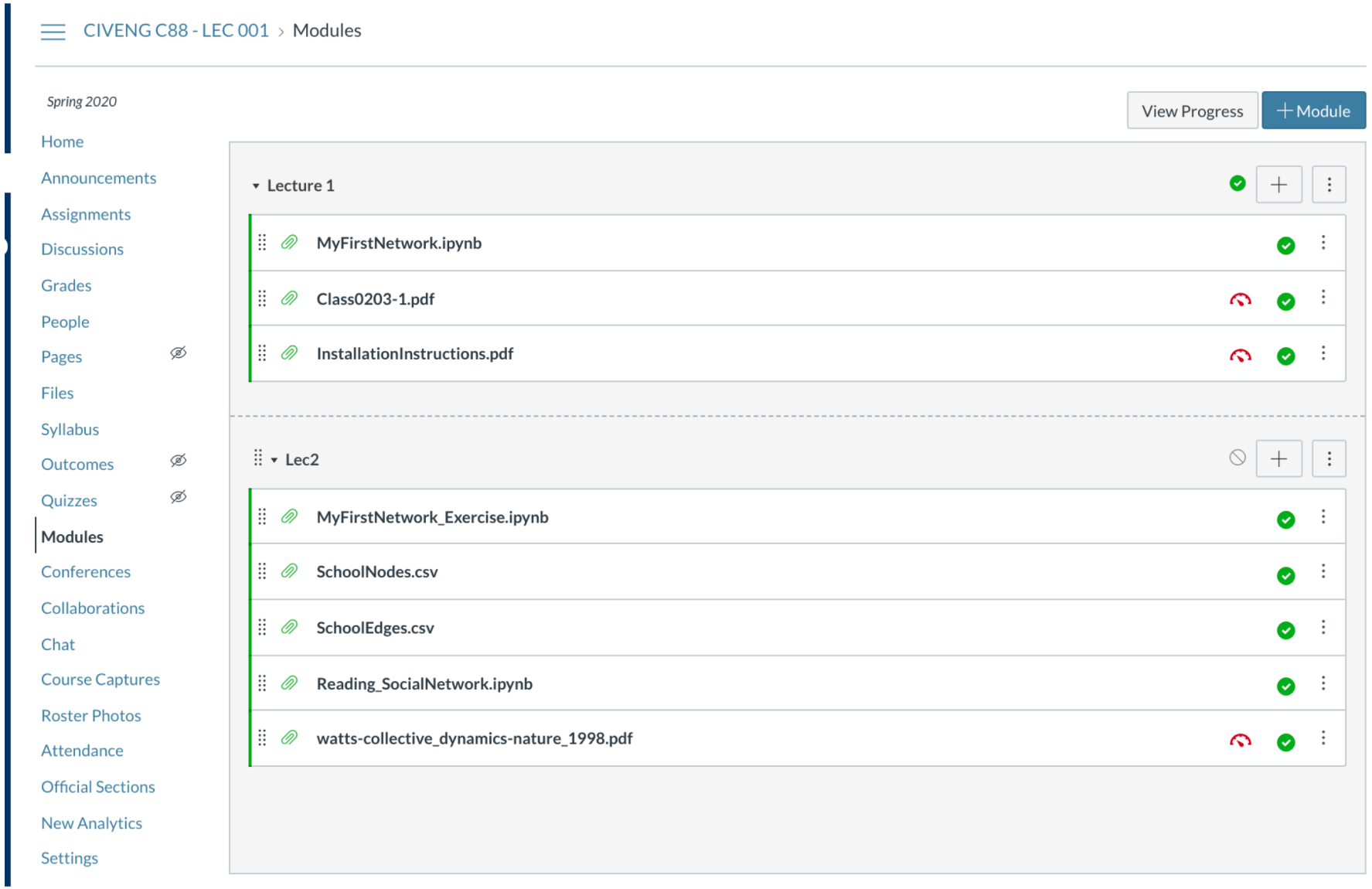
In bcourses be sure to see Modules to see find files separated by class



Materials Part I

MyFirstNetwork_Exercise.ipynb.ipynb

Class0210_partI.pdf

The core Method of CE88: Network Science in a framework of Complex Systems thinking to study urban systems.

Weeks 1-3: Network Measures
Weeks 3-6: Clusters and Centrality
Weeks 7-8: Tools Open Street Maps and Visualization

Mid Term: Project Proposal

Week 9-11: Methods and Papers based on projects:
Multigraphs, Robustness, Spreading Dynamics

Network Measures:

- Degree
- Degree Distribution
- Shortest path length
- Average path length

Node network properties

from immediate connections

- indegree

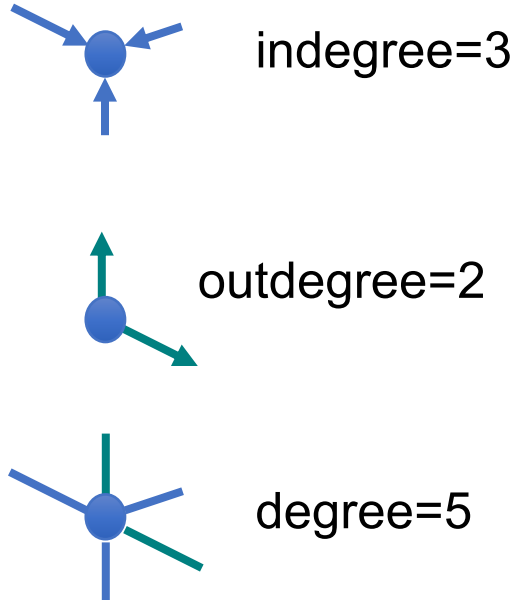
  how many directed edges (arcs) are incident on a node

- outdegree

  how many directed edges (arcs) originate at a node

- degree (in or out)

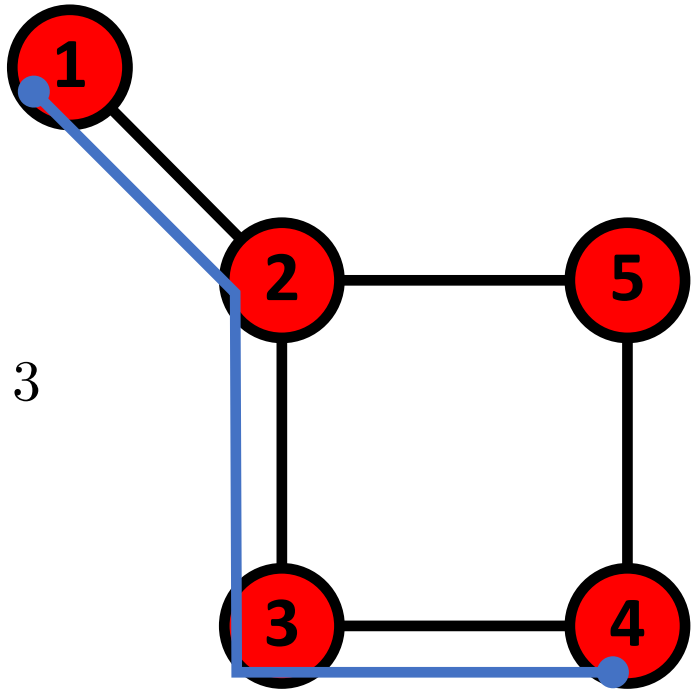
  number of edges incident on a node



Average Path Length

$$(l_{1\to2} + l_{1\to3} + l_{1\to4} + \\ + l_{1\to5} + l_{2\to3} + l_{2\to4} + \\ + l_{2\to5} + l_{3\to4} + l_{3\to5} + \\ + l_{4\to5})/10 = 1.6$$

$l_{1\to4} = 3$



The average of the shortest paths for all pairs of nodes.

Numbers of pairs for a network of size N is N(N-1)/2

Note: You will need to use the documentation of NetworkX to develop the ideas and projects of he class.

No previous experience is needed

https://networkx.github.io/

NetworkX

Stable (notes)

2.4 — October 2019

download | doc | pdf

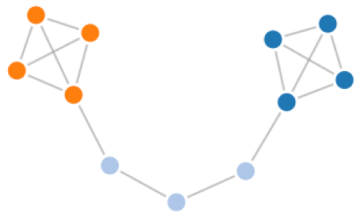
Latest (notes)

2.5 development

github | doc | pdf

Software for complex networks

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.



average_shortest_path_length(*G, weight=None*) [source]

Return the average shortest path length.

The average shortest path length is

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

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`.

shortest_path(*G, source=None, target=None, weight=None*) [source]

Compute shortest paths in the graph.

Parameters:

- **G** (*NetworkX graph*)
- **source** (*node, optional*) – Starting node for path. If not specified, compute shortest paths for each possible starting node.
- **target** (*node, optional*) – Ending node for path. If not specified, compute shortest paths to all possible nodes.
- **weight** (*None or string, 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.

Returns:

path – All returned paths include both the source and target in the path.

shortest_path_length(*G, source=None, target=None, weight=None*) [source]

Compute shortest path lengths in the graph.

Parameters:

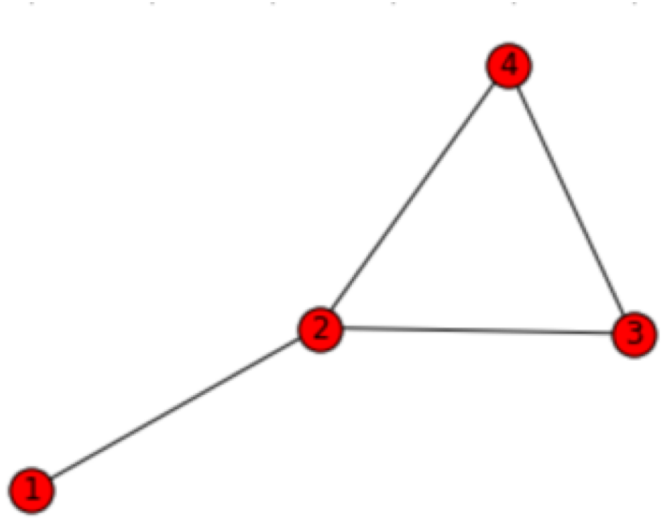
- **G** (*NetworkX graph*)
- **source** (*node, optional*) – Starting node for path. If not specified, compute shortest path lengths using all nodes as source nodes.
- **target** (*node, optional*) – Ending node for path. If not specified, compute shortest path lengths using all nodes as target nodes.
- **weight** (*None or string, 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.

Returns:

length – If the source and target are both specified, return the length of the shortest path from the source to the target.

Lab 1

1) Nodes
2) Edges
3) Number of Nodes
4) Number of Edges
5) Neighbors of each node
6) Degree of each node
7) Average degree calculation and short equation

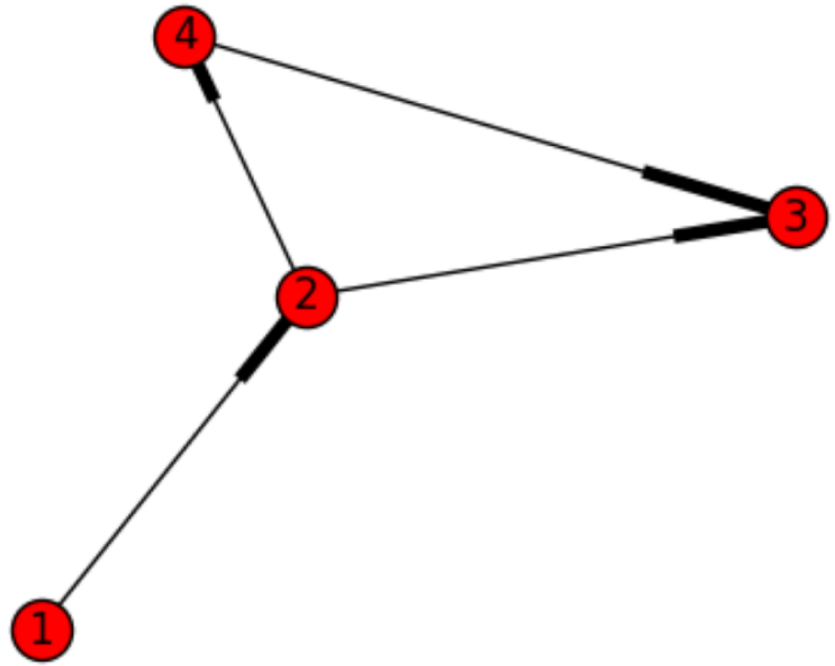


Lab 1

1) Nodes
2) Edges
3) Number of Nodes
4) Number of Edges
5) Neighbors of each node
6) Degree of each node

   In-degree of each node

   out-degree of each node

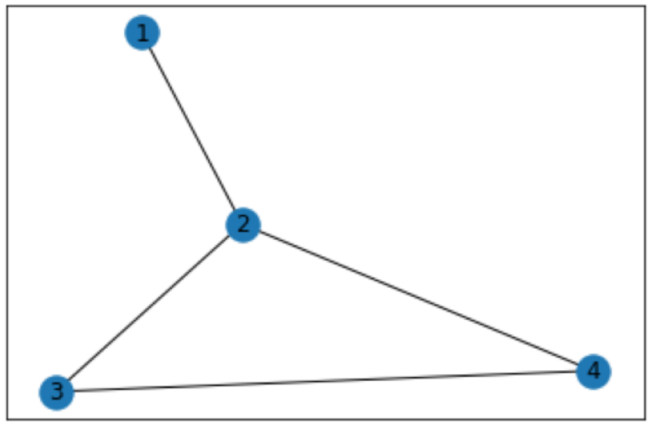


7) Average indegree outdegree and degree calculation with short equation

Exercise: Create the 4 node network of Lecture 1 and respond the following questions (use networkx documentation or Google the questions)

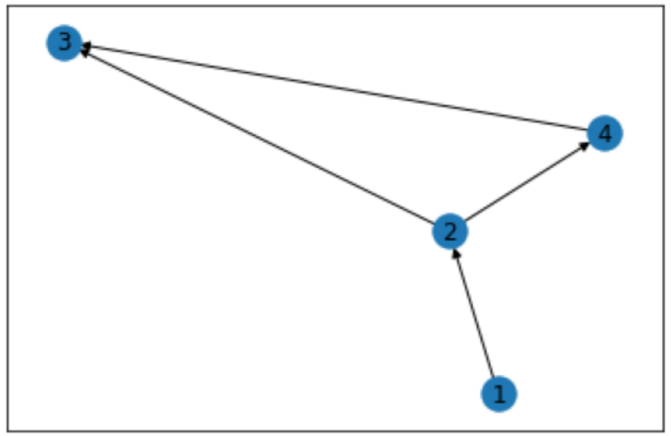
```
g = nx.Graph()  #Graph base class for undirected graphs. See also:DiGrap
g.add_edge('1','2',weight=1.0) #Nodes can be arbitrary (hashable) Python
g.add_edge('2','3',weight=1.0) #Edges are represented as links between n
g.add_edge('2','4',weight=1.0) #We create a network by adding 11 edges t
g.add_edge('4','3',weight=1.0)
```

```
nx.draw_networkx(g) #The draw_networkx function is called (check online
```




```
gd = nx.DiGraph()  #Graph base class for undirected graphs. See also:DiG
gd.add_edge('1','2',weight=1.0) #Nodes can be arbitrary (hashable) Pytho
gd.add_edge('2','3',weight=1.0) #Edges are represented as links between
gd.add_edge('2','4',weight=1.0) #We create a network by adding 11 edges
gd.add_edge('4','3',weight=1.0)
```

```
nx.draw_networkx(gd)
```



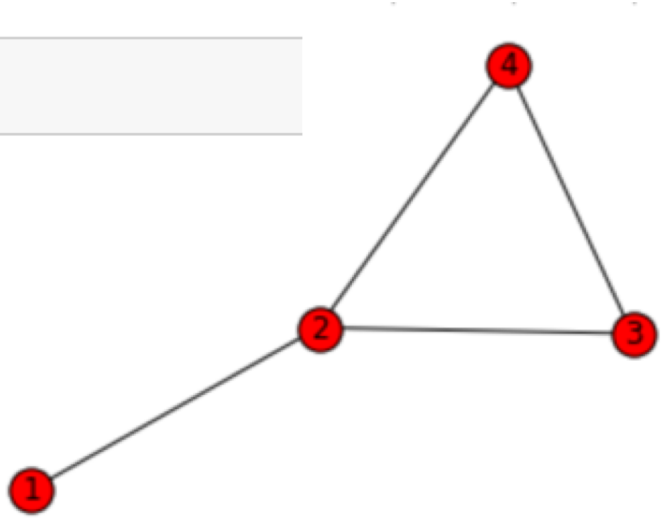
Note the Difference between Graph and DiGraph, they create Undirected and Dierected Graphfs respectively
Here the Weight of each link is the same

Lab 1

1) Nodes :

```
print ("Nodes: ", g.nodes())
```

```
Nodes:  ['1', '2', '3', '4']
```



2) Edges

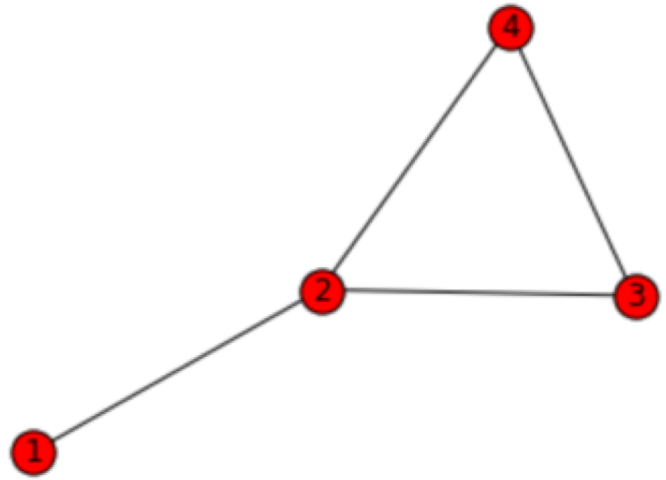
```
print ("Edges: " , g.edges())
```

```
Edges:  [('1', '2'), ('2', '3'), ('2', '4'), ('3', '4')]
```

3) Print number of nodes

```
print ("Number of nodes:" , g.number_of_nodes())
```

```
Number of nodes: 4
```



4) Print number of Edges

```
print ("Number of edges:" , g.number_of_edges())
```

```
Number of edges: 4
```

5) Iterate through nodes and print neighbors

```
for node in g.nodes():
    print ("Neighbors of ", node, " are : ", list(g.neighbors(node)))
```

```
Neighbors of  1  are :  ['2']
Neighbors of  2  are :  ['1', '3', '4']
Neighbors of  3  are :  ['2', '4']
Neighbors of  4  are :  ['2', '3']
```

6) Iterate through nodes and print degree

```
for node in g.nodes():
    print ("Degree of ", node, " is : ", len(list(g.neighbors(node))),"or ",g.degree(node))
```

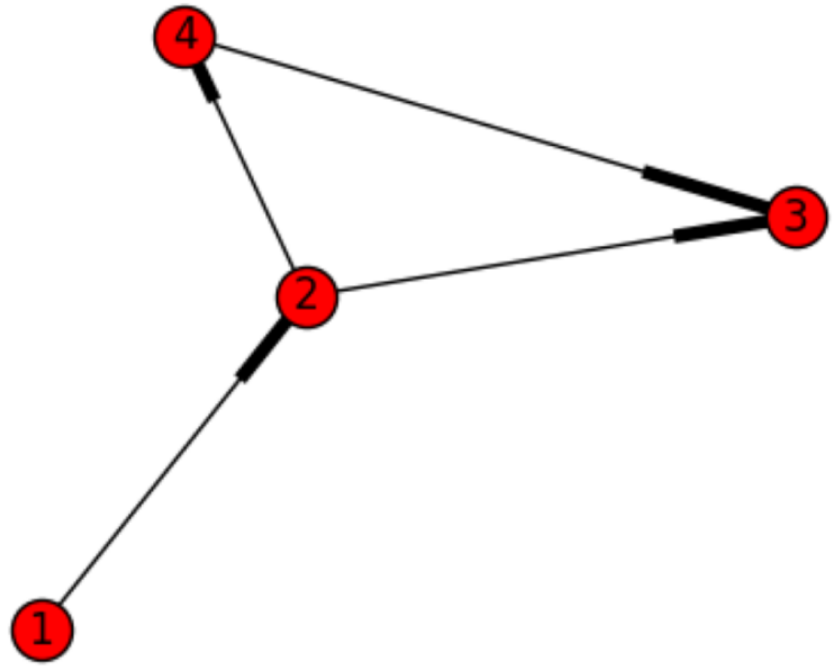
```
Degree of  1  is :  1 or  1
Degree of  2  is :  3 or  3
Degree of  3  is :  2 or  2
Degree of  4  is :  2 or  2
```

Lab 1

1) Nodes
2) Edges
3) Number of Nodes
4) Number of Edges
5) Number of Neighbors per node
6) Degree of each node

   In-degree of each node

   out-degree of each node



7) Average indegree outdegree and degree calculation with short equation

```
print ("Nodes: " , gd.nodes())
```

```
Nodes:  ['1', '2', '3', '4']
```

```
print ("Edges: " , gd.edges())
```

```
Edges:  [('1', '2'), ('2', '3'), ('2', '4'), ('4', '3')]
```

```
print ("#Nodes: " , gd.number_of_nodes())
```

```
#Nodes:  4
```

```
print ("#Edges: " , gd.number_of_edges())
```

```
#Edges:  4
```

```
for node in gd.nodes():
    print ("Neighbors of ", node, " are : ", list(gd.neighbors(node)))
```

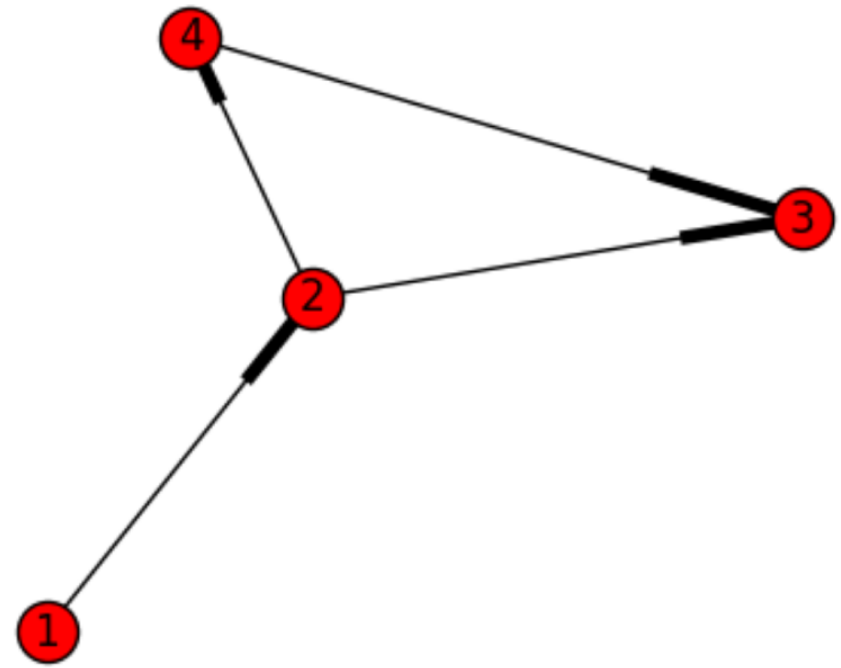
```
Neighbors of  1  are :  ['2']
Neighbors of  2  are :  ['3', '4']
Neighbors of  3  are :  []
Neighbors of  4  are :  ['3']
```

```
for u,v in gd.edges():
    print ("Weight of edge [", u, "][", v,"] is ", g[u][v]['weight'])
```

```
Weight of edge [ 1 ][ 2 ] is  1.0
Weight of edge [ 2 ][ 3 ] is  1.0
Weight of edge [ 2 ][ 4 ] is  1.0
Weight of edge [ 4 ][ 3 ] is  1.0
```

```
for node in gd.nodes():
    print ("Degree of ", node, " is : ", gd.degree(node))
```

```
Degree of  1  is :  1
Degree of  2  is :  3
Degree of  3  is :  2
Degree of  4  is :  2
```



```
for node in gd.nodes():
    print ("Degree of ", node, " is : ", gd.degree(node))
```

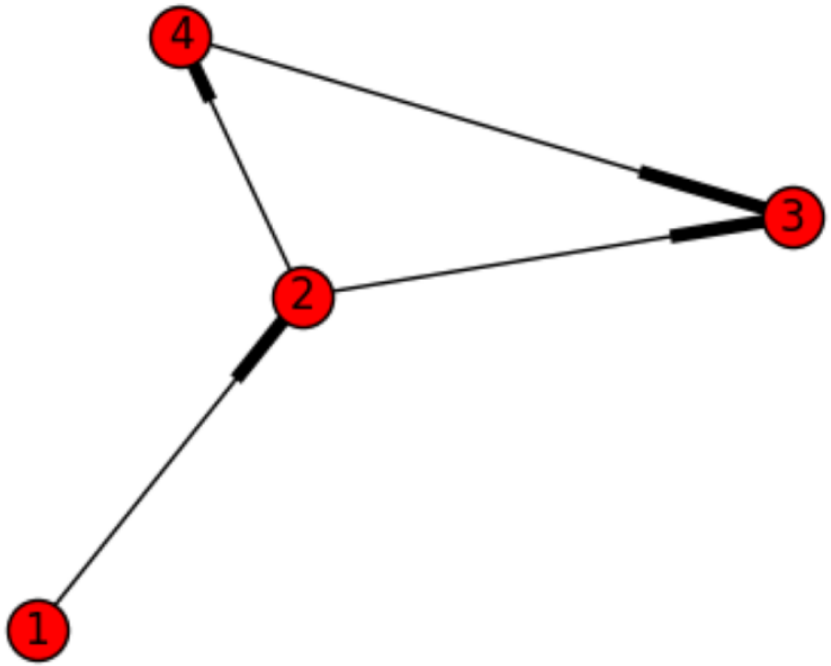
```
Degree of  1  is :  1
Degree of  2  is :  3
Degree of  3  is :  2
Degree of  4  is :  2
```

```
for node in gd.nodes():
    print ("In-Degree of ", node, " is : ", gd.in_degree(node))
```

```
In-Degree of  1  is :  0
In-Degree of  2  is :  1
In-Degree of  3  is :  2
In-Degree of  4  is :  1
```

```
for node in gd.nodes():
    print ("Out-degree of ", node, " is : ", gd.out_degree(node))
```

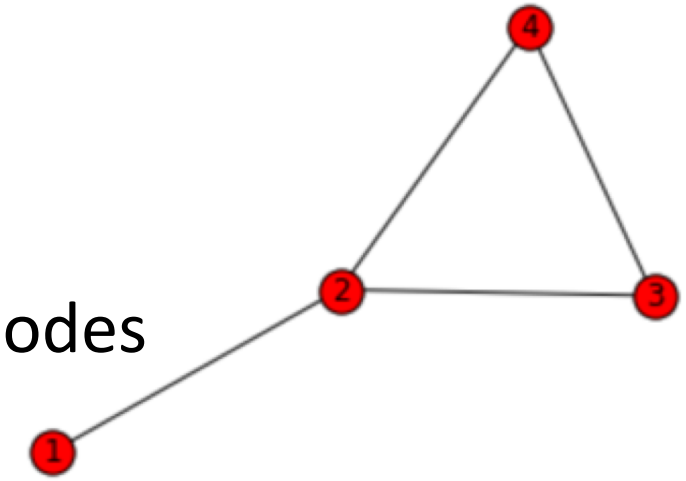
```
Out-degree of  1  is :  1
Out-degree of  2  is :  2
Out-degree of  3  is :  0
Out-degree of  4  is :  1
```



How do we calculate the average degree?

- It is the sum of the degree of each node divided by the number of nodes

<degree> = (1+3+2+2)/4 = 2



- It can be also written as the 2x(Number of Links)/#Nodes

<degree> = 2x(4)/4 = 2

```
undirected graph = ", np.mean(list(dict(g.degree()).values())), ", or: ", 2.0*g.number_of_edges()/g.number_of_nodes())
```

```
Average degree of  undirected graph =  2.0 , or:  2.0
```

```
np.mean(list(dict(g.degree()).values()))
```

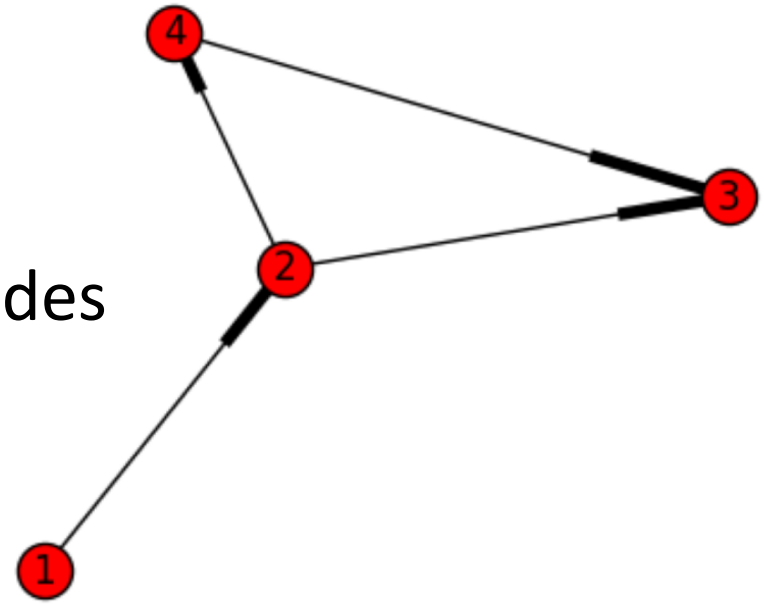
```
2.0
```

How do we calculate the average degree?

- It is the sum of the degree of each node divided by the number of nodes

  <in-degree> = (0+1+2+1)/4 = 1

  <out-degree> = (1+2+0+1)/4 = 1



- It can be also written as the (Number of Links)/#Nodes

  <in-degree> = (4)/4 = 1 <out-degree> = (4)/4 = 1

  - Finally, the average degree is

    2x(Number of Links)/#Nodes or

    <degree>=<out-degree> + <in-degree> = 2

```
print ("Average out-degree of  directed graph = ",np.mean(list(dict(gd.out_degree()).values())),", or: ",gd.number_of_
```

```
Average out degree of  directed graph =  1.0 , or:  1.0
```

```
print ("Average in-degree of  directed graph = ",np.mean(list(dict(gd.in_degree()).values())),", or: ",gd.number_of_ed
```

```
Average in-degree of  directed graph =  1.0 , or:  1.0
```

```
print ("Average degree of  directed graph = ",np.mean(list(dict(gd.degree()).values())),", or: ",np.mean(list(dict(gd.
```

```
Average degree of  directed graph =  2.0 , or:  2.0
```

```
np.mean(list(dict(gd.in_degree()).values()))+np.mean(list(dict(gd.out_degree()).values()))
```

```
2.0
```

```
np.mean(list(dict(gd.degree()).values()))
```

```
2.0
```

Use 10 minutes to complete MyFirstNetwork_Exercise

Question 1: extract the degree of node 3 in Graph g

In [ ]:

Question 2: extract the in-degree, out-degree and degree of node 4 in Graph dg

Hint:Check this link for in-degree and out-degree of a directed graph

https://networkx.github.io/documentation/networkx-1.10/reference/classes.digraph.html

In [ ]:

In [ ]:

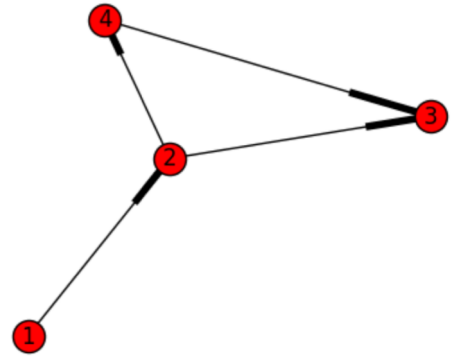
In [ ]:

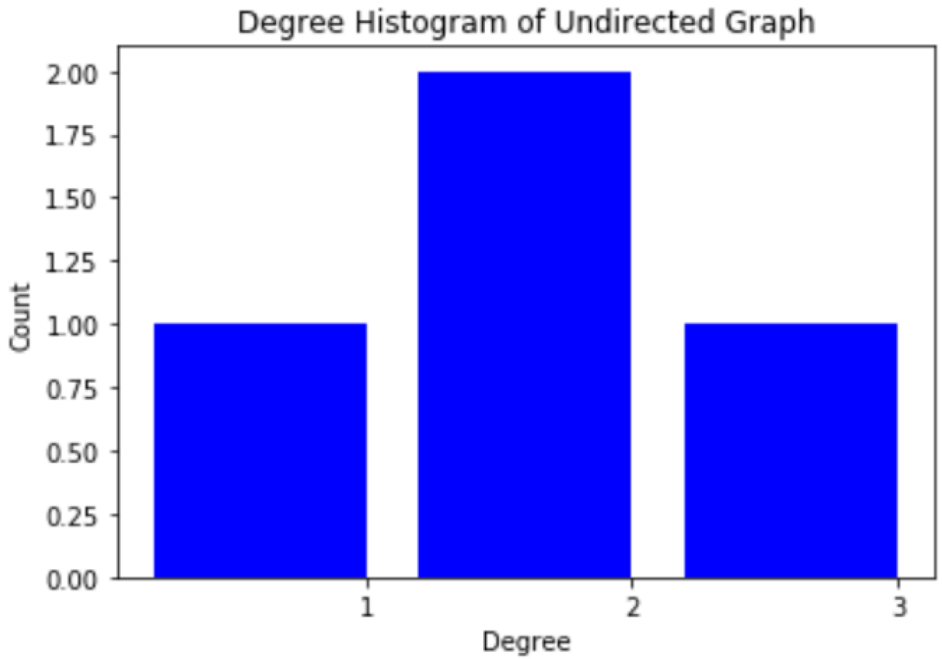
Question 3: Create the plot of the degree distribution of both graphs

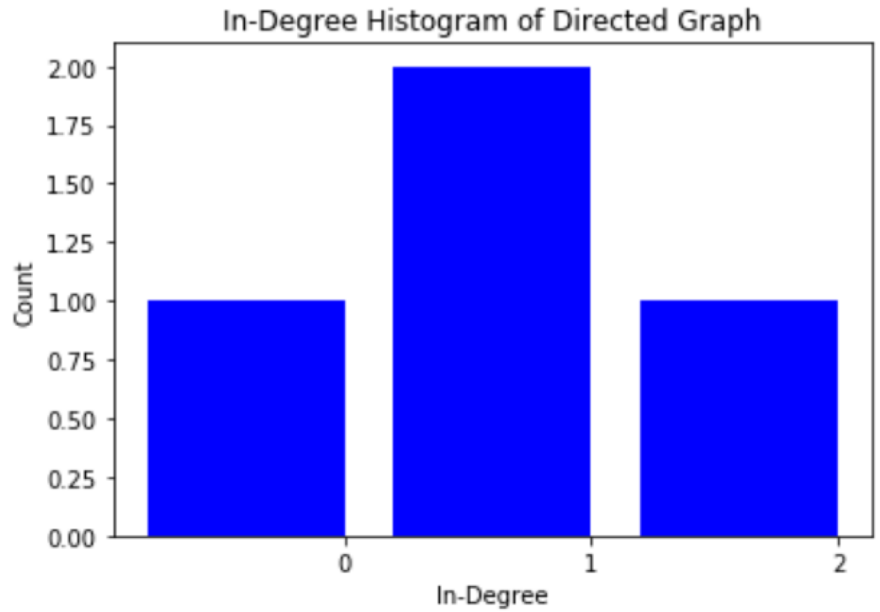
Use the code of this example https://networkx.github.io/documentation/stable/auto_examples/drawing/plot_degree_histogram.html Note that you do not need to use their G = nx.gnp_random_graph(100, 0.02), but the g and dg defined above

In [ ]:

In [ ]:


4
3
2
1


Degree Histogram of Undirected Graph
Count
Degree
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1
2
3


In-Degree Histogram of Directed Graph
Count
In-Degree
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0
1
2