Assignment2

September 1, 2021

1 Assignment-2

1.0.1 Name: P. V. Sriram

1.0.2 Roll No.: 1801CS37

2 Centrality Functions

[1]:
```
import numpy as np
import networkx as nx
from sklearn.metrics.pairwise import cosine_similarity
from networkx.algorithms.link_analysis.pagerank_alg import pagerank
from networkx.algorithms.centrality import eigenvector_centrality,␣
↪katz_centrality, betweenness_centrality
```

[2]:
```
# Utility fumctiom to print Dictionaries
def dict_print(dictionary):
    print("Node\tCentrality")
    for key, value in dict(dictionary).items():
        print(key, ' : ', value)
```

[3]:
```
# Centrality Class to calculate different
# Kinds of centrality measures
class Centrality():
    def __init__(self, Graph):
        self.graph = Graph
        self.eigen = None
        self.katz = None
        self.page = None
        self.btwn = None
        self.adj_matrix = None

    # Calculate Eigen Vector Centrality
    # and Sort in descending order
    def eigen_centrality(self):
        kv= eigenvector_centrality(self.graph)
        self.eigen = sorted(kv.items(), key = lambda kv:(kv[1]))
        return(self.eigen)
```

```
    # Calculate Katz Centrality
    # and Sort in descending order
    def katz__centrality(self):
        kv = katz_centrality(self.graph)
        self.katz = sorted(kv.items(), key = lambda kv:(kv[1]))
        return(self.katz)

    # Calculate Page Rank Centrality
    # and Sort in descending order
    def pg_rank_centrality(self):
        kv = pagerank(self.graph)
        self.page = sorted(kv.items(), key = lambda kv:(kv[1]))
        return(self.page)

    # Calculate Betweenness Centrality
    # and Sort in descending order
    def btwn_centrality(self):
        kv = betweenness_centrality(self.graph)
        self.btwn = sorted(kv.items(), key = lambda kv:(kv[1]))
        return(self.btwn)

    # Print node order for each measure
    def centrality_measures(self):
        print("Node ranks based on Eigen Vector Centrality:")
        self.eigen_centrality()
        dict_print(self.eigen)

        print("\nNode ranks based on Katz Centrality:")
        self.katz__centrality()
        dict_print(self.katz)

        print("\nNode ranks based on Page rank Centrality:")
        self.pg_rank_centrality()
        dict_print(self.page)

        print("\nNode ranks based on Betweenness Centrality:")
        self.btwn_centrality()
        dict_print(self.btwn)

    # Claculate most similar nodes
    # In terms of Cosine Similarity
    def cosine_sim(self):
        self.adj_matrix = nx.linalg.graphmatrix.adjacency_matrix(self.graph)
        sim = cosine_similarity(self.adj_matrix)
        sim[range(len(sim)), range(len(sim))] = -np.inf
        max_ = np.amax(sim)
```

```
        coords = np.unravel_index(np.argmax(sim, axis=None), sim.shape)
        print("\nMost similar nodes are: ", coords[0], coords[1], " with␣
↪similarity: ", max_)
```

3 Problem 1

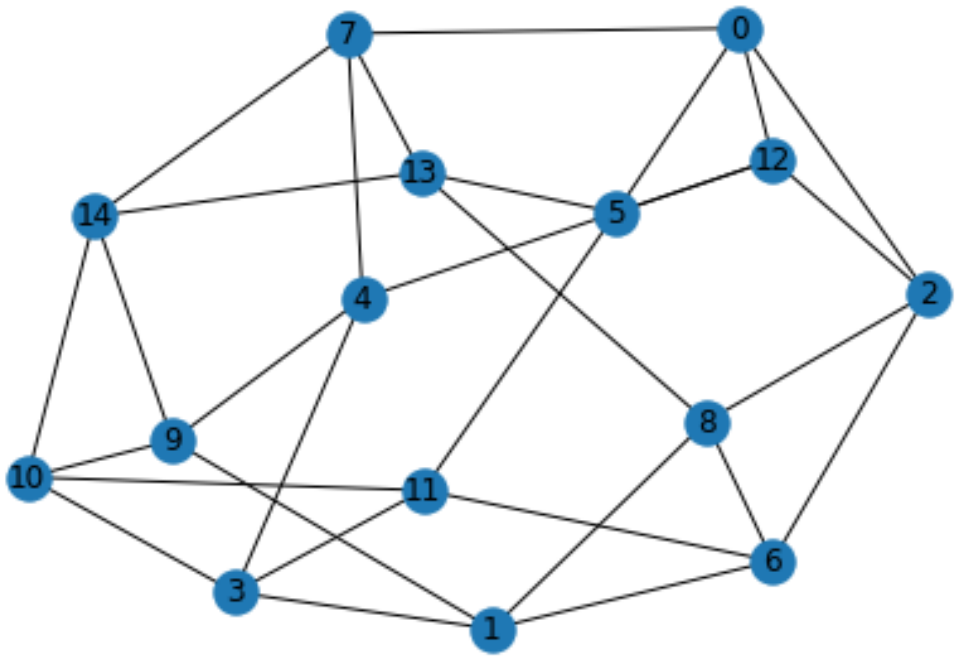
Create a k-regular undirected graph (each node has fixed degree k). Draw the graph for k = 4 and number of nodes n = 15 using draw() function and print the adjacency matrix.

[5]:
```
# Create an Regular Undirected Multi Graph Datatype
G1 = nx.random_regular_graph(4, 15, seed=10)

# Create a Centrality class instance for G1
C1 = Centrality(G1)
```

[6]:
```
# Edges List
G1.edges
```

[6]:
```
EdgeView([(3, 4), (3, 10), (3, 1), (3, 11), (4, 9), (4, 12), (4, 7), (9, 14),
(9, 1), (9, 10), (12, 5), (12, 0), (12, 2), (10, 11), (10, 14), (5, 13), (5, 0),
(5, 11), (13, 14), (13, 7), (13, 8), (0, 2), (0, 7), (2, 8), (2, 6), (14, 7),
(1, 6), (1, 8), (6, 11), (6, 8)])
```

[7]:
```
# Visualisation
nx.draw(G1, with_labels=True)
```



[8]:
```
# Verification
print("Is G1 regular: ", nx.algorithms.regular.is_k_regular(G1, k=4))
```

```
Is G1 regular:  True
```

[9]:
```
# Adjacency Matrix
print(nx.adjacency_matrix(G1,nodelist=[x for x in range(15)]).todense())
```

```
[[0 0 1 0 0 1 0 1 0 0 0 0 1 0 0]
 [0 0 0 1 0 0 1 0 1 1 0 0 0 0 0]
 [1 0 0 0 0 0 1 0 1 0 0 0 1 0 0]
 [0 1 0 0 1 0 0 0 0 0 1 1 0 0 0]
 [0 0 0 1 0 0 0 1 0 1 0 0 1 0 0]
 [1 0 0 0 0 0 0 0 0 0 0 1 1 1 0]
 [0 1 1 0 0 0 0 0 1 0 0 1 0 0 0]
 [1 0 0 0 1 0 0 0 0 0 0 0 0 1 1]
 [0 1 1 0 0 0 1 0 0 0 0 0 0 1 0]
 [0 1 0 0 1 0 0 0 0 0 1 0 0 0 1]
 [0 0 0 1 0 0 0 0 0 1 0 1 0 0 1]
 [0 0 0 1 0 1 1 0 0 0 1 0 0 0 0]
 [1 0 1 0 1 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 0 1 1 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 1 0 1 1 0 0 1 0]]
```

[10]:
```
# Print All Centrality Measures in descending order
C1.centrality_measures()

# Print the nodes which are most similar
C1.cosine_sim()
```

```
Node ranks based on Eigen Vector Centrality:
Node    Centrality
3  :  0.25819888974716115
4  :  0.25819888974716115
9  :  0.25819888974716115
12  :  0.25819888974716115
10  :  0.25819888974716115
5  :  0.25819888974716115
13  :  0.25819888974716115
0  :  0.25819888974716115
2  :  0.25819888974716115
14  :  0.25819888974716115
1  :  0.25819888974716115
6  :  0.25819888974716115
8  :  0.25819888974716115
```

```
11  :  0.25819888974716115
7  :  0.25819888974716115

Node ranks based on Katz Centrality:
Node    Centrality
3  :  0.2581988897471611
4  :  0.2581988897471611
9  :  0.2581988897471611
12  :  0.2581988897471611
10  :  0.2581988897471611
5  :  0.2581988897471611
13  :  0.2581988897471611
0  :  0.2581988897471611
2  :  0.2581988897471611
14  :  0.2581988897471611
1  :  0.2581988897471611
6  :  0.2581988897471611
8  :  0.2581988897471611
11  :  0.2581988897471611
7  :  0.2581988897471611

Node ranks based on Page rank Centrality:
Node    Centrality
3  :  0.06666666666666667
4  :  0.06666666666666667
9  :  0.06666666666666667
12  :  0.06666666666666667
10  :  0.06666666666666667
5  :  0.06666666666666667
13  :  0.06666666666666667
0  :  0.06666666666666667
2  :  0.06666666666666667
14  :  0.06666666666666667
1  :  0.06666666666666667
6  :  0.06666666666666667
8  :  0.06666666666666667
11  :  0.06666666666666667
7  :  0.06666666666666667

Node ranks based on Betweenness Centrality:
Node    Centrality
0  :  0.04395604395604396
10  :  0.049450549450549455
12  :  0.05586080586080585
3  :  0.05952380952380952
9  :  0.059523809523809534
14  :  0.06227106227106227
6  :  0.06776556776556776
```

```
2  :  0.07142857142857144
7  :  0.07142857142857144
5  :  0.07326007326007326
8  :  0.07692307692307691
1  :  0.07783882783882785
13  :  0.08516483516483517
11  :  0.09890109890109891
4  :  0.10164835164835166

Most similar nodes are:  0 2  with similarity:  0.75
```

4 Problem 2

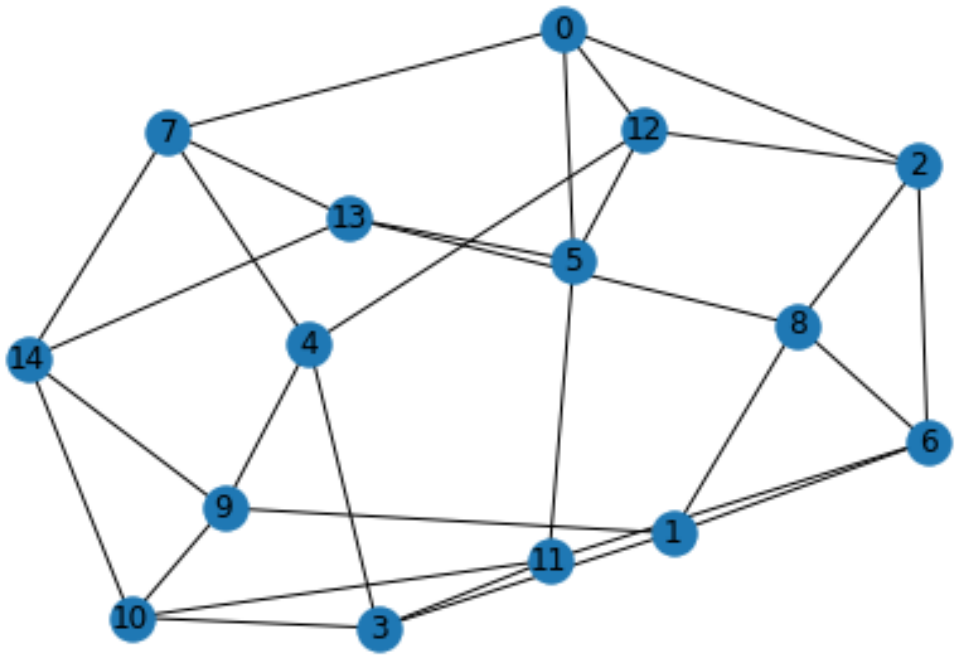
Add node attributes like node ID and color. Show the graph attributes using the function G.nodes.data() function.

[11]:
```
# Initialise another Undirected Graph Data type
G2 = nx.Graph()

# Create a Centrality class instance for G2
C2 = Centrality(G2)
```

[12]:
```
# Add Node attributes
G2.add_nodes_from([
    (0, {"nodeID": "Lioniel Messi", "color":"Black"}),
    (1, {"nodeID": "Neymar", "color":"White"}),
    (2, {"nodeID": "Sergio Ramos", "color":"Gray"}),
    (3, {"nodeID": "Marquinhos", "color":"Silver"}),
    (4, {"nodeID": "Kimpembe", "color":"Maroon"}),
    (5, {"nodeID": "Kurzawa", "color":"Red"}),
    (6, {"nodeID": "Verratti", "color":"Purple"}),
    (7, {"nodeID": "Draxier", "color":"Green"}),
    (8, {"nodeID": "Mbappe", "color":"Lime"}),
    (9, {"nodeID": "Di Maria", "color":"Olive"}),
    (10, {"nodeID": "Draxier", "color":"Yellow"}),
    (11, {"nodeID": "Bernat", "color":"Navy"}),
    (12, {"nodeID": "Diallo", "color":"Blue"}),
    (13, {"nodeID": "Icardi", "color":"Teal"}),
    (14, {"nodeID": "Keylor Navas", "color":"Aqua"}),
])
```

[13]:
```
# Use same nodes from G1
G2.add_edges_from(G1.edges)
```

[14]:
```
# Visualisation
nx.draw(G2, with_labels = True)
```



[15]:
```
# Node data
G2.nodes.data()
```

[15]:
```
NodeDataView({0: {'nodeID': 'Lioniel Messi', 'color': 'Black'}, 1: {'nodeID':
'Neymar', 'color': 'White'}, 2: {'nodeID': 'Sergio Ramos', 'color': 'Gray'}, 3:
{'nodeID': 'Marquinhos', 'color': 'Silver'}, 4: {'nodeID': 'Kimpembe', 'color':
'Maroon'}, 5: {'nodeID': 'Kurzawa', 'color': 'Red'}, 6: {'nodeID': 'Verratti',
'color': 'Purple'}, 7: {'nodeID': 'Draxier', 'color': 'Green'}, 8: {'nodeID':
'Mbappe', 'color': 'Lime'}, 9: {'nodeID': 'Di Maria', 'color': 'Olive'}, 10:
{'nodeID': 'Draxier', 'color': 'Yellow'}, 11: {'nodeID': 'Bernat', 'color':
'Navy'}, 12: {'nodeID': 'Diallo', 'color': 'Blue'}, 13: {'nodeID': 'Icardi',
'color': 'Teal'}, 14: {'nodeID': 'Keylor Navas', 'color': 'Aqua'}})
```

[16]:
```
# Print All Centrality Measures in descending order
C2.centrality_measures()

# Print the nodes which are most similar
C2.cosine_sim()
```

```
Node ranks based on Eigen Vector Centrality:
Node    Centrality
0   :  0.25819888974716115
1   :  0.25819888974716115
```

```
2  :  0.25819888974716115
3  :  0.25819888974716115
4  :  0.25819888974716115
5  :  0.25819888974716115
6  :  0.25819888974716115
7  :  0.25819888974716115
8  :  0.25819888974716115
9  :  0.25819888974716115
10  :  0.25819888974716115
11  :  0.25819888974716115
12  :  0.25819888974716115
13  :  0.25819888974716115
14  :  0.25819888974716115

Node ranks based on Katz Centrality:
Node    Centrality
0  :  0.2581988897471611
1  :  0.2581988897471611
2  :  0.2581988897471611
3  :  0.2581988897471611
4  :  0.2581988897471611
5  :  0.2581988897471611
6  :  0.2581988897471611
7  :  0.2581988897471611
8  :  0.2581988897471611
9  :  0.2581988897471611
10  :  0.2581988897471611
11  :  0.2581988897471611
12  :  0.2581988897471611
13  :  0.2581988897471611
14  :  0.2581988897471611

Node ranks based on Page rank Centrality:
Node    Centrality
0  :  0.06666666666666667
1  :  0.06666666666666667
2  :  0.06666666666666667
3  :  0.06666666666666667
4  :  0.06666666666666667
5  :  0.06666666666666667
6  :  0.06666666666666667
7  :  0.06666666666666667
8  :  0.06666666666666667
9  :  0.06666666666666667
10  :  0.06666666666666667
11  :  0.06666666666666667
12  :  0.06666666666666667
13  :  0.06666666666666667
```

```
14  :  0.06666666666666667

Node ranks based on Betweenness Centrality:
Node    Centrality
0  :  0.04395604395604396
10  :  0.049450549450549455
12  :  0.055860805860805864
9  :  0.05952380952380952
3  :  0.059523809523809534
14  :  0.06227106227106227
6  :  0.06776556776556776
2  :  0.07142857142857144
7  :  0.07142857142857144
5  :  0.07326007326007326
8  :  0.07692307692307693
1  :  0.07783882783882784
13  :  0.08516483516483517
11  :  0.09890109890109888
4  :  0.10164835164835166

Most similar nodes are:  3 9  with similarity:  0.75
```

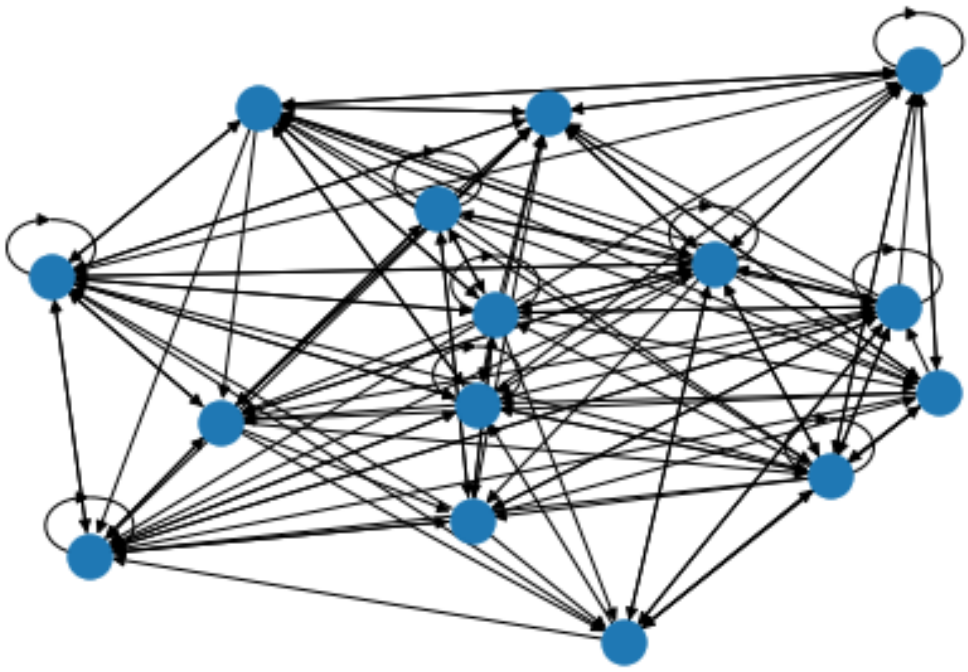
5 Problem 3

Create a directed graph of n = 15 nodes and random directed edges, where the probability of an edge from node i to j is 0.6. Draw the graph using draw() function.

[17]:
```
# Initialise a Directed Graph Datatype
G3 = nx.DiGraph()

# Create a Centrality class instance for G3
C3 = Centrality(G3)

# Add 15 nodes
G3.add_nodes_from(range(15))
```

[18]:
```
import random

# Define a threshold
threshold = 0.6

# Add node if random number is less than threshold
for i in range(15):
    for j in range(15):
        if(random.random() <= threshold):
            G3.add_edge(i, j)
```

[19]:
```
# Visualisation
nx.draw(G3)
```



[20]:
```
# Print All Centrality Measures in descending order
C3.centrality_measures()

# Print the nodes which are most similar
C3.cosine_sim()
```

```
Node ranks based on Eigen Vector Centrality:
Node    Centrality
1  :  0.12088424420862345
11  :  0.17904960298638542
7  :  0.18457272148380047
8  :  0.19360267504530654
14  :  0.2152850097346663
6  :  0.25438653639418496
13  :  0.25631523224510794
5  :  0.25701599631692695
3  :  0.26121815456399083
12  :  0.2712983165083896
9  :  0.27518333069992973
2  :  0.3206061781823563
```

```
4  :  0.32084615690891183
10  :  0.322097977434518
0  :  0.33454900484776257

Node ranks based on Katz Centrality:
Node    Centrality
1  :  0.1336144585481042
11  :  0.1870762634121063
7  :  0.1917947023290831
8  :  0.19940336807569503
14  :  0.21974253050713743
13  :  0.25633892162119293
5  :  0.257158248079177
6  :  0.2572307968345597
3  :  0.2626888818701953
12  :  0.2711098845482896
9  :  0.27453966309552563
4  :  0.3147393685279149
2  :  0.31499100924804785
10  :  0.3161339988386179
0  :  0.3289518372734584

Node ranks based on Page rank Centrality:
Node    Centrality
1  :  0.033634904305924154
7  :  0.04584472010637659
11  :  0.04795752244942557
8  :  0.05030136363990969
14  :  0.053835658196216475
6  :  0.06666980432192723
5  :  0.06699008936541842
3  :  0.06748148291861181
13  :  0.07080637558966973
9  :  0.07109081151001054
12  :  0.07383424720192697
10  :  0.08495484751164287
2  :  0.08498149934394172
4  :  0.08976142957799962
0  :  0.09185524396099842

Node ranks based on Betweenness Centrality:
Node    Centrality
1  :  0.006471306471306472
7  :  0.007463568177853893
5  :  0.013505145648002793
2  :  0.016104332175760745
8  :  0.020301523872952445
11  :  0.022231141873999015
```

```
9  :  0.02360476824762539
3  :  0.024801785516071234
0  :  0.024851735566021285
4  :  0.03143443857729572
6  :  0.03441717013145584
14  :  0.0348227169655741
12  :  0.05027710384853242
13  :  0.05427588284731142
10  :  0.08598683062968776

Most similar nodes are:  10 14  with similarity:  0.9285714285714285
```

6 Problem 4

Create an undirected bipartite graph with 10 nodes in 1st layer and 5 in another. Create random edges between nodes of 2 layers, where the probability of an edge appearing between node i and j is 0.5. Draw the graph using draw() function.

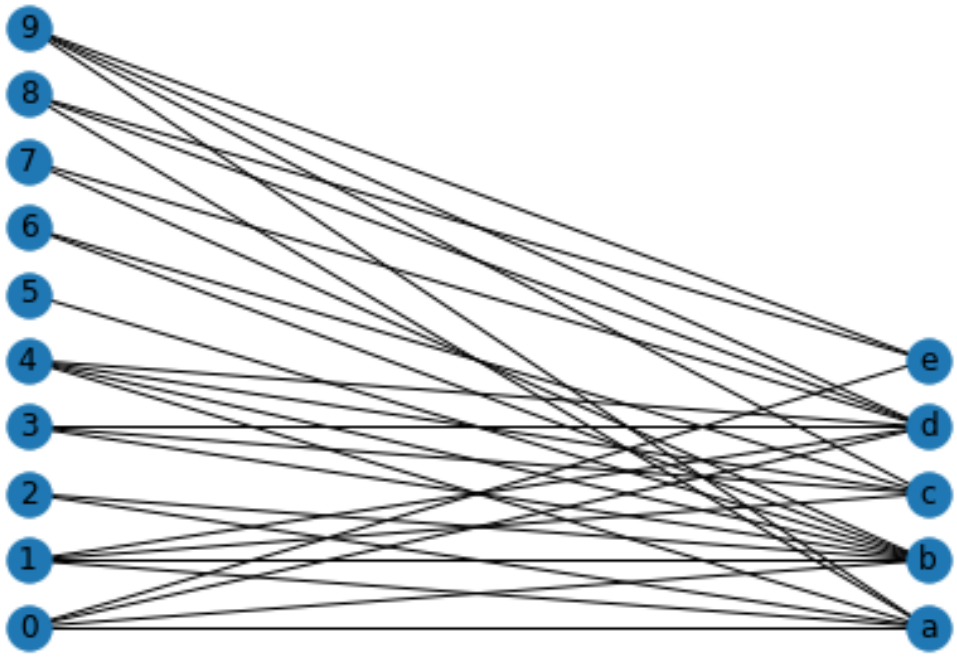
[21]:
```
from networkx.algorithms import bipartite

# Initialise a Graph Datatype
G4 = nx.Graph()

# Create a Centrality class instance for G4
C4 = Centrality(G4)
```

[22]:
```
# Define Actor and Group nodes
actors = range(10)
groups = ['a', 'b', 'c', 'd', 'e']

# Add nodes to the each group respectively
G4.add_nodes_from(actors, bipartite = 0)
G4.add_nodes_from(groups, bipartite = 1)
```

[23]:
```
# Define threshold
threshold = 0.5

# Add node if random number is less than threshold
for i in actors:
    for j in groups:
        if(random.random() <= threshold):
            G4.add_edge(i, j)
```

[24]:
```
pos = {}

# Update position for node from each group
pos.update((node, (1, index)) for index, node in enumerate(actors))
```

```
pos.update((node, (2, index)) for index, node in enumerate(groups))

nx.draw(G4, pos=pos, with_labels = True)
```



[25]:
```
# Verification
bipartite.is_bipartite(G4)
```

[25]: True

[26]:
```
# Print All Centrality Measures in descending order
C4.centrality_measures()

# Print the nodes which are most similar
C4.cosine_sim()
```

```
Node ranks based on Eigen Vector Centrality:
Node    Centrality
5  :  0.0824580046853728
6  :  0.14371526336637708
2  :  0.15647705269577109
e  :  0.16289303772934718
7  :  0.16843082424695885
8  :  0.19622200655356983
```

```
3  :  0.2296880829279631
9  :  0.25747926523457415
c  :  0.27541657158376787
0  :  0.27868001123894265
1  :  0.3037071309383614
4  :  0.3037071309383614
a  :  0.3327944010368205
b  :  0.3707364862503027
d  :  0.38653932525568374

Node ranks based on Katz Centrality:
Node    Centrality
5  :  0.1818674606694596
6  :  0.20925046303763978
2  :  0.2118590973396202
7  :  0.21430034097896308
e  :  0.22439557310955516
8  :  0.23320684657617255
3  :  0.2416833433471432
9  :  0.2605898489443526
0  :  0.26673153079679013
1  :  0.2716749800173038
4  :  0.2716749800173038
c  :  0.2738300940738805
a  :  0.2999164517585722
d  :  0.3243289018883399
b  :  0.33524693696049335

Node ranks based on Page rank Centrality:
Node    Centrality
5  :  0.024628452887827935
7  :  0.03835608889768244
2  :  0.03854257249716979
6  :  0.0389065623122681
3  :  0.05263419832212261
8  :  0.05281526793143132
e  :  0.053552815596106404
1  :  0.06654831793146447
4  :  0.06654831793146447
9  :  0.06709337735587148
0  :  0.06744372081925926
c  :  0.08398761684961972
a  :  0.09821580034826155
d  :  0.11304935311640935
b  :  0.13767753720304116

Node ranks based on Betweenness Centrality:
Node    Centrality
```

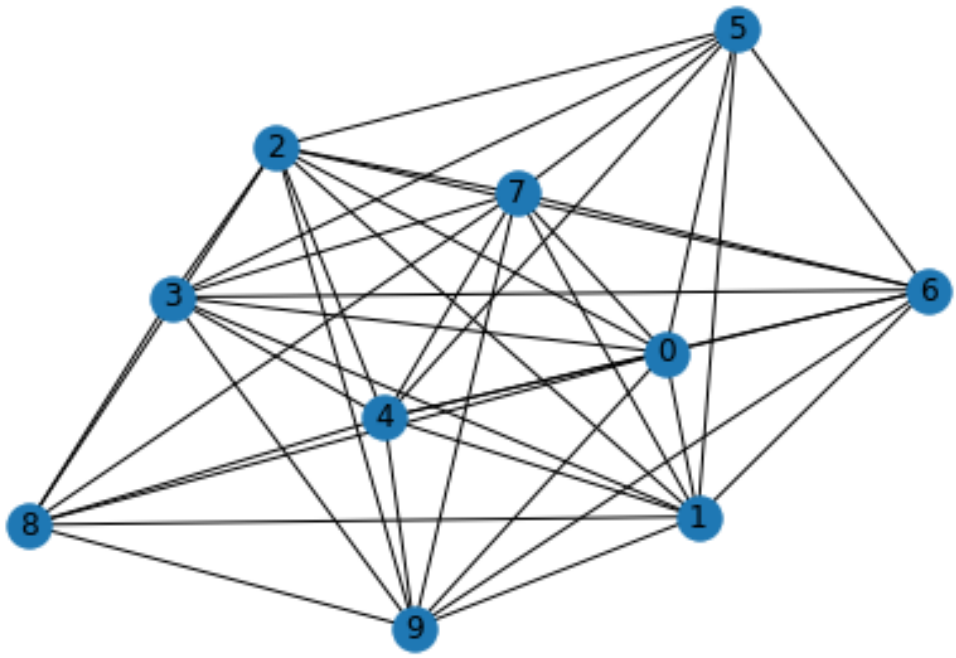
```
5  :  0.0
6  :  0.010923600209314497
7  :  0.01109366823652538
2  :  0.01357927786499215
e  :  0.018267050409907552
8  :  0.025946275946275948
3  :  0.03025902668759812
1  :  0.0598901098901099
4  :  0.0598901098901099
9  :  0.06913483342054771
c  :  0.08431013431013432
0  :  0.09290947148090005
a  :  0.13073870573870575
d  :  0.1853828710971568
b  :  0.3395429966858539

Most similar nodes are:  1 4  with similarity:  1.0
```

7 Problem 5

Create an one-mode projection on layer 1 and draw the graph using draw() function

[27]:
```
# Make a projected graph on actors
G5 = bipartite.projected_graph(G4, nodes = actors)

# Create a Centrality class instance for G5
C5 = Centrality(G5)
```

[28]:
```
# Visualisation
nx.draw(G5, with_labels = True)
```



[29]:
```
# Print All Centrality Measures in descending order
C5.centrality_measures()

# Print the nodes which are most similar
C5.cosine_sim()
```

```
Node ranks based on Eigen Vector Centrality:
Node    Centrality
5  :  0.2722877158720702
8  :  0.2722877158720702
6  :  0.3044675148368999
9  :  0.3044675148368999
0  :  0.3332461213548794
1  :  0.3332461213548794
2  :  0.3332461213548794
3  :  0.3332461213548794
4  :  0.3332461213548794
7  :  0.33324612135487947

Node ranks based on Katz Centrality:
Node    Centrality
5  :  0.27810428767410444
8  :  0.27810428767410444
```

```
6  :  0.3059147124606197
9  :  0.30591471246061974
0  :  0.3311969171944969
1  :  0.3311969171944969
2  :  0.3311969171944969
3  :  0.3311969171944969
4  :  0.3311969171944969
7  :  0.3311969171944969

Node ranks based on Page rank Centrality:
Node    Centrality
5  :  0.085399321223249
8  :  0.085399321223249
6  :  0.09576917581352022
9  :  0.09576917581352022
7  :  0.1062771676544102
0  :  0.10627716765441021
1  :  0.10627716765441021
2  :  0.10627716765441021
3  :  0.10627716765441021
4  :  0.10627716765441021

Node ranks based on Betweenness Centrality:
Node    Centrality
5  :  0.0
8  :  0.0
6  :  0.003968253968253968
9  :  0.003968253968253968
0  :  0.012566137566137564
1  :  0.012566137566137564
2  :  0.012566137566137564
3  :  0.012566137566137564
4  :  0.012566137566137564
7  :  0.012566137566137564

Most similar nodes are:  5 9  with similarity:  0.9354143466934851
```

[ ]: