*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

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():

    Gdf = pd.read_csv('Employee_Movie_Choices.txt', sep='\t',header=None, skiprows = 1, names=['Employees', 'Movies'])
    Gra1 = nx.from_pandas_dataframe(Gdf, 'Employees', 'Movies')
    return Gra1

answer_one()
```

Out[2]:
```
<networkx.classes.graph.Graph at 0x7f7d950f15f8>
```

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():


    Gra2 = answer_one()
    Gra2.add_nodes_from(employees, bipartite=0, type = 'employee')
    Gra2.add_nodes_from(movies, bipartite=1, type = 'movie')
    return Gra2

answer_two()
```

Out[3]:
```
<networkx.classes.graph.Graph at 0x7f7d950c1710>
```

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():

    Gra3 = answer_two()
    weighted_projection = bipartite.weighted_projected_graph(Gra3, employees)
    return weighted_projection

#plot_graph(answer_three())
answer_three()
```

Out[4]:
```
<networkx.classes.graph.Graph at 0x7f7d668c5208>
```

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():

    Rel = nx.read_edgelist('Employee_Relationships.txt', data=[('relationship_score', int)])
    Rel_df = pd.DataFrame(Rel.edges(data=True), columns=['From', 'To', 'relationship_score'])
    G = answer_three()
    Gdf = pd.DataFrame(G.edges(data=True), columns=['From', 'To', 'movies_score'])


    Gdf_doppel = Gdf.copy()
    Gdf_doppel.rename(columns={"From":"From_", "To":"From"}, inplace=True)
    Gdf_doppel.rename(columns={"From_":"To"}, inplace=True)
    Gdf_result = pd.concat([Gdf, Gdf_doppel])

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

    result_df['movies_score'] = result_df['movies_score'].map(lambda x: x['weight'] if type(x)==dict else None)
    result_df['relationship_score'] = result_df['relationship_score'].map(lambda x: x['relationship_score'])
    result_df['movies_score'].fillna(value=0, inplace=True)


    return result_df['movies_score'].corr(result_df['relationship_score'])

answer_four()
```

Out[5]:
```
0.78839622217334737
```

In [ ]:
```

```