Clustering Assignment

There will be some functions that start with the word "grader" ex: grader_actors(), grader_movies(), grader_cost1() etc, you should not change those function definition.

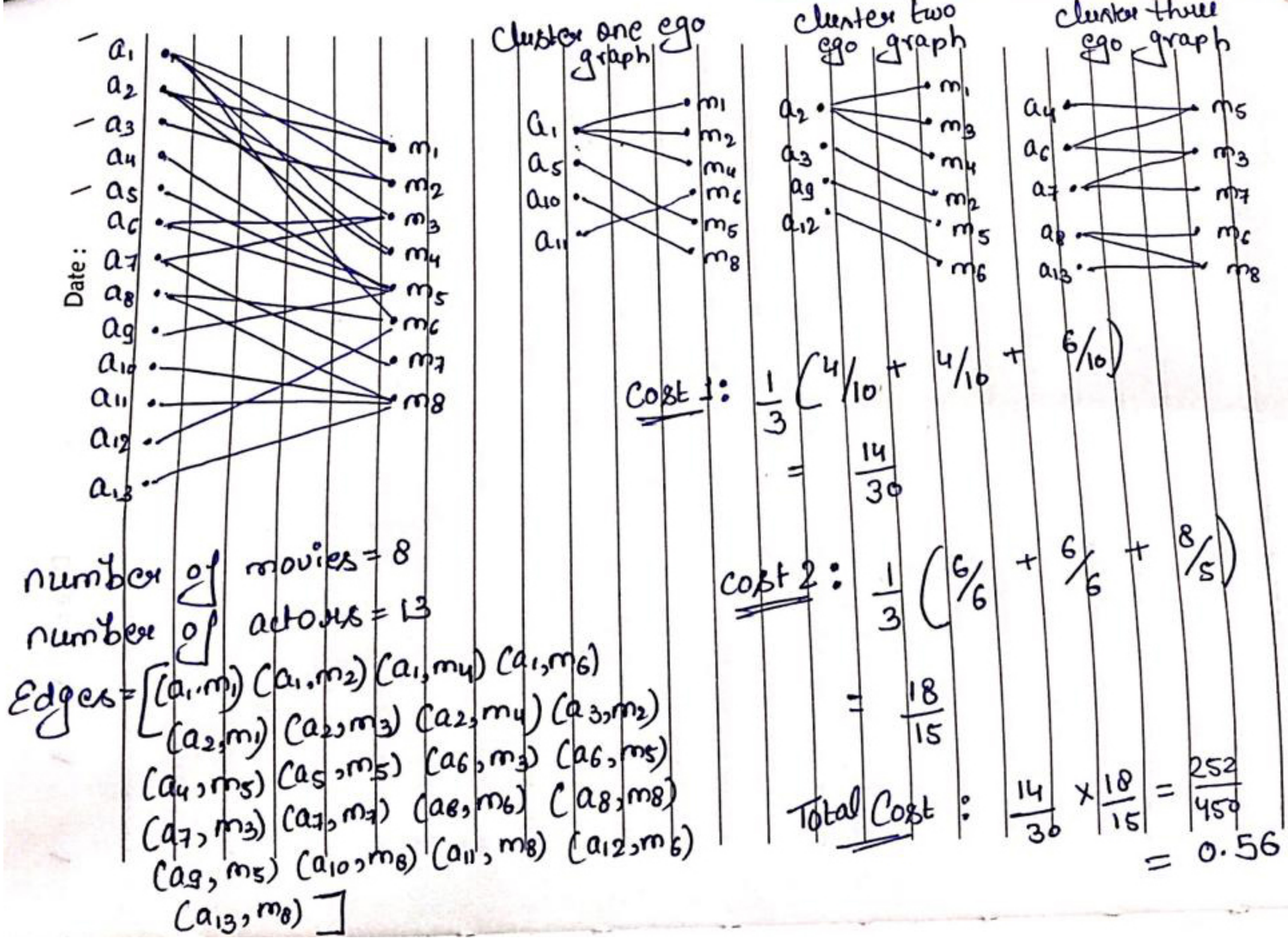
Every Grader function has to return True.

Please check clustering assignment helper functions notebook before attempting this assignment.

- Read graph from the given movie_actor_network.csv (note that the graph is bipartite graph.)
- Using stellergaph and gensim packages, get the dense representation(128dimensional vector) of every node in the graph. [Refer Clustering_Assignment_Reference.ipynb]
- Split the dense representation into actor nodes, movies nodes.(Write you code in def data_split())

Task 1 : Apply clustering algorithm to group similar actors

1. For this task consider only the actor nodes
2. Apply any clustering algorithm of your choice
   Refer : https://scikit-learn.org/stable/modules/clustering.html
3. Choose the number of clusters for which you have maximum score of $Cost1 * Cost2$
4. Cost1 =
   $\frac{1}{N}$
   $\sum_{\text{each cluster i}} \frac{(\text{number of nodes in the largest connected component in the graph with the actor nodes and its movie neighbours in clus}}{(\text{total number of nodes in that cluster i})}$
   where N= number of clusters
   (Write your code in def cost1())
5. Cost2 =
   $\frac{1}{N} \sum_{\text{each cluster i}} \frac{(\text{sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours in cluster i})}{(\text{number of unique movie nodes in the graph with the actor nodes and its movie neighbours in cluster i})}$
   where N= number of clusters
   (Write your code in def cost2())
6. Fit the clustering algorithm with the opimal number_of_clusters and get the cluster number for each node
7. Convert the d-dimensional dense vectors of nodes into 2-dimensional using dimensionality reduction techniques (preferably TSNE)
8. Plot the 2d scatter plot, with the node vectors after step e and give colors to nodes such that same cluster nodes will have same color



Task 2 : Apply clustering algorithm to group similar movies

1. For this task consider only the movie nodes
2. Apply any clustering algorithm of your choice 3.Choose the number of clusters for which you have maximum score of $Cost1 * Cost2$

   Cost1 =
   $\frac{1}{N}$
   $\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the movie nodes and its actor neighbours in clus}}{\text{(total number of nodes in that cluster i)}}$
   where N= number of clusters
   (Write your code in def cost1())

3. Cost2 =
   $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of movie nodes in the graph with the movie nodes and its actor neighbours in cluster i)}}{\text{(number of unique actor nodes in the graph with the movie nodes and its actor neighbours in cluster i)}}$ where
   N= number of clusters
   (Write your code in def cost2())

Algorithm for actor nodes

```
for number_of_clusters in [3, 5, 10, 30, 50, 100, 200, 500]:
```

```
        algo = clustering_algorith(clusters=number_of_clusters)
        # you will be passing a matrix of size N*d where N number of actor
nodes and d is dimension from gensim
        algo.fit(the dense vectors of actor nodes)
        You can get the labels for corresponding actor nodes (algo.labels_)
        Create a graph for every cluster(ie., if n_clusters=3, create 3 graphs)
        (You can use ego_graph to create subgraph from the actual graph)
        compute cost1,cost2
           (if n_cluster=3, cost1=cost1(graph1)+cost1(graph2)+cost1(graph3) #
here we are doing summation
            cost2=cost2(graph1)+cost2(graph2)+cost2(graph3)
        computer the metric Cost = Cost1*Cost2
    return number_of_clusters which have maximum Cost
```

In [2]:
```
!pip install networkx==2.3
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publ
ic/simple/
Collecting networkx==2.3
  Downloading networkx-2.3.zip (1.7 MB)
     |████████████████████████████████| 1.7 MB 5.2 MB/s
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/dist-package
s (from networkx==2.3) (4.4.2)
Building wheels for collected packages: networkx
  Building wheel for networkx (setup.py) ... done
  Created wheel for networkx: filename=networkx-2.3-py2.py3-none-any.whl size=1556008 sh
a256=7cf6d634c262e3cffc371a599bd8ba75ccfe9c6f0cc86522c3f49707d684f90a
  Stored in directory: /root/.cache/pip/wheels/44/e6/b8/4efaab31158e9e9ca9ed80b11f6b1113
0bac9a9672b3cbbeaf
Successfully built networkx
Installing collected packages: networkx
  Attempting uninstall: networkx
    Found existing installation: networkx 2.6.3
    Uninstalling networkx-2.6.3:
      Successfully uninstalled networkx-2.6.3
ERROR: pip's dependency resolver does not currently take into account all the packages t
hat are installed. This behaviour is the source of the following dependency conflicts.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which is
 incompatible.
Successfully installed networkx-2.3
```

In [3]:
```
!pip install stellargraph
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publ
ic/simple/
Collecting stellargraph
  Downloading stellargraph-1.2.1-py3-none-any.whl (435 kB)
     |████████████████████████████████| 435 kB 5.2 MB/s
Requirement already satisfied: matplotlib>=2.2 in /usr/local/lib/python3.7/dist-packages
(from stellargraph) (3.2.2)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.7/dist-packa
ges (from stellargraph) (1.0.2)
Requirement already satisfied: networkx>=2.2 in /usr/local/lib/python3.7/dist-packages
(from stellargraph) (2.3)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.7/dist-packages (f
rom stellargraph) (1.3.5)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (f
rom stellargraph) (1.4.1)
Requirement already satisfied: gensim>=3.4.0 in /usr/local/lib/python3.7/dist-packages
(from stellargraph) (3.6.0)
```

```
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python3.7/dist-packages (fr
om stellargraph) (1.21.6)
Requirement already satisfied: tensorflow>=2.1.0 in /usr/local/lib/python3.7/dist-packag
es (from stellargraph) (2.8.2+zzzcolab20220527125636)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-packag
es (from gensim>=3.4.0->stellargraph) (5.2.1)
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (fro
m gensim>=3.4.0->stellargraph) (1.15.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packag
es (from matplotlib>=2.2->stellargraph) (1.4.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (f
rom matplotlib>=2.2->stellargraph) (0.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/li
b/python3.7/dist-packages (from matplotlib>=2.2->stellargraph) (3.0.9)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-pac
kages (from matplotlib>=2.2->stellargraph) (2.8.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packag
es (from kiwisolver>=1.0.1->matplotlib>=2.2->stellargraph) (4.1.1)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/dist-package
s (from networkx>=2.2->stellargraph) (4.4.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (f
rom pandas>=0.24->stellargraph) (2022.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (f
rom scikit-learn>=0.20->stellargraph) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-pac
kages (from scikit-learn>=0.20->stellargraph) (3.1.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.7/dist-packag
es (from tensorflow>=2.1.0->stellargraph) (3.3.0)
Requirement already satisfied: flatbuffers>=1.12 in /usr/local/lib/python3.7/dist-packag
es (from tensorflow>=2.1.0->stellargraph) (2.0)
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.7/dist-packages
(from tensorflow>=2.1.0->stellargraph) (14.0.1)
Requirement already satisfied: tensorflow-estimator<2.9,>=2.8 in /usr/local/lib/python3.
7/dist-packages (from tensorflow>=2.1.0->stellargraph) (2.8.0)
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.7/dist-packages
(from tensorflow>=2.1.0->stellargraph) (1.1.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.7/dist-packages
(from tensorflow>=2.1.0->stellargraph) (1.14.1)
Requirement already satisfied: gast>=0.2.1 in /usr/local/lib/python3.7/dist-packages (fr
om tensorflow>=2.1.0->stellargraph) (0.5.3)
Requirement already satisfied: keras<2.9,>=2.8.0rc0 in /usr/local/lib/python3.7/dist-pac
kages (from tensorflow>=2.1.0->stellargraph) (2.8.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.7/dist-packages (fr
om tensorflow>=2.1.0->stellargraph) (3.1.0)
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.7/di
st-packages (from tensorflow>=2.1.0->stellargraph) (1.1.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (fro
m tensorflow>=2.1.0->stellargraph) (57.4.0)
Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/dist-pa
ckages (from tensorflow>=2.1.0->stellargraph) (3.17.3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/py
thon3.7/dist-packages (from tensorflow>=2.1.0->stellargraph) (0.26.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.7/dist-pack
ages (from tensorflow>=2.1.0->stellargraph) (1.46.3)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.7/dist-packag
es (from tensorflow>=2.1.0->stellargraph) (1.6.3)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.7/dist-pack
ages (from tensorflow>=2.1.0->stellargraph) (0.2.0)
Requirement already satisfied: tensorboard<2.9,>=2.8 in /usr/local/lib/python3.7/dist-pa
ckages (from tensorflow>=2.1.0->stellargraph) (2.8.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-package
s (from tensorflow>=2.1.0->stellargraph) (1.1.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.7/dist-packa
ges (from astunparse>=1.6.0->tensorflow>=2.1.0->stellargraph) (0.37.1)
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages
(from h5py>=2.9.0->tensorflow>=2.1.0->stellargraph) (1.5.2)
```

```
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (0.6.1)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (2.23.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (1.8.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (1.35.0)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (1.0.1)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (0.4.6)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (3.3.7)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (4.2.4)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (4.8)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (0.2.8)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist-packages (from google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (1.3.1)
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (4.11.4)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.8->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (3.8.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (0.4.8)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.21.0->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (2022.6.15)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.9,>=2.8->tensorflow>=2.1.0->stellargraph) (3.2.0)
Installing collected packages: stellargraph
Successfully installed stellargraph-1.2.1
```

In [4]:

```python
import networkx as nx
from networkx.algorithms import bipartite
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
# you need to have tensorflow
from stellargraph.data import UniformRandomMetaPathWalk
from stellargraph import StellarGraph
```

In [5]:
```
#uploading files from system to import them here in colab
from google.colab import files
files = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving movie_actor_network.csv to movie_actor_network.csv
```

In [6]:
```
data=pd.read_csv('movie_actor_network.csv', index_col=False, names=['movie','actor'])
```

In [7]:
```
edges = [tuple(x) for x in data.values.tolist()]
```

In [8]:
```
B = nx.Graph()
B.add_nodes_from(data['movie'].unique(), bipartite=0, label='movie')
B.add_nodes_from(data['actor'].unique(), bipartite=1, label='actor')
B.add_edges_from(edges, label='acted')
```

In [9]:
```
A = list(nx.connected_component_subgraphs(B))[0] #getting all the connections in the gra
```

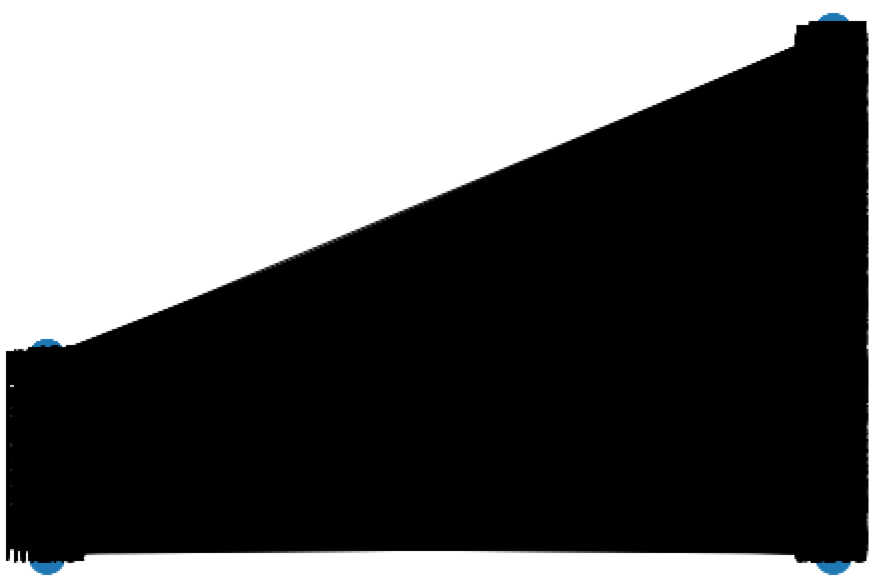
In [10]:
```
print("number of nodes", A.number_of_nodes())
print("number of edges", A.number_of_edges())
```

```
number of nodes 4703
number of edges 9650
```

In [11]:
```
#creating bipartite sets - movies (label - 1) and actors (label - 2). Then plotting the
l, r = nx.bipartite.sets(A)
pos = {}

pos.update((node, (1, index)) for index, node in enumerate(l))
pos.update((node, (2, index)) for index, node in enumerate(r))

nx.draw(A, pos=pos, with_labels=True)
plt.show()
```



In [12]:
```
movies = []
actors = []
for i in A.nodes():
    if 'm' in i:
        movies.append(i)
    if 'a' in i:
        actors.append(i)
print('number of movies ', len(movies))
print('number of actors ', len(actors))
```

```
number of movies  1292
number of actors  3411
```

In [13]:
```python
#Randomly selects a node then another to reach from the one previously selected and so o
# Create the random walker
rw = UniformRandomMetaPathWalk(StellarGraph(A))

# specify the metapath schemas as a list of lists of node types.
metapaths = [
    ["movie", "actor", "movie"],
    ["actor", "movie", "actor"]
]

walks = rw.run(nodes=list(A.nodes()), # root nodes
               length=100,   # maximum length of a random walk
               n=1,         # number of random walks per root node
               metapaths=metapaths
              )

print("Number of random walks: {}".format(len(walks)))
```

```
Number of random walks: 4703
```

In [14]:
```python
from gensim.models import Word2Vec
model = Word2Vec(walks, size=128, window=5)
```

In [15]:
```python
model.wv.vectors.shape  # 128-dimensional vector for each node in the graph
```

Out[15]:
```
(4703, 128)
```

In [16]:
```python
# Retrieve node embeddings and corresponding subjects
node_ids = model.wv.index2word  # list of node IDs
node_embeddings = model.wv.vectors  # numpy.ndarray of size number of nodes times embedd
node_targets = [ A.node[node_id]['label'] for node_id in node_ids]
```

In [17]:
```python
node_embeddings.shape
```

Out[17]:
```
(4703, 128)
```

```python
print(node_ids[:15], end='')
```

```
['a973', 'a967', 'a964', 'a1731', 'a969', 'a970', 'a1028', 'a1057', 'a965', 'a1003', 'm1094', 'a966', 'm67', 'a988', 'm1111']
```

```python
print(node_targets[:15],end='')
```

```
['actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'actor', 'movie', 'actor', 'movie', 'actor', 'movie']
```

In [18]:
```python
def data_split(node_ids,node_targets,node_embeddings):
    '''In this function, we will split the node embeddings into actor_embeddings , movie
    actor_nodes,movie_nodes=[],[]
    actor_embeddings,movie_embeddings=[],[]
    # split the node_embeddings into actor_embeddings,movie_embeddings based on node_ids
    # By using node_embedding and node_targets, we can extract actor_embedding and movie
    # By using node_ids and node_targets, we can extract actor_nodes and movie nodes
    for i in range(len(node_ids)):
      if 'm' in node_ids[i]:
        movie_nodes.append(node_ids[i])
        movie_embeddings.append(node_embeddings[i])
      else:
        actor_nodes.append(node_ids[i])
        actor_embeddings.append(node_embeddings[i])
    return actor_nodes,movie_nodes,actor_embeddings,movie_embeddings
```

In [19]:
```python
actor_nodes,movie_nodes,actor_embeddings,movie_embeddings = data_split(node_ids,node_tar
```

Grader function - 1

In [20]:
```
def grader_actors(data):
    assert(len(data)==3411)
    return True
grader_actors(actor_nodes)
```

Out[20]:
```
True
```

Grader function - 2

In [21]:
```
def grader_movies(data):
    assert(len(data)==1292)
    return True
grader_movies(movie_nodes)
```

Out[21]:
```
True
```

Calculating cost1

Cost1 =

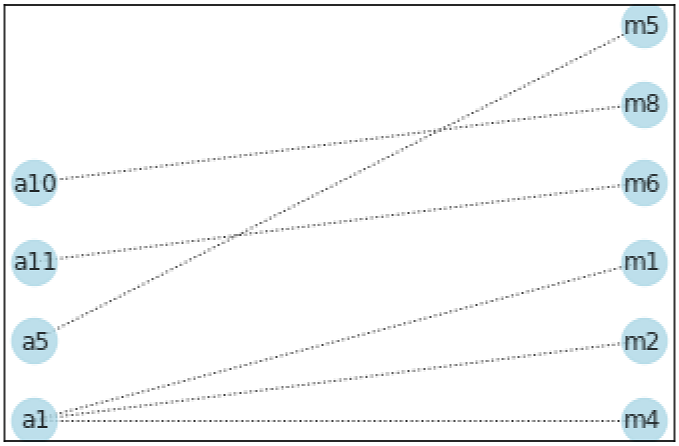
$$\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(number of nodes in the largest connected component in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(total number of nodes in that cluster i)}}$$

where N= number of clusters

In [22]:
```
def cost1(graph,number_of_clusters):
    '''In this function, we will calculate cost1'''
    #The maximum of connected components in the graph -> the function returns a set of n
    # i.e the connection which has the maximum no. of nodes
    cost1= len(max(nx.connected_components(graph)))/len(graph.nodes())

    return cost1/number_of_clusters
```

In [23]:
```
import networkx as nx
from networkx.algorithms import bipartite
graded_graph= nx.Graph()
graded_graph.add_nodes_from(['a1','a5','a10','a11'], bipartite=0) # Add the node attribu
graded_graph.add_nodes_from(['m1','m2','m4','m6','m5','m8'], bipartite=1)
graded_graph.add_edges_from([('a1','m1'),('a1','m2'),('a1','m4'),('a11','m6'),('a5','m5'
l={'a1','a5','a10','a11'};r={'m1','m2','m4','m6','m5','m8'}
pos = {}
pos.update((node, (1, index)) for index, node in enumerate(l))
pos.update((node, (2, index)) for index, node in enumerate(r))
nx.draw_networkx(graded_graph, pos=pos, with_labels=True,node_color='lightblue',alpha=0.
```



Grader function - 3

In [24]:
```
graded_cost1=cost1(graded_graph,3)
def grader_cost1(data):
    assert(data==((1/3)*(4/10))) # 1/3 is number of clusters
    return True
grader_cost1(graded_cost1)
```

Out[24]:
```
True
```

Calculating cost2

Cost2 = $\frac{1}{N}\sum_{\text{each cluster i}} \frac{\text{(sum of degress of actor nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}{\text{(number of unique movie nodes in the graph with the actor nodes and its movie neighbours in cluster i)}}$
where N= number of clusters

In [25]:
```
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost2'''
    cost2= len(graph.edges())/len([x for x in set(graph.nodes()) if 'm' in x])

    return cost2/number_of_clusters
```

Grader function - 4

In [26]:
```
graded_cost2=cost2(graded_graph,3)
def grader_cost2(data):
    assert(data==((1/3)*(6/6))) # 1/3 is number of clusters
    return True
grader_cost2(graded_cost2)
```

Out[26]:
```
True
```

Grouping similar actors

In [27]:
```
from sklearn.cluster import KMeans

clusters = [3, 5, 10, 30, 50, 100, 200, 500]
total_cost = {}
for c in clusters:
 model1 = KMeans(n_clusters = c).fit(actor_embeddings)
 labels = model1.labels_
 COST1 = 0
 COST2 = 0
 for i in range(c):
   cluster_nodes = np.where((labels == i))
   egos = np.array(actor_nodes)[cluster_nodes]
```

```
    G = nx.Graph()
    for e in egos:
      subgraph = nx.ego_graph(B, e)
      G.add_nodes_from(subgraph.nodes)
      G.add_edges_from(subgraph.edges())
    COST1 += cost1(G, c)
    COST2 += cost2(G, c)
  total_cost[c] = COST1*COST2
print(total_cost)
```

```
{3: 3.734891825142042, 5: 2.539175292259236, 10: 2.2938889266038136, 30: 1.7585200651374
981, 50: 1.505295950687873, 100: 1.5292169239343842, 200: 1.6371217835383158, 500: 1.726
5168345454027}
```

In [36]:

```
#Finding the optimal k and fitting the data using that k value.
optimal_k = [k for k in total_cost.keys() if total_cost[k]==max(total_cost.values())][0]
print("The value of optimal k is: ",optimal_k)

optimal_model = KMeans(n_clusters = optimal_k).fit(actor_embeddings)
labels_for_optimal = optimal_model.labels_
```

```
The value of optimal k is:  3
```

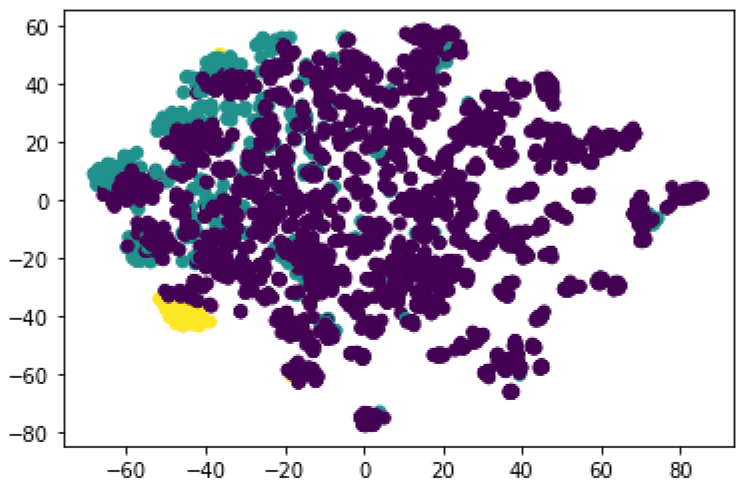
Displaying similar actor clusters

In [40]:

```
from sklearn.manifold import TSNE
actor_embeddings_2d = TSNE(n_components = 2).fit_transform(actor_embeddings)

plt.scatter(actor_embeddings_2d[:,0], actor_embeddings_2d[:,1], c = labels_for_optimal.a
plt.title("SCATTER PLOT OF SIMILAR ACTOR CLUSTERS AFTER DIMENSIONALITY REDUCTION")
plt.show()
```

SCATTER PLOT OF SIMILAR ACTOR CLUSTERS AFTER DIMENSIONALITY REDUCTION



Grouping similar movies

In [44]:

```
#modified to accomodate the cost2 formula calculations for movie clustering
def cost2(graph,number_of_clusters):
    '''In this function, we will calculate cost2'''
    cost2= len(graph.edges())/len([x for x in set(graph.nodes()) if 'a' in x])

    return cost2/number_of_clusters
```

In [45]:

```
clusters = [3, 5, 10, 30, 50, 100, 200, 500]
total_cost = {}
for c in clusters:
 model1 = KMeans(n_clusters = c).fit(movie_embeddings)
 labels = model1.labels_
 COST1 = 0
```

```
 COST2 = 0
 for i in range(c):
   cluster_nodes = np.where((labels == i))
   egos = np.array(movie_nodes)[cluster_nodes]
   G = nx.Graph()
   for e in egos:
     subgraph = nx.ego_graph(B, e)
     G.add_nodes_from(subgraph.nodes)
     G.add_edges_from(subgraph.edges())
   COST1 += cost1(G, c)
   COST2 += cost2(G, c)
 total_cost[c] = COST1*COST2
print(total_cost)
```

```
{3: 2.905417701748463, 5: 2.605639617553022, 10: 1.9089047128678454, 30: 1.9891174887289
167, 50: 1.843497509287282, 100: 1.540082506092833, 200: 1.3752132789793758, 500: 1.2080
63399602931}
```

In [46]:

```
#Finding the optimal k and fitting the data using that k value.
optimal_k = [k for k in total_cost.keys() if total_cost[k]==max(total_cost.values())][0]
print("The value of optimal k is: ",optimal_k)

optimal_model = KMeans(n_clusters = optimal_k).fit(movie_embeddings)
labels_for_optimal = optimal_model.labels_
```

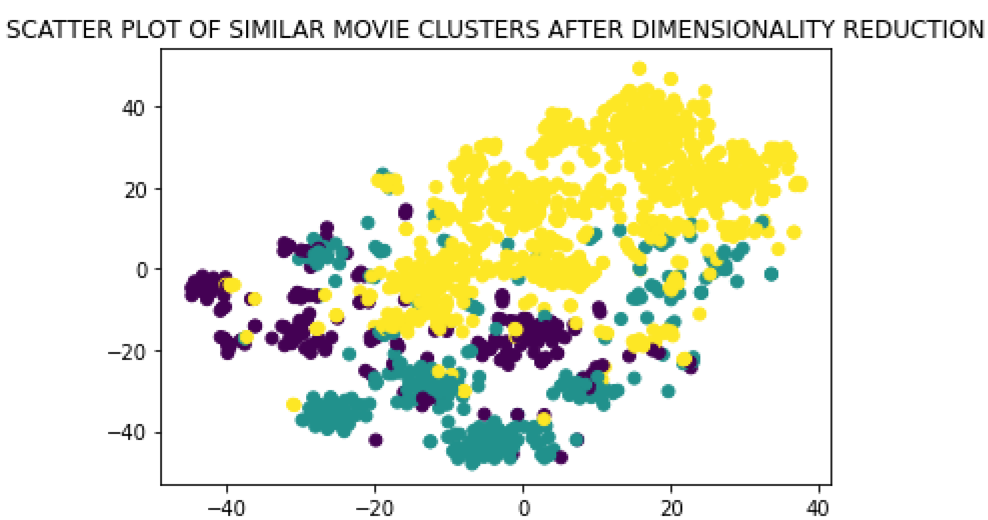
```
The value of optimal k is:  3
```

Displaying similar movie clusters

In [47]:

```
movie_embeddings_2d = TSNE(n_components = 2).fit_transform(movie_embeddings)

plt.scatter(movie_embeddings_2d[:,0], movie_embeddings_2d[:,1], c = labels_for_optimal.a
plt.title("SCATTER PLOT OF SIMILAR MOVIE CLUSTERS AFTER DIMENSIONALITY REDUCTION")
plt.show()
```



In [ ]: