



Complex Network

Home work2

Dr.Rahmani

Sara Sadat Younesi – 98533053

Question :

 **Download two networks :**

1. Search for network datasets

2. One of them: at least 1000 nodes

Answer :

email-Eu-core network

Dataset information

The network was generated using email data from a large European research institution. We have anonymized information about all incoming and outgoing email between members of the research institution. There is an edge (u, v) in the network if person u sent person v at least one email. The e-mails only represent communication between institution members (the core), and the dataset does not contain incoming messages from or outgoing messages to the rest of the world.

The dataset also contains "ground-truth" community memberships of the nodes. Each individual belongs to exactly one of 42 departments at the research institute.

This network represents the "core" of the [email-EuAll](#) network, which also contains links between members of the institution and people outside of the institution (although the node IDs are not the same).

Dataset statistics	
Nodes	1005
Edges	25571
Nodes in largest WCC	986 (0.981)
Edges in largest WCC	25552 (0.999)
Nodes in largest SCC	803 (0.799)
Edges in largest SCC	24729 (0.967)
Average clustering coefficient	0.3994
Number of triangles	105461
Fraction of closed triangles	0.1085
Diameter (longest shortest path)	7
90-percentile effective diameter	2.9

Source (citation)

Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. "Local Higher-order Graph Clustering." In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2017.

J. Leskovec, J. Kleinberg and C. Faloutsos. [Graph Evolution: Densification and Shrinking Diameters](#). ACM Transactions on Knowledge Discovery from Data (ACM TKDD), 1(1), 2007.

Files

File	Description
email-Eu-core.txt.gz	Email communication links between members of the institution
email-Eu-core-department-labels.txt.gz	Department membership labels

Data format for community membership

NODEID DEPARTMENT

- NODEID: id of the node (a member of the institute)
- DEPARTMENT: id of the member's department (number in 0, 1, ..., 41)

2. Social circles: Facebook

Dataset information

This dataset consists of 'circles' (or 'friends lists') from Facebook. Facebook data was collected from survey participants using this [Facebook app](#). The dataset includes node features (profiles), circles, and ego networks.

Facebook data has been anonymized by replacing the Facebook-internal ids for each user with a new value. Also, while feature vectors from this dataset have been provided, the interpretation of those features has been obscured. For instance, where the original dataset may have contained a feature "political=Democratic Party", the new data would simply contain "political=anonymized feature 1". Thus, using the anonymized data it is possible to determine whether two users have the same political affiliations, but not what their individual political affiliations represent.

Data is also available from [Google+](#) and [Twitter](#).

Note that these statistics were compiled by combining the ego-networks, including the ego nodes themselves (along with an edge to each of their friends).

Dataset statistics	
Nodes	4039
Edges	88234
Nodes in largest WCC	4039 (1.000)
Edges in largest WCC	88234 (1.000)
Nodes in largest SCC	4039 (1.000)
Edges in largest SCC	88234 (1.000)
Average clustering coefficient	0.6055
Number of triangles	1612010
Fraction of closed triangles	0.2647
Diameter (longest shortest path)	8
90-percentile effective diameter	4.7

Source (citation)

J. McAuley and J. Leskovec. [Learning to Discover Social Circles in Ego Networks](#). NIPS, 2012.

Files

File	Description
facebook.tar.gz	Facebook data (10 networks, anonymized)
facebook_combined.txt.gz	Edges from all egonets combined
readme-Ego.txt	Description of files

Files:

nodeId.edges : The edges in the ego network for the node 'nodeId'. Edges are undirected for facebook, and directed (a follows b) for twitter and gplus. The 'ego' node does not appear, but it is assumed that they follow every node id that appears in this file.

nodeId.circles : The set of circles for the ego node. Each line contains one circle, consisting of a series of node ids. The first entry in each line is the name of the circle.

nodeId.feats : The features for each of the nodes that appears in the edge file.

nodeId.ego_feats : The features for the ego user.

nodeId.feats_names : The names of each of the feature dimensions. Features are '1' if the user has this property in their profile, and '0' otherwise. This file has been anonymized for facebook users, since the names of the features would reveal private data.



Prepare one network of known nodes and edges

E.g., network of cities, people, software, web-pages –

at least 10 nodes – You should prepare this network yourself

- Important: mention your references

I had two datasets of Facebook and email, but for better intuition I checked the metrics on the networks I built with the lowest number of nodes.

I use network library to do this homework.

Create an empty graph with no nodes and no edges.

```
import networkx as nx
>>> G = nx.DiGraph()
```

I make directed Graph.

By definition, a [Graph](#) is a collection of nodes (vertices) along with identified pairs of nodes (called edges, links, etc). In NetworkX, nodes can be any [hashable](#) object e.g., a text string, an image, an XML object, another Graph, a customized node object, etc.

The graph G can be grown in several ways. NetworkX includes many [graph generator functions](#) and [facilities to read and write graphs in many formats](#). To get started though we'll look at simple manipulations. You can add one node at a time.

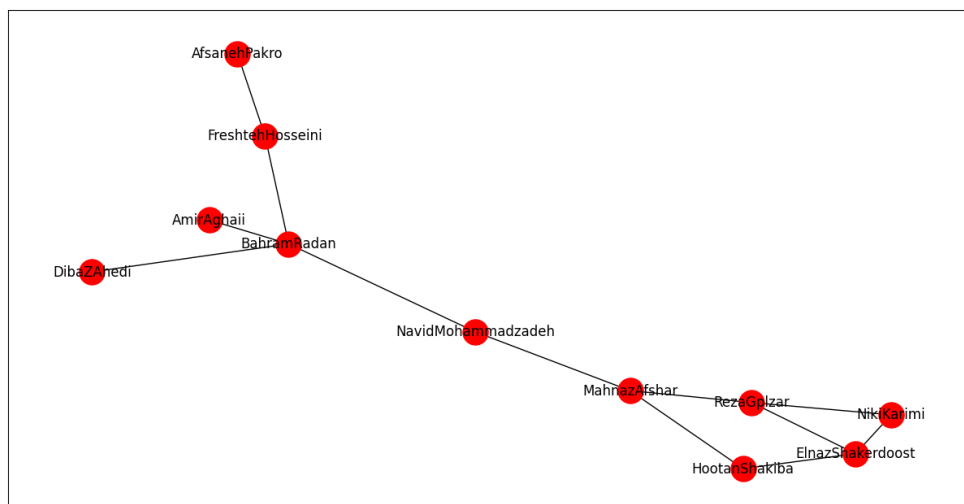
The graph G now contains H as a node. This flexibility is very powerful as it allows graphs of graphs, graphs of files, graphs of functions and much more. It is worth thinking about how to structure your application so that the nodes are useful entities. Of course you can always use a unique identifier in G and have a separate dictionary keyed by identifier to the node information if you prefer.

1.View Of network Iranian Actor and Actress

Node: Name of Actor and Actress

Edge : Relationship between Actor and Actress in films

Weight : Number Of films

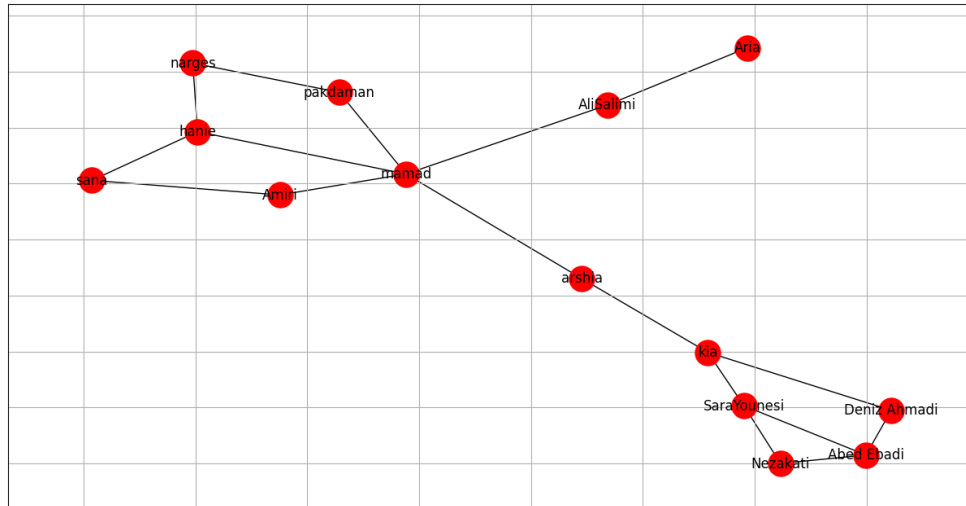


2.View Of network IUST Student

Node: Name of Student

Edge : Relationship between Student

Weight : Intimacy students

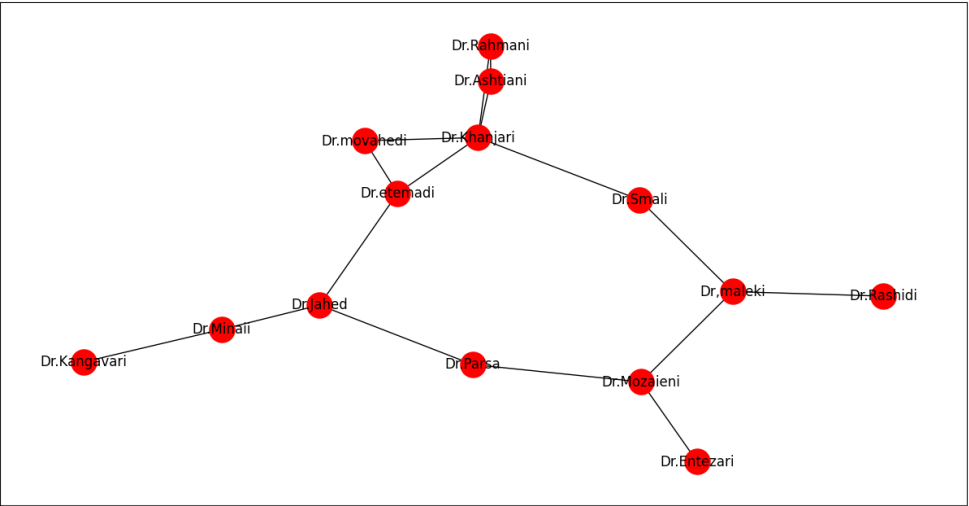


3.View Of network IUST Professors

Node: Name of Professor

Edge : Relationship between Professors

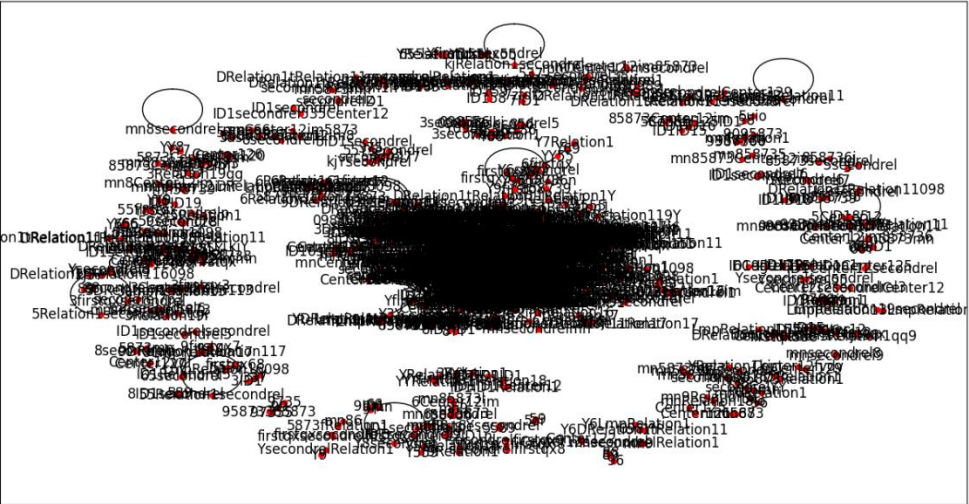
Weight : Number of articles worked together



4..View Of network Facebook Relations

Node: Name of person in facebook

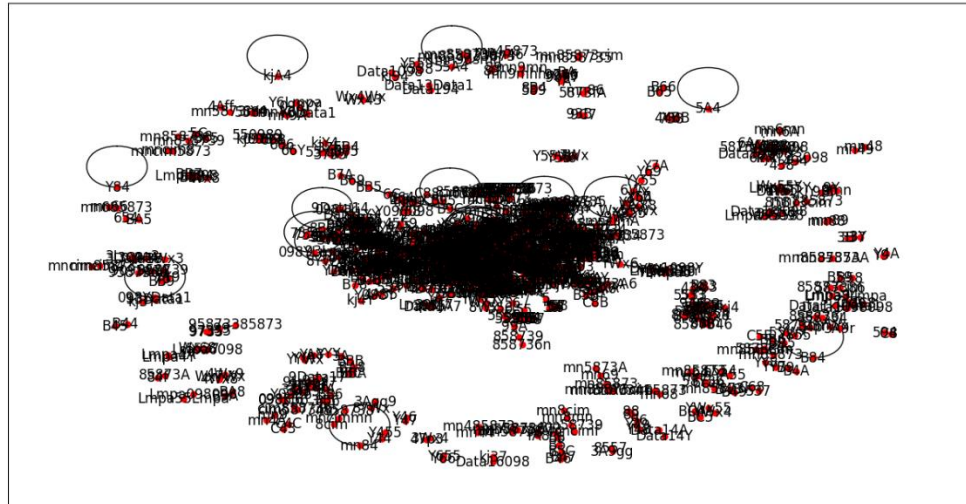
Edge : Relationship humans



5.View Of network Email Core

Node: Name of Email Center

Edge : Emails sent to each other



2- Compute Some Metrics of These

1. # Nodes

```
len(G.nodes)
```

2. # Edges

```
len(G.edges)
```

3. Average degree

```
sum(dict(G.degree).values()) / len(G.nodes)
```

4. Density

```
((len(G.edges) / (len(G.nodes) * (len(G.nodes) - 1))))
```

Returns the density of a graph.

The density for undirected graphs is

$$d = \frac{2m}{n(n-1)},$$

and for directed graphs is

$$d = \frac{m}{n(n-1)},$$

where n is the number of nodes and m is the number of edges in G.

Notes

The density is 0 for a graph without edges and 1 for a complete graph. The density of multigraphs can be higher than 1.

Self loops are counted in the total number of edges so graphs with self loops can have density higher than 1.

5. Clustering coefficient 1

```
nx.clustering(G)
```

```
clustering(G, nodes=None, weight=None)\[source\]
```

Compute the clustering coefficient for nodes.

For unweighted graphs, the clustering of a node is the fraction of possible triangles through that node that exist,

where $T(u)$ is the number of triangles through node u and $\deg(u)$ is the degree of u .

For weighted graphs, there are several ways to define clustering [\[1\]](#). the one used here is defined as the geometric average of the subgraph edge weights [\[2\]](#),

The edge weights W are normalized by the maximum weight in the network $W=w/\max(w)$.

The value of C_u is assigned to 0 if $\deg(u)<2$.

Additionally, this weighted definition has been generalized to support negative edge weights [\[3\]](#).

For directed graphs, the clustering is similarly defined as the fraction of all possible directed triangles or geometric average of the subgraph edge weights for unweighted and weighted directed graph respectively [\[4\]](#).

Parameters:

G : graph

Nodes: node, iterable of nodes, or None (default=None)

If a singleton node, return the number of triangles for that node. If an iterable, compute the number of triangles for each of those nodes. If [None](#) (the default) compute the number of triangles for all nodes in G.

Weight: string or None, optional (default=None)

The edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1.

Returns:

Out: float, or dictionary

Clustering coefficient at specified nodes

6. Clustering coefficient 2

```
list(nx.triangles(G, (0, 1)).values())
```

triangles(G, nodes=None)[\[source\]](#)

Compute the number of triangles.

Finds the number of triangles that include a node as one vertex.

Parameters:

G: graph

A networkx graph

Nodes: container of nodes, optional (default= all nodes in G)

Compute triangles for nodes in this container.

Returns:

Out: dictionary

Number of triangles keyed by node label.

7. Diameter

```
max([max(j.values())  
     for (i, j) in nx.shortest_path_length(G)]))
```

8. Average shortest path length

```
(nx.average_shortest_path_length(G))
```

`average_shortest_path_length(G, weight=None, method=None)`

[\[source\]](#)

Returns the average shortest path length.

The average shortest path length is

$$\alpha = \sum_{\substack{s, t \in V \\ s \neq t}} \frac{d(s, t)}{n(n-1)}$$

where V is the set of nodes in G , $d(s, t)$ is the shortest path from s to t , and n is the number of nodes in G .

Parameters:

G: NetworkX graph

Weight: None, string or function, optional (default = None)

If None, every edge has weight/distance/cost 1. If a string, use this edge attribute as the edge weight. Any edge attribute not present defaults to 1. If this is a function, the weight of an edge is the value returned by the function. The function must accept exactly three positional arguments: the two endpoints of an edge and the dictionary of edge attributes for that edge. The function must return a number.

methodstring, optional (default = 'unweighted' or 'dijkstra')

The algorithm to use to compute the path lengths. Supported options are 'unweighted', 'dijkstra', 'bellman-ford', 'floyd-warshall' and 'floyd-warshall-numpy'. Other method values produce a ValueError. The default method is 'unweighted' if weight is None, otherwise the default method is 'dijkstra'.

Raises:

NetworkXPointlessConcept

If G is the null graph (that is, the graph on zero nodes).

NetworkXError

If G is not connected (or not strongly connected, in the case of a directed graph).

ValueError

If method is not among the supported options.

9. Plot degree distribution

```
def plot_degree_dist(G):  
    degrees = [G.degree(n) for n in G.nodes()]  
    plt.hist(degrees)  
    plt.show()
```

```
plot_degree_dist(G)
```

10. Assortativity (Degree Correlation)

```
nx.degree_assortativity_coefficient(G)
```

```
degree_assortativity_coefficient(G, x='out', y='in', weight=None, nodes=None)  
\[source\]
```

Compute degree assortativity of graph.

Assortativity measures the similarity of connections in the graph with respect to the node degree.

Parameters:

G : *NetworkX graph*

x: *string ('in','out')*

The degree type for source node (directed graphs only).

y: *string ('in','out')*

The degree type for target node (directed graphs only).

weight: *string or None, optional (default=None)*

The edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1. The degree is the sum of the edge weights adjacent to the node.

nodes: *list or iterable (optional)*

Compute degree assortativity only for nodes in container. The default is all nodes.

Returns:

r : *float*

Assortativity of graph by degree.

11. Find top 5 nodes based on 3 different centrality measures

```
nx.degree_centrality(G)
```

degree centrality(*G*)

[\[source\]](#)

Compute the degree centrality for nodes.

The degree centrality for a node *v* is the fraction of nodes it is connected to.

Parameters:

***G* : graph**

A networkx graph

Returns:

nodes : dictionary

Dictionary of nodes with degree centrality as the value.

12. Network Centralization

nx.eigenvector centrality(*G*)

eigenvector centrality(*G*, *max_iter*=100, *tol*=1e-06, *nstart*=None, *weight*=None)

Compute the eigenvector centrality for the graph *G*.

[\[source\]](#)

Eigenvector centrality computes the centrality for a node based on the centrality of its neighbors.

The eigenvector centrality for node *i* is the *i*-th element of the vector *x* defined by the equation

$$Ax = \lambda x$$

where *A* is the adjacency matrix of the graph *G* with eigenvalue λ . By virtue of the Perron–Frobenius theorem, there is a unique solution *x*, all of whose entries are positive, if λ is the largest eigenvalue of the adjacency matrix *A* ([2]).

Parameters:

***G*graph**

A networkx graph

***max_iter*integer, optional (default=100)**

Maximum number of iterations in power method.

***tol*float, optional (default=1.0e-6)**

Error tolerance used to check convergence in power method iteration.

***nstart*dictionary, optional (default=None)**

Starting value of eigenvector iteration for each node.

***weight*None or string, optional (default=None)**

If None, all edge weights are considered equal. Otherwise holds the name of the edge attribute used as weight. In this measure the weight is interpreted as the connection strength.

Returns:

nodesdictionary

Dictionary of nodes with eigenvector centrality as the value.

Raises:

NetworkXPointlessConcept

If the graph G is the null graph.

NetworkXError

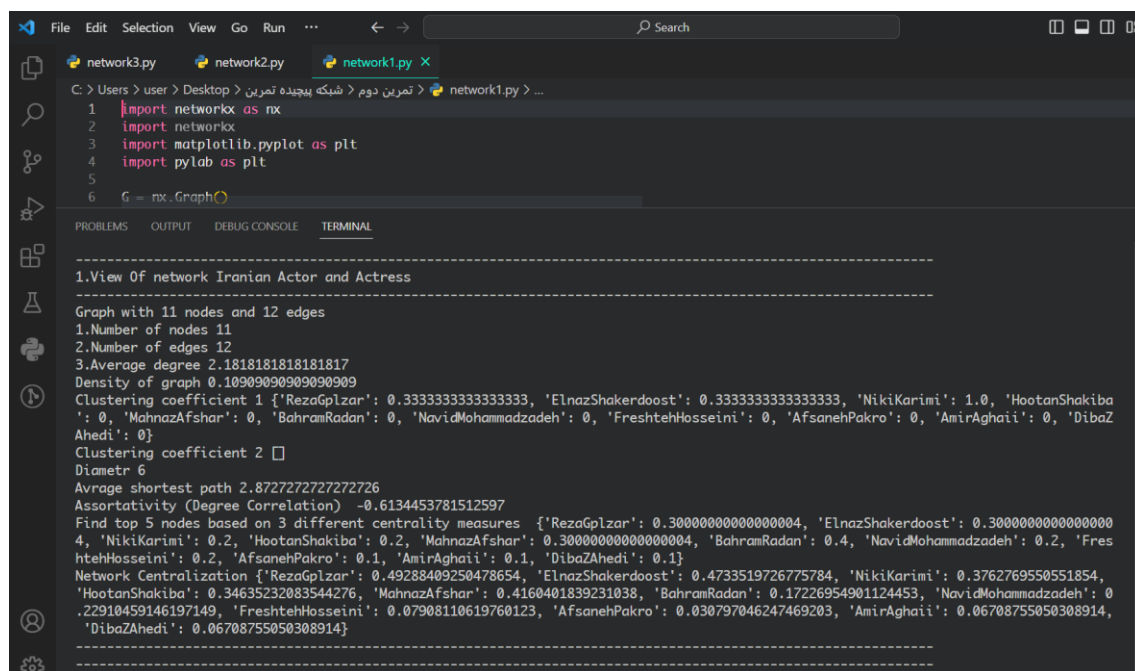
If each value in nstart is zero.

PowerIterationFailedConvergence

If the algorithm fails to converge to the specified tolerance within the specified number of iterations of the power iteration method.

Results Of 5 Networks

1.View Of network Iranian Actor and Actress



```
File Edit Selection View Go Run ... Search
network3.py network2.py network1.py x
C:\Users\user\Desktop> شیکه پیچیده تمرین > تمرین دوم > network1.py > ...
1 import networkx as nx
2 import networkx
3 import matplotlib.pyplot as plt
4 import pylab as plt
5
6 G = nx.Graph()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
-----
1.View Of network Iranian Actor and Actress
-----
Graph with 11 nodes and 12 edges
1.Number of nodes 11
2.Number of edges 12
3.Average degree 2.1818181818181817
Density of graph 0.10909090909090909
Clustering coefficient 1 {'RezaGplzar': 0.3333333333333333, 'ElnazShakerdoost': 0.3333333333333333, 'NikiKarimi': 1.0, 'HootanShakiba': 0, 'MahnazAfshar': 0, 'BahramRadan': 0, 'NavidMohammadzadeh': 0, 'FreshtehHosseini': 0, 'AfsanehPakro': 0, 'AmirAghai': 0, 'DibaZAhedi': 0}
Clustering coefficient 2 □
Diameter 6
Average shortest path 2.8727272727272726
Assortativity (Degree Correlation) -0.6134453781512597
Find top 5 nodes based on 3 different centrality measures {'RezaGplzar': 0.30000000000000004, 'ElnazShakerdoost': 0.30000000000000004, 'NikiKarimi': 0.2, 'HootanShakiba': 0.2, 'MahnazAfshar': 0.30000000000000004, 'BahramRadan': 0.4, 'NavidMohammadzadeh': 0.2, 'FreshtehHosseini': 0.2, 'AfsanehPakro': 0.1, 'AmirAghai': 0.1, 'DibaZAhedi': 0.1}
Network Centralization {'RezaGplzar': 0.49288409250478654, 'ElnazShakerdoost': 0.4733519726775784, 'NikiKarimi': 0.3762769550551854, 'HootanShakiba': 0.34635232083544276, 'MahnazAfshar': 0.4160401839231038, 'BahramRadan': 0.17226954901124453, 'NavidMohammadzadeh': 0.22910459146197149, 'FreshtehHosseini': 0.07908110619760123, 'AfsanehPakro': 0.030797046247469203, 'AmirAghai': 0.06708755050308914, 'DibaZAhedi': 0.06708755050308914}
-----
```

2.View Of network IUST Student

```
File Edit Selection View Go Run ... Search
network3.py network2.py X
C:\Users\user\Desktop> شیکه پیچیده تمرین > network2.py > ...
5
6 G = nx.Graph()
7 G.add_edges_from([('SaraYounesi', 'Abed Ebadi', {'weight': 100}), ('SaraYounesi', 'Nezakati', {'weight': 200}), ('Nezakati', 'Abed Ebadi', {'weight': 100}), ('mamad', 'arshia', {'weight': 100}), ('SaraYounesi', 'kia', {'weight': 400}), ('AliSalimi', 'mamad', {'weight': 100})])
9
10 plt.figure()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\user\Desktop> خود نیرم‌ل‌ی‌ر‌م‌ت ه‌دی‌ج‌ی‌پ و‌ک‌ش‌ی‌ن c:; cd 'c:\Users\user\Desktop\و‌ک‌ش‌ی‌ن'; & 'C:\Users\user\AppData\Local\Programs\Python\Python38\python.exe' 'c:\Users\user\.vscode\extensions\ms-python.python-2023.10.0\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '3310' '--' 'c:\Users\user\Desktop\و‌ک‌ش‌ی‌ن\network2.py'

-----
2.View Of network IUST Student
-----
Graph with 14 nodes and 17 edges
1.Number of nodes 14
2.Number of edges 17
3.Average degree 2.4285714285714284
Density of graph 0.09340659340659341
Clustering coefficient 1 {'SaraYounesi': 0.3333333333333333, 'Abed Ebadi': 0.3333333333333333, 'Nezakati': 1.0, 'Deniz Ahmadi': 0, 'kia': 0, 'mamad': 0, 'arshia': 0, 'AliSalimi': 0, 'Aria': 0, 'Amiri': 0, 'paktaman': 0, 'hanie': 0, 'narges': 0, 'sana': 0}
Clustering coefficient 2 0.0
Diameter 6
Average shortest path 3.065934065934066
Assortativity (Degree Correlation) -0.25925925925925897
Find top 5 nodes based on 3 different centrality measures {'SaraYounesi': 0.23076923076923078, 'Abed Ebadi': 0.23076923076923078, 'Nezakati': 0.15384615384615385, 'Deniz Ahmadi': 0.15384615384615385, 'kia': 0.23076923076923078, 'mamad': 0.38461538461538464, 'arshia': 0.15384615384615385, 'AliSalimi': 0.15384615384615385, 'Aria': 0.07692307692307693, 'Amiri': 0.15384615384615385, 'paktaman': 0.15384615384615385, 'hanie': 0.23076923076923078, 'narges': 0.15384615384615385, 'sana': 0.15384615384615385}
```

3.View Of network IUST Professors

```
File Edit Selection View Go Run ... Search
network3.py X
C:\Users\user\Desktop> شیکه پیچیده تمرین > network3.py > ...
1 import networkx as nx
2 import networkx
3 import matplotlib.pyplot as plt
4 import pylab as plt
5
6 G = nx.Graph()

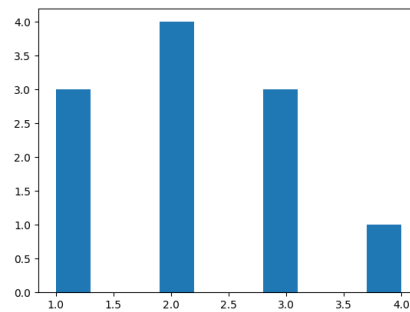
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
-----
Graph with 14 nodes and 16 edges
1.Number of nodes 14
2.Number of edges 16
3.Average degree 2.2857142857142856
Density of graph 0.08791208791208792
Clustering coefficient 1 {'Dr.Rahmani': 1.0, 'Dr.Khanjari': 0.2, 'Dr.Ashtiani': 1.0, 'Dr.etemadi': 0.3333333333333333, 'Dr.Parsa': 0, 'Dr.Mozaieni': 0, 'Dr.Jahed': 1.0, 'Dr.Minaii': 0, 'Dr.maleki': 0, 'Dr.Kangavari': 0, 'Dr.Entezari': 0, 'Dr.Rashidi': 0, 'Dr.Smali': 0}
Clustering coefficient 2 0.0
Diameter 6
Average shortest path 2.857142857142857
Assortativity (Degree Correlation) -0.2857142857142857
Find top 5 nodes based on 3 different centrality measures {'Dr.Rahmani': 0.15384615384615385, 'Dr.Khanjari': 0.38461538461538464, 'Dr.Ashtiani': 0.15384615384615385, 'Dr.etemadi': 0.23076923076923078, 'Dr.Parsa': 0.15384615384615385, 'Dr.Mozaieni': 0.23076923076923078, 'Dr.Jahed': 0.23076923076923078, 'Dr.maleki': 0.15384615384615385, 'Dr.Minaii': 0.15384615384615385, 'Dr.Kangavari': 0.07692307692307693, 'Dr.Entezari': 0.07692307692307693, 'Dr.Rashidi': 0.07692307692307693, 'Dr.Smali': 0.15384615384615385}
Network Centralization {'Dr.Rahmani': 0.3190196967829373, 'Dr.Khanjari': 0.5835156451629239, 'Dr.Ashtiani': 0.3190196967829373, 'Dr.etemadi': 0.40524603495104145, 'Dr.Parsa': 0.1124207555049563, 'Dr.Mozaieni': 0.10458247409859014, 'Dr.Jahed': 0.21346023034462122, 'Dr.maleki': 0.3494986401740873, 'Dr.Minaii': 0.08622633816817148, 'Dr.Kangavari': 0.1464771834261647, 'Dr.Entezari': 0.03047894339121731, 'Dr.Rashidi': 0.03696804755875295, 'Dr.Smali': 0.2580321222960162}
-----
PS C:\Users\user\Desktop> خود نیرم‌ل‌ی‌ر‌م‌ت ه‌دی‌ج‌ی‌پ و‌ک‌ش‌ی‌ن

Ln 1, Col 1 Spaces: 6 UTF-8 CRLF Python 3.8.0 64-bit Go Live linter ready
```

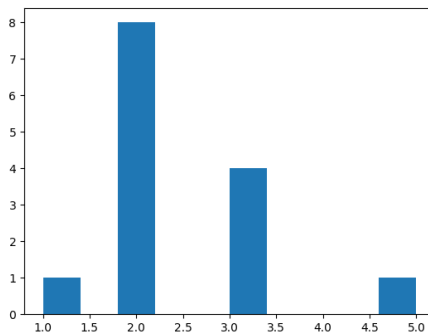
4.View Of network Facebook Relations

Plot degree distribution

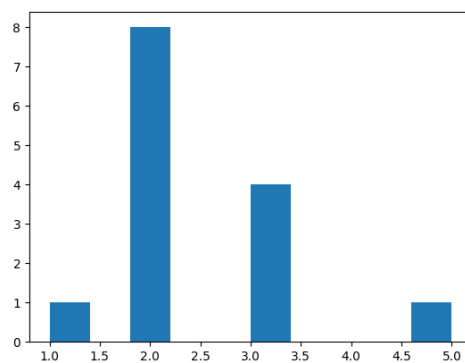
1.View Of network Iranian Actor and Actress



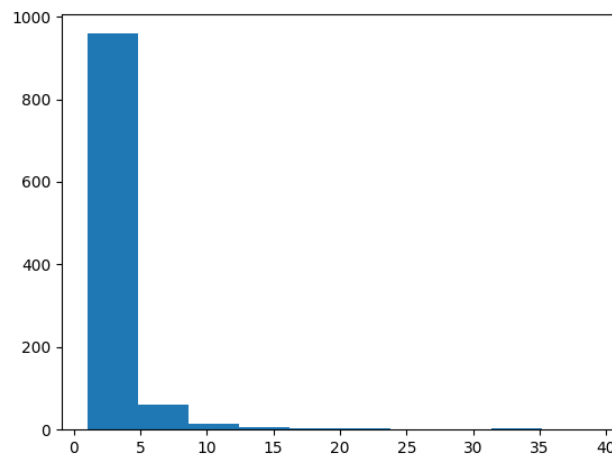
2.View Of network IUST Student



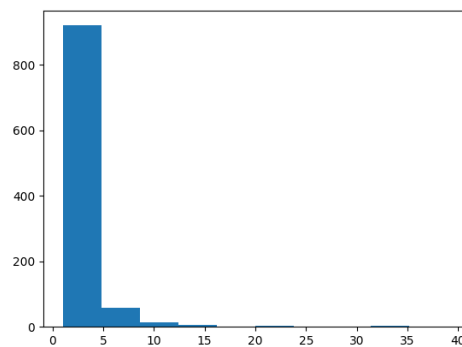
3.View Of network IUST Professors



4..View Of network Facebook Relations



5.View Of network Email Core



What do you see interesting in the computed measures?

The diameter of a graph is the longest distance between any two nodes. The degree of a node is the number of edges or links from and to this node. Intuitively, the higher the node degrees, the denser the graph. If you have n nodes and the maximal degree of the nodes is $n-1$, then the graph diameter is 1.

The larger the network, the larger its diameter, so that in a network with a thousand nodes, the diameter is eighteen, but in other networks, it is 5.

The higher the number of nodes, the more it is possible to visualize them.

The average degree of an undirected graph is used to measure the number of edges compared to the number of nodes. To do this we simply divide the summation of all nodes' degree by the total number of nodes.

The larger the network, the lower the density.

The bigger the network, the longer the shortest path.

Source

— [NetworkX 3.1 documentation](#)