tells us how many movies different pairs of employees have in common.

This function should return a weighted projected graph.

In [4]:

```
def answer_three():  
  
    G = answer_two()  
    weighted_projected_graph = bipartite.weighted_projected_graph(G, employees)  
  
    return weighted_projected_graph  
  
answer_three()
```

Out[4]:

<networkx.classes.graph.Graph at 0x7f05b1825940>

Question 4

Suppose you'd like to find out if people that have a high relationship score also like the same types of movies.

Find the Pearson correlation (using `DataFrame.corr()`) between employee relationship scores and the number of movies they have in common. If two employees have no movies in common it should be treated as a 0, not a missing value, and should be included in the correlation calculation.

This function should return a float.

In [5]:

```

def answer_four():

    G = answer_three()
    Rel = nx.read_edgelist('Employee_Relationships.txt' ,data=[('relationship_score', int)])

    G_df = pd.DataFrame(G.edges(data=True), columns=['From', 'To', 'movies_score'])
    Rel_df = pd.DataFrame(Rel.edges(data=True), columns=['From', 'To', 'relationship_score'])

    G_copy_df = G_df.copy()
    G_copy_df.rename(columns={"From":"From1", "To":"From"}, inplace=True)
    G_copy_df.rename(columns={"From1":"To"}, inplace=True)

    G_final_df = pd.concat([G_df, G_copy_df])

    final_df = pd.merge(G_final_df, Rel_df, on = ['From', 'To'], how='right')

    final_df['movies_score'] = final_df['movies_score'].map(set_value)

    final_df['movies_score'] = final_df['movies_score'].map(lambda x: x['weight'])
    final_df['relationship_score'] = final_df['relationship_score'].map(lambda x: x['relationship_score'])

    value = final_df['movies_score'].corr(final_df['relationship_score'])

    return value

def set_value(val):
    if val is np.nan:
        return {'weight': 0}
    else:
        return val

answer_four()

```

Out[5]:

0.78839622217334737

In []: