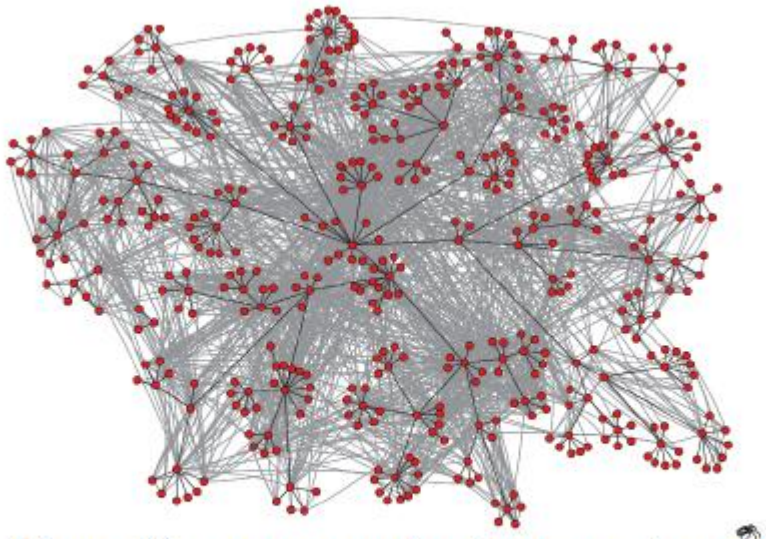
Why Study Networks?

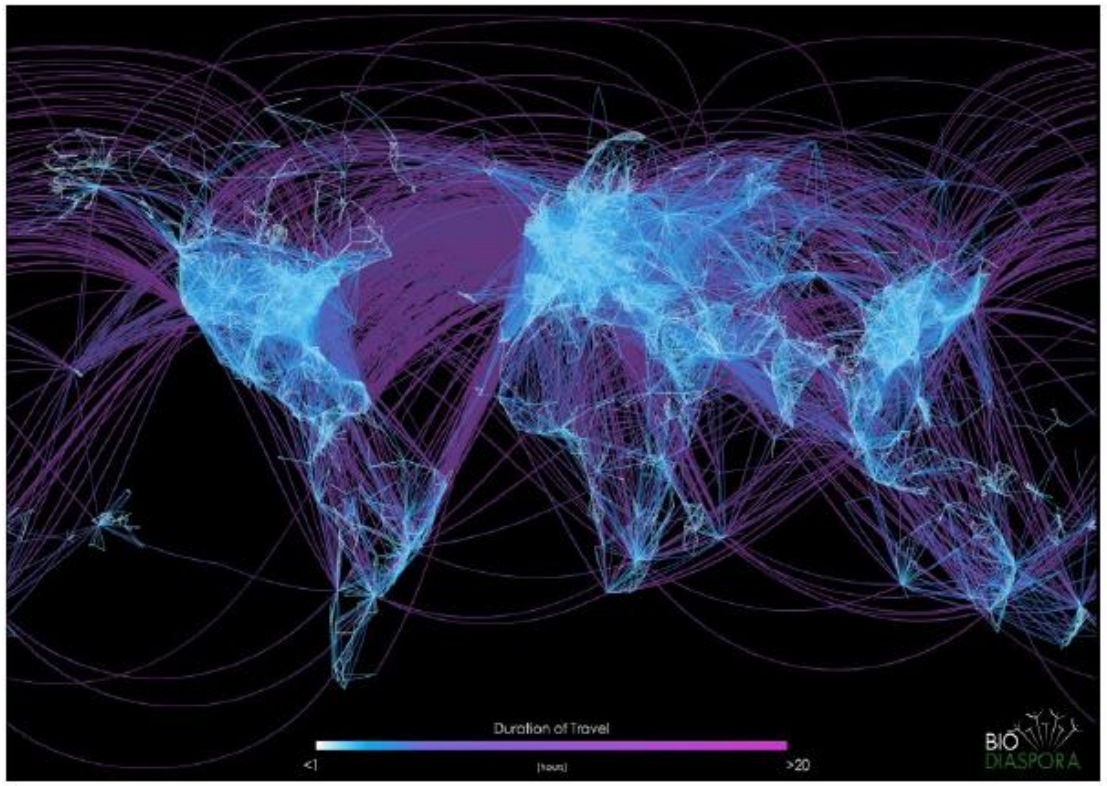
Social Network



E-mail communication network
among 436 HP employees [Adamic & Adar 2005

Is rumor likely to spread in this network?
Who is the most influential person in this organization?

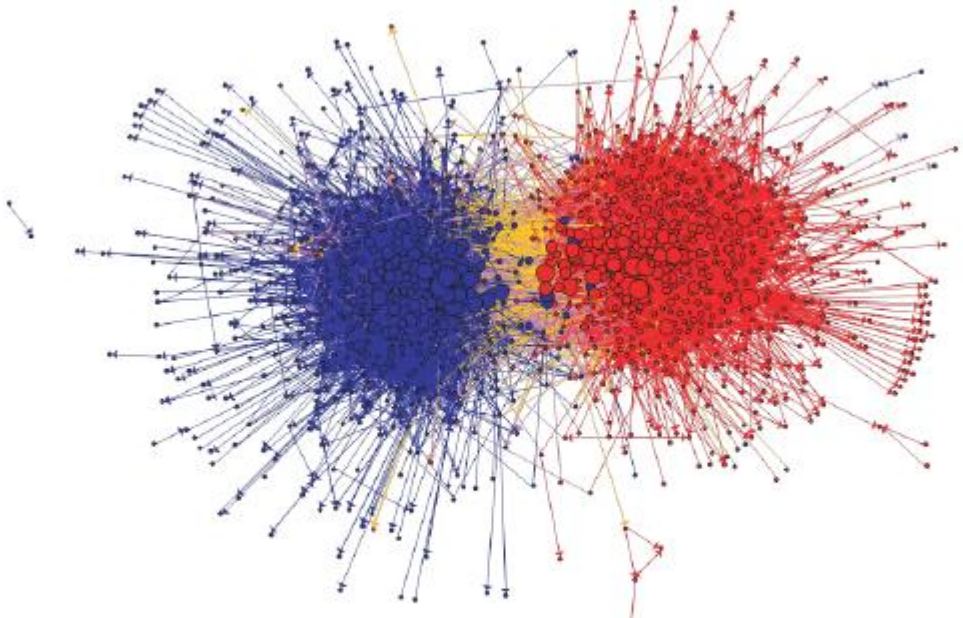
Transportation/Mobility Network



Network of direct flights around the world

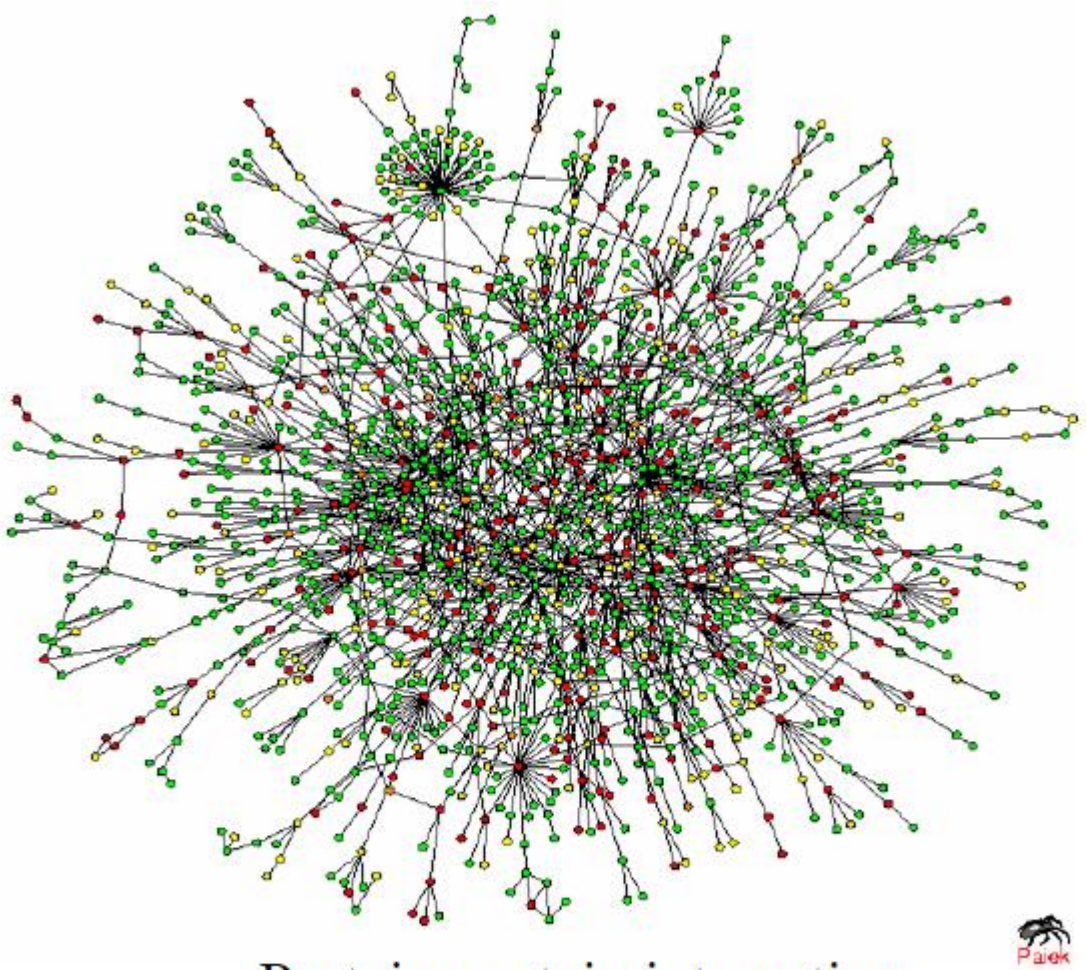
Which airports are at the highest risk of spreading virus?
Which part of the world is most difficult to reach?
What is the smallest legs possible between two points?

Information Network



Communication between left-wing and right-wing political blogs [Adamic & Glance 2005]

Biological Network



Protein-protein interactions [Jeong et al. 2001]

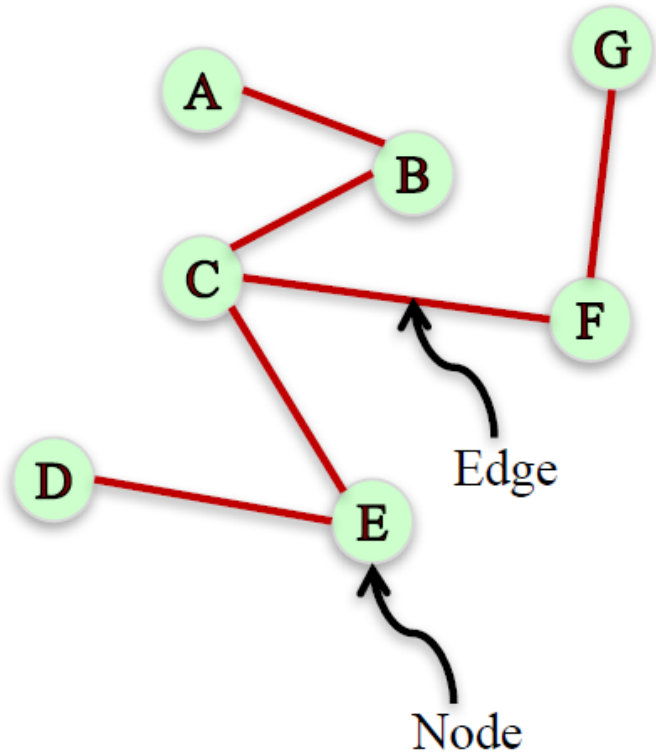
And More…

- Financial networks
- Co-authorship networks
- Trade networks
- Citation networks

Network Definition and Vocabulary

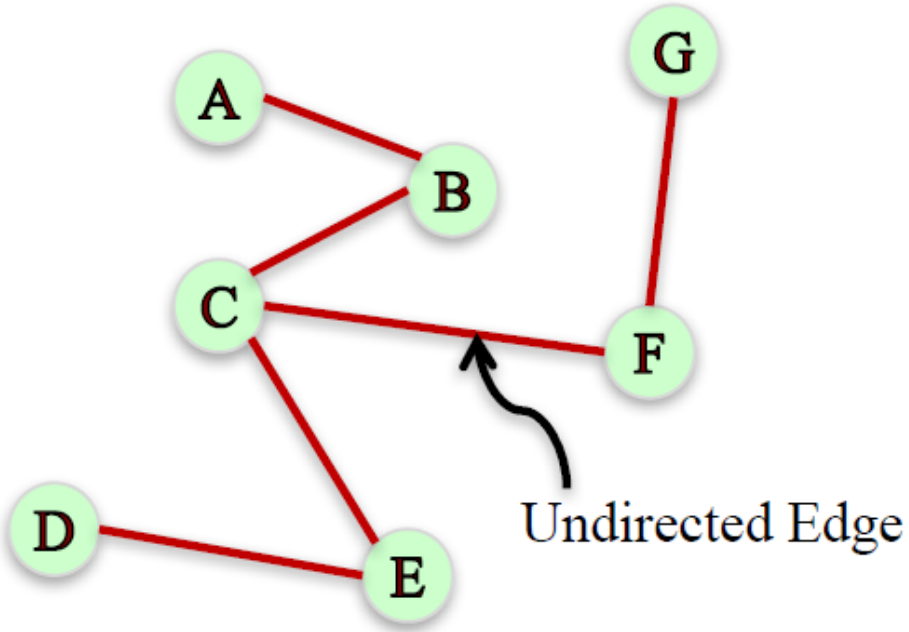
Network (or Graph): A representation of connections among a set of items.

- Items are called nodes (or vertices)
- Connections are called edges (or link or ties)



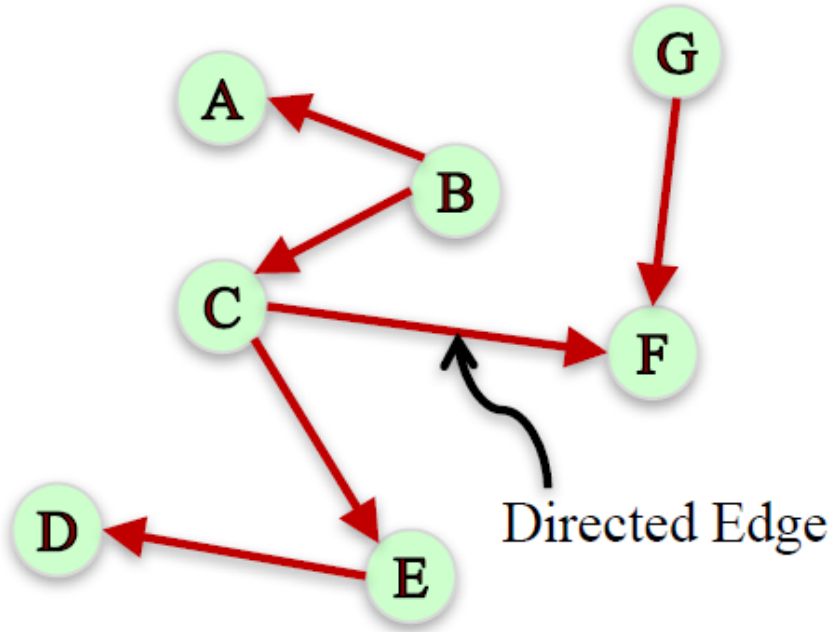

```
import networkx as nx
G=nx.Graph()
G.add_edge('A','B')
G.add_edge('B','C')
```

Edge Direction



Undirected network:
edges have no direction

```
G=nx.Graph()
G.add_edge('A','B')
G.add_edge('B','C')
```



Directed network:
edges have direction

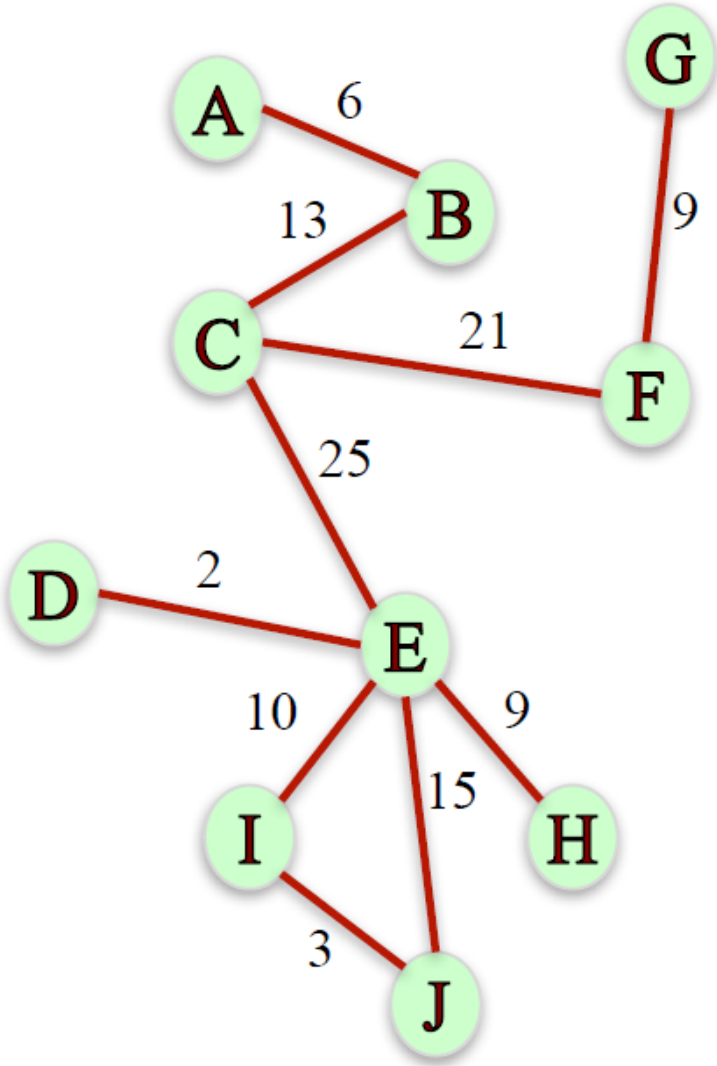
```
G=nx.DiGraph()
G.add_edge('B', 'A')
G.add_edge('B','C')
```

Weighted Networks

Not all relationships are equal.
Some edges carry higher weight than others.

Weighted network: a network where edges are assigned a (typically numerical) weight.

```
G=nx.Graph()
G.add_edge('A','B', weight = 6)
G.add_edge('B','C', weight = 13)
```

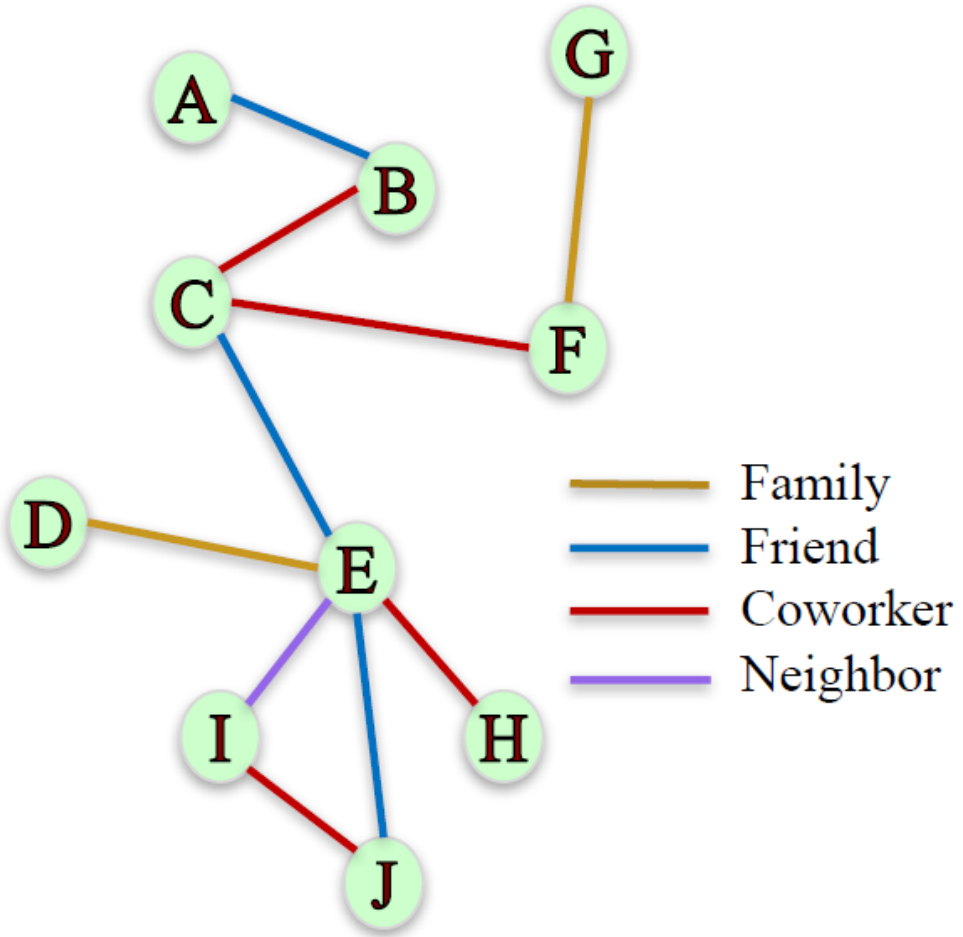


Number of times coworkers had lunch together in one year

Other Edge Attributes

Edges can carry many other labels or attributes

```
G=nx.Graph()
G.add_edge('A','B', relation= 'friend')
G.add_edge('B','C', relation= 'coworker')
G.add_edge('D','E', relation= 'family')
G.add_edge('E','I', relation= 'neighbor')
```

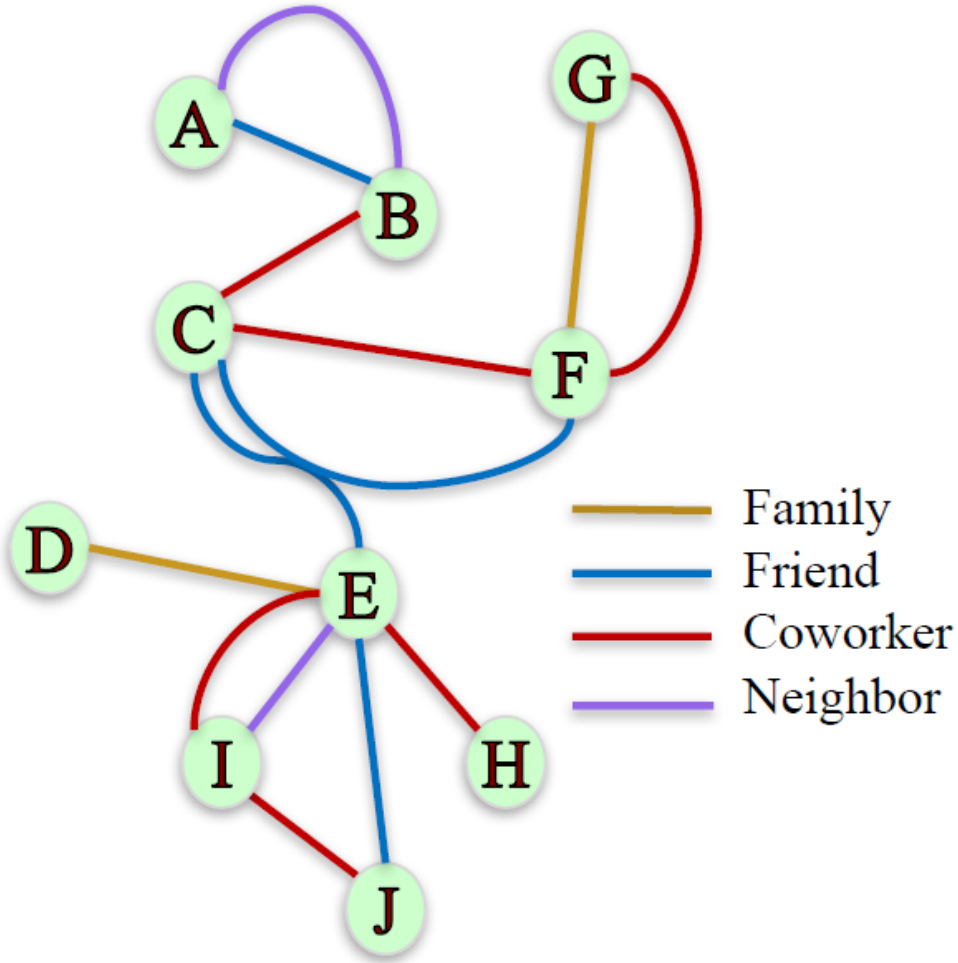


Mutigraphs

A pair of nodes can have different types of relationships simultaneously

Multigraph: A network where multiple edges can connect the same nodes (parallel edges).

```
G=nx.MultiGraph()
G.add_edge('A','B', relation= 'friend')
G.add_edge('A','B', relation= neighbor')
G.add_edge('G','F', relation= 'family')
G.add_edge('G','F', relation= 'coworker')
```



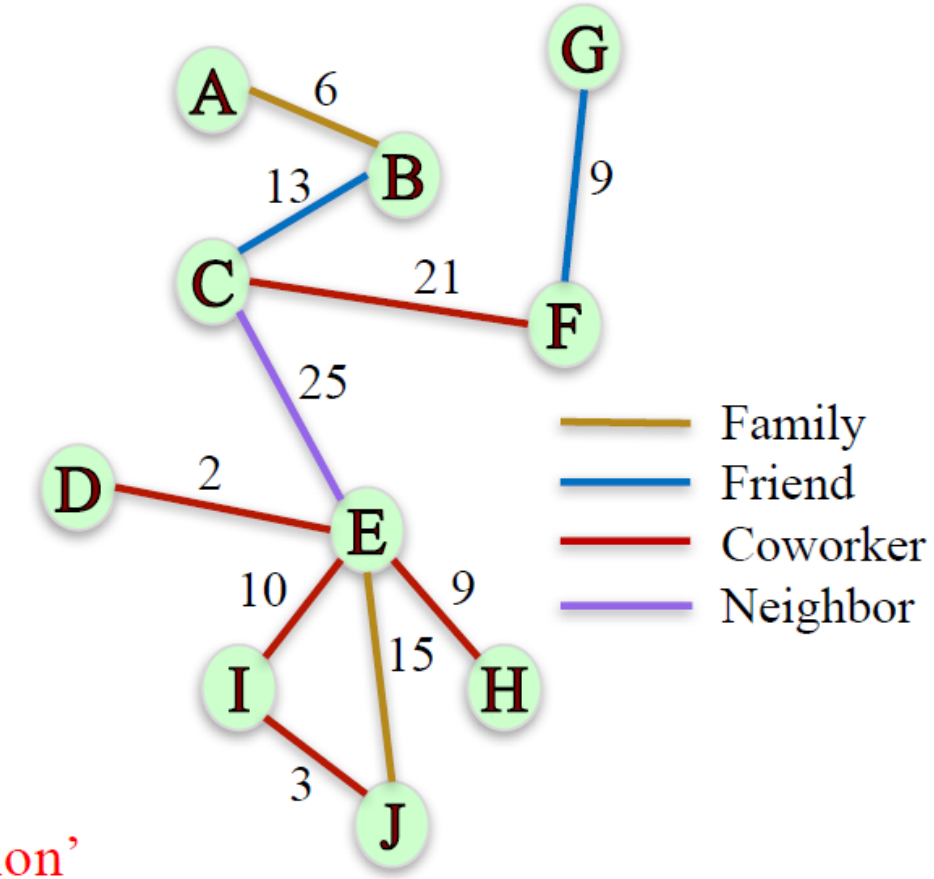
Edge Attributes in NetworkX

```
G=nx.Graph()
G.add_edge('A','B', weight= 6, relation = 'family')
G.add_edge('B','C', weight= 13, relation = 'friend')
```

```
In: G.edges() #list of all edges
Out: [('A', 'B'), ('C', 'B')]
```

```
In: G.edges(data= True)  #list of all edges with attributes
Out: [('A', 'B', {'relation': 'family', 'weight': 6}),
('C', 'B', {'relation': 'friend', 'weight': 13})]
```

```
In: G.edges(data= 'relation') #list of all edges with attribute 'relation'
Out: [('A', 'B', 'family'), ('C', 'B', 'friend')]
```

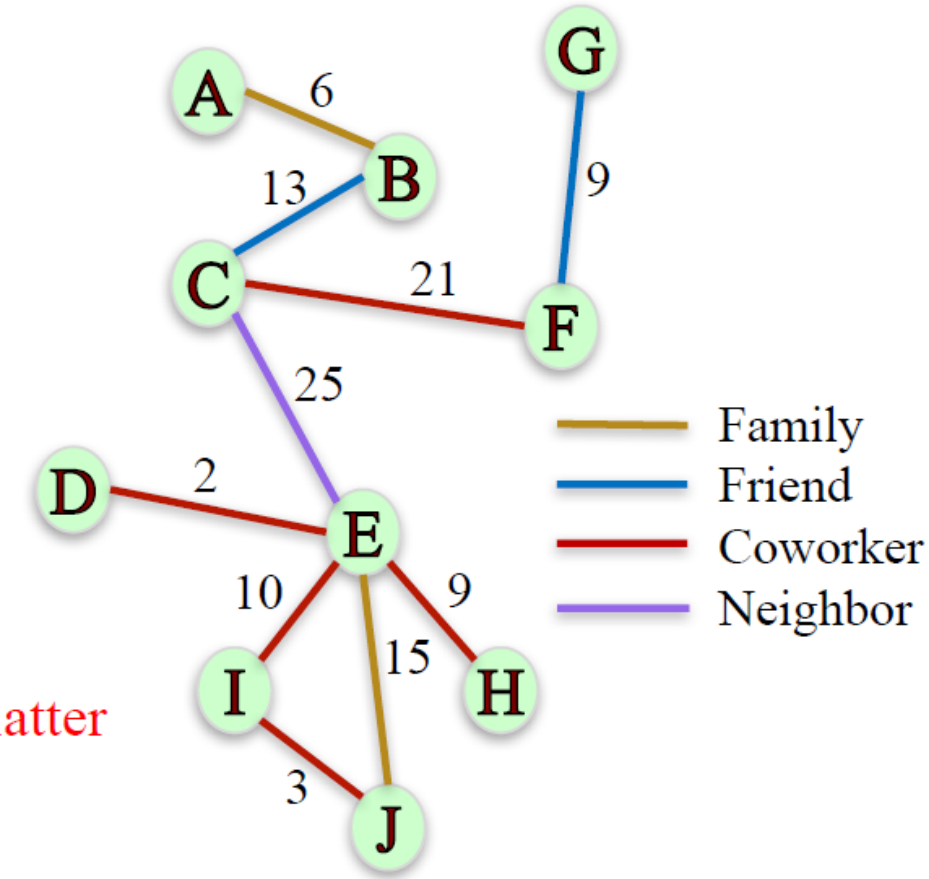


Number of times coworkers had lunch together in one year

Edge Attributes in NetworkX

Accessing attributes of a specific edge:

```
In: G.edge['A']['B'] # dictionary of attributes of edge (A, B)
Out: {'relation': 'family', 'weight': 6}

In: G.edge['B']['C']['weight']
Out: 13

In: G.edge['C']['B']['weight'] # undirected graph, order does not matter
Out: 13
```



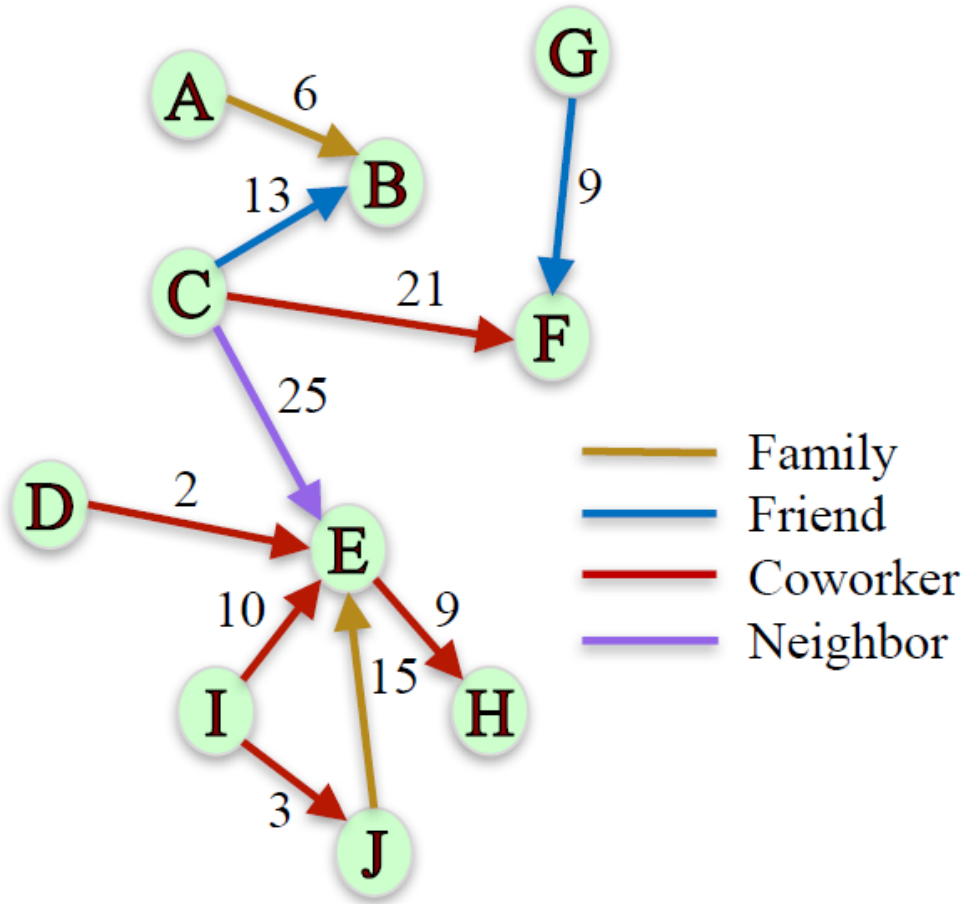
Number of times coworkers had lunch together in one year

Directed, weighted network:

```
G=nx.DiGraph()
G.add_edge('A','B', weight= 6, relation = 'family')
G.add_edge('C', 'B', weight= 13, relation = 'friend')
```

Accessing edge attributes:

```
In: G.edge['C']['B']['weight']
Out: 13

In: G.edge['B']['C']['weight'] # directed graph, order matters
Out: KeyError: 'C'
```

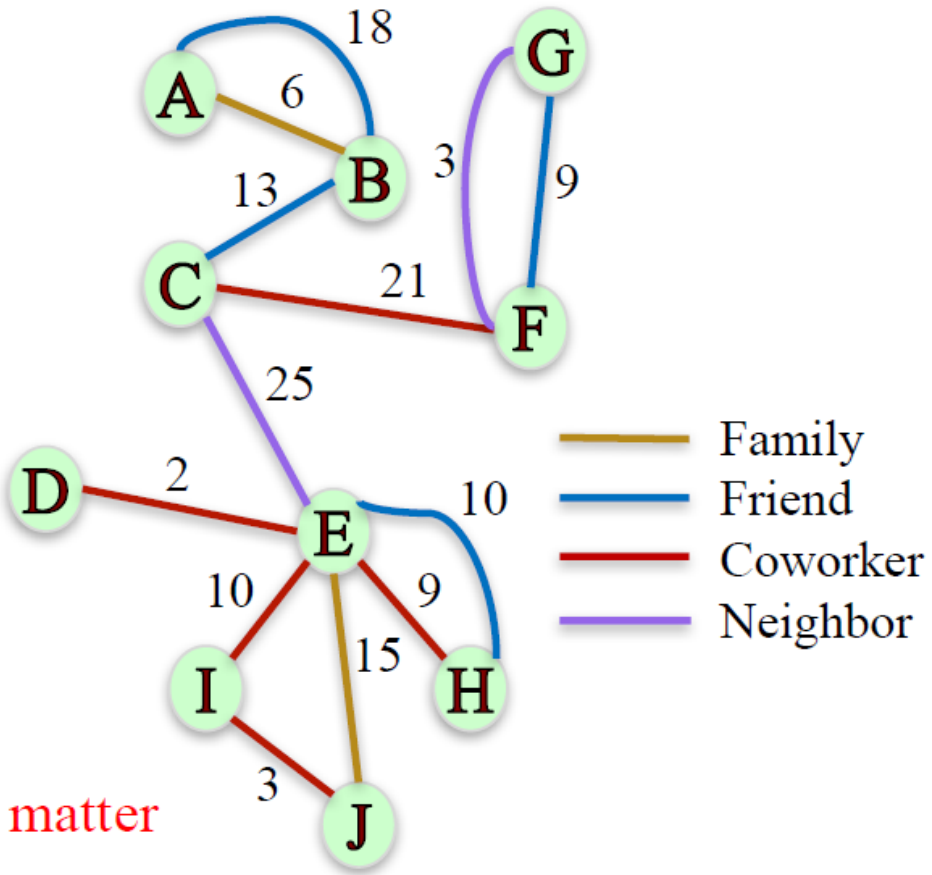


MultiGraph:

```
G=nx.MultiGraph()
G.add_edge('A','B', weight= 6, relation = 'family')
G.add_edge('A','B', weight= 18, relation = 'friend')
G.add_edge('C','B', weight= 13, relation = 'friend')
```

Accessing edge attributes:

```
In: G.edge['A']['B'] # One dictionary of attributes per (A,B) edge
Out: {0: {'relation': 'family', 'weight': 6},
1: {'relation': 'friend', 'weight': 18}}

In: G.edge['A']['B'][0]['weight'] # undirected graph, order does not matter
Out: 6
```

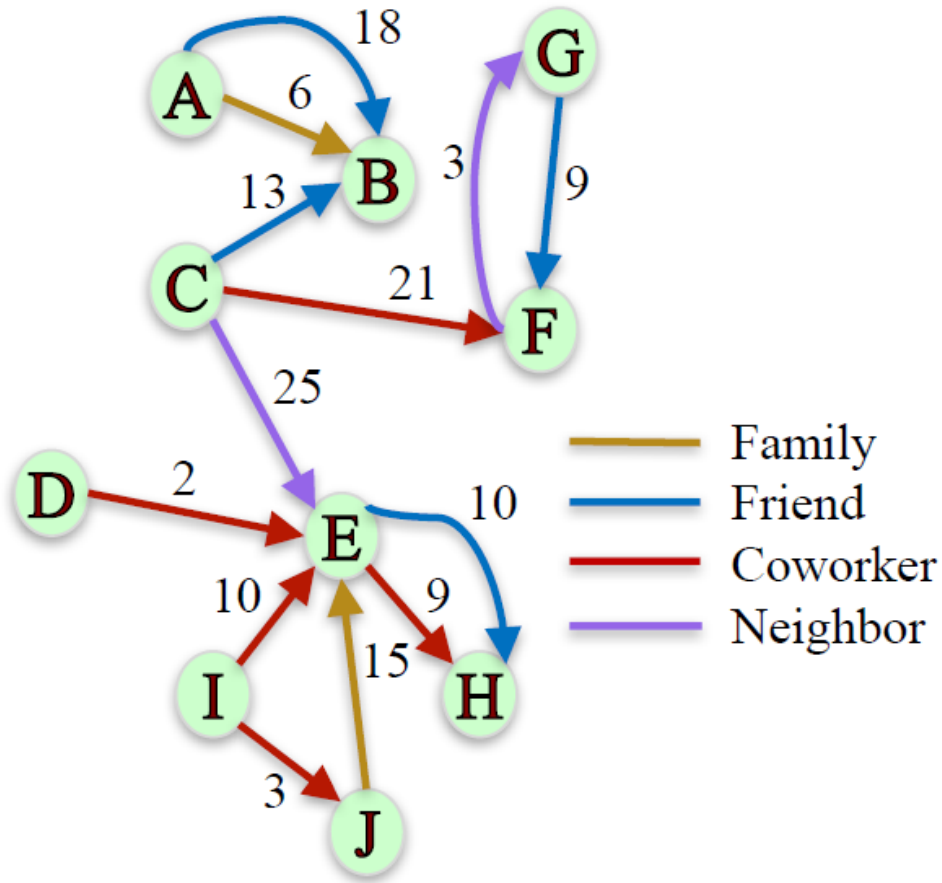


Directed MultiGraph:

```
G=nx.MultiDiGraph()
G.add_edge('A','B', weight= 6, relation = 'family')
G.add_edge('A','B', weight= 18, relation = 'friend')
G.add_edge('C','B', weight= 13, relation = 'friend')
```

Accessing edge attributes:

```
In: G.edge['A']['B'][0]['weight']
Out: 6

In: G.edge['B']['A'][0]['weight'] # directed graph, order matters
Out: KeyError: 'A'
```

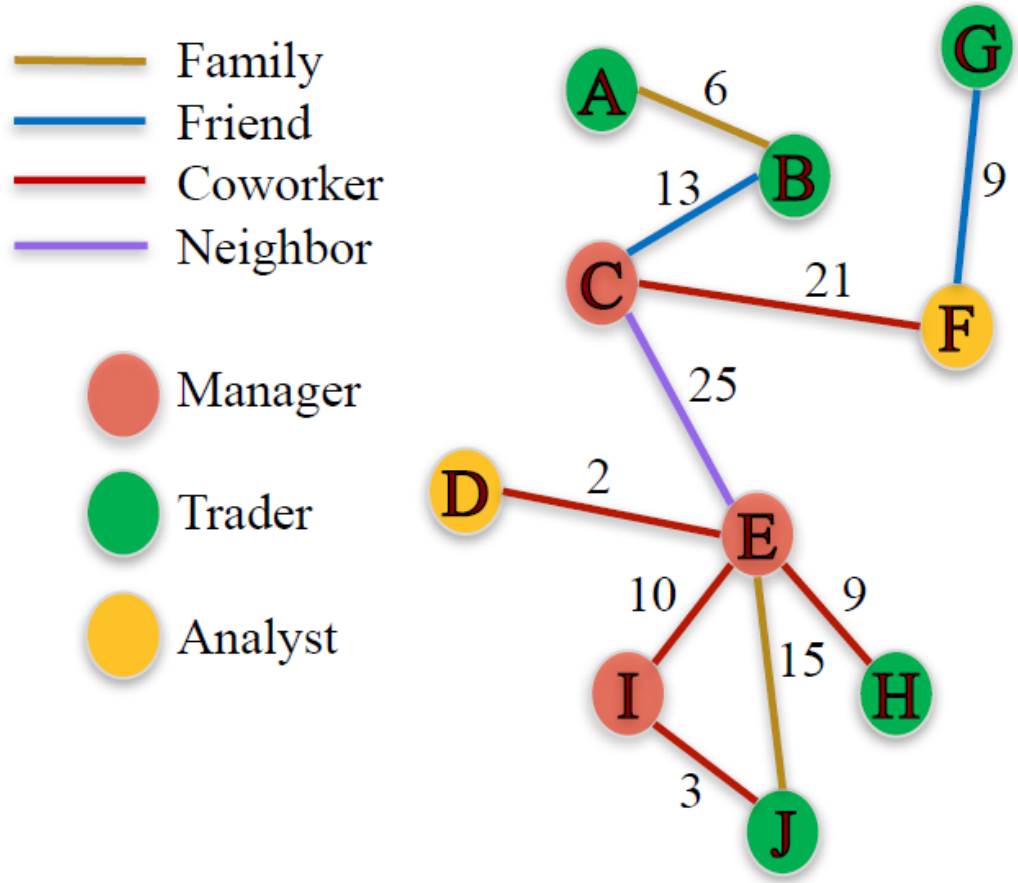


Node Attributes in NetworkX

```
G=nx.Graph()
G.add_edge('A','B', weight= 6, relation = 'family')
G.add_edge('B','C', weight= 13, relation = 'friend')
```

Adding node attributes:

```
G.add_node('A', role = 'trader')
G.add_node('B', role = 'trader')
G.add_node('C', role = 'manager')
```

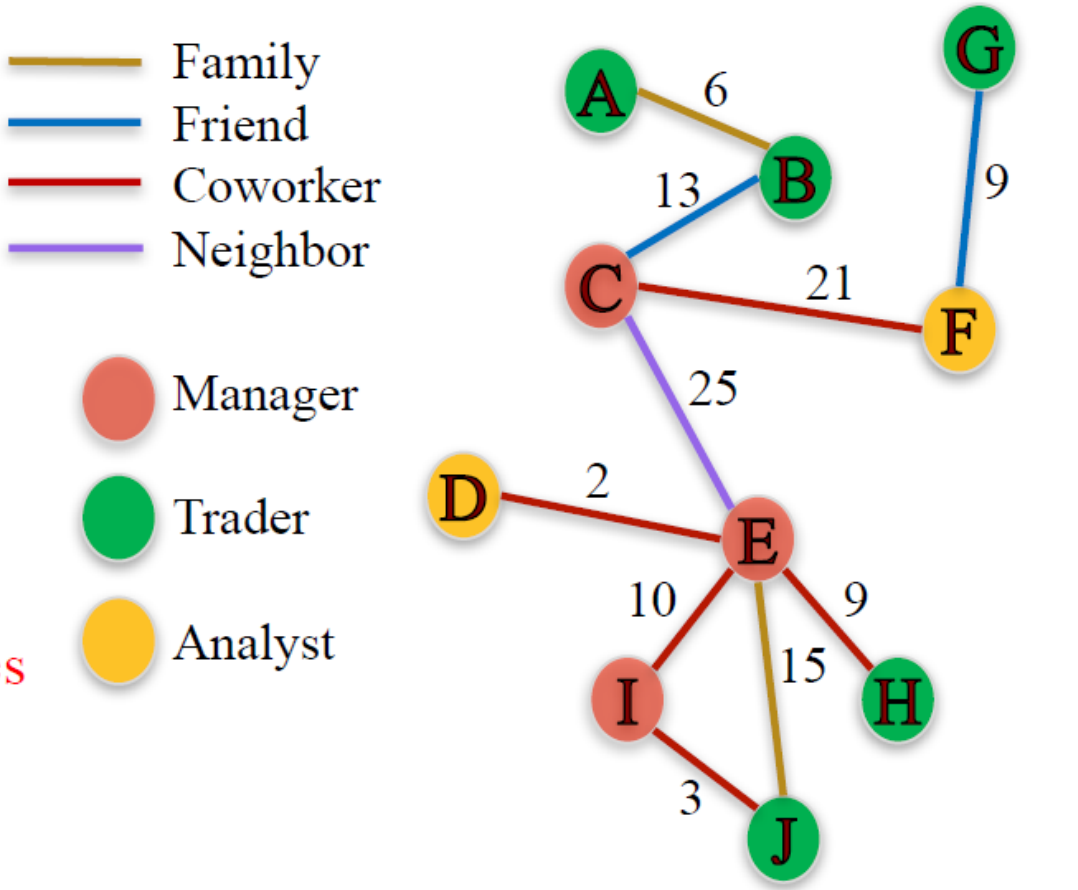


Number of times coworkers had lunch together in one year

```
G=nx.Graph()
G.add_edge('A','B', weight= 6, relation = 'family')
G.add_edge('B','C', weight= 13, relation = 'friend')
```

Accessing node attributes:

```
In: G.nodes() # list of all nodes
Out: ['A', 'C', 'B']
In: G.nodes(data= True) #list of all nodes with attributes
Out: [('A', {'role': 'trader'}), ('C', {'role': 'manager'})
, ('B', {'role': 'trader'})]
In: G.node['A']['role']
Out: 'manager'
```

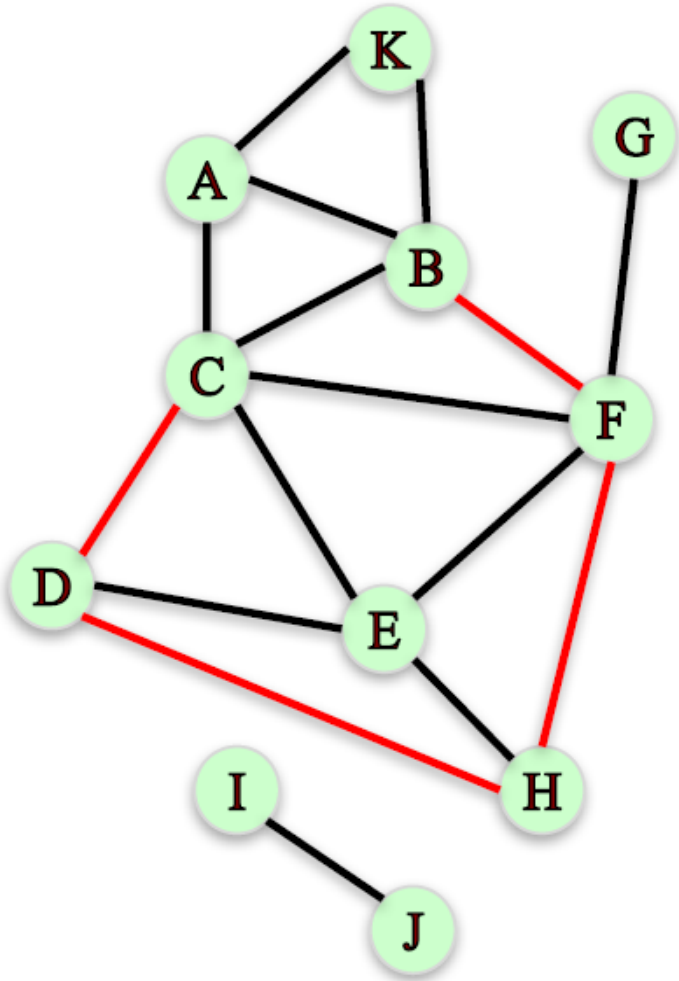


Number of times coworkers had lunch together in one year

Triadic Closure

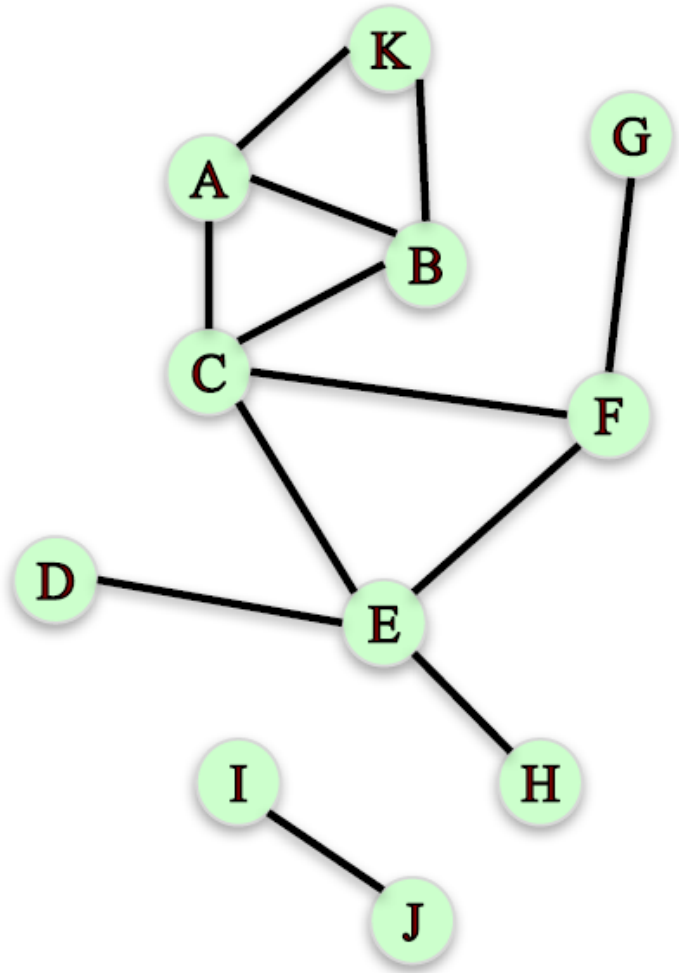
Triadic closure: The tendency for people who share connections in a social network to become connected.

How can we measure the prevalence of triadic closure in a network?



Local Clustering Coefficient

Local clustering coefficient of a node:
Fraction of pairs of the node's friends that are friends with each other.



Local Clustering Coefficient

Compute the local clustering coefficient of node C:

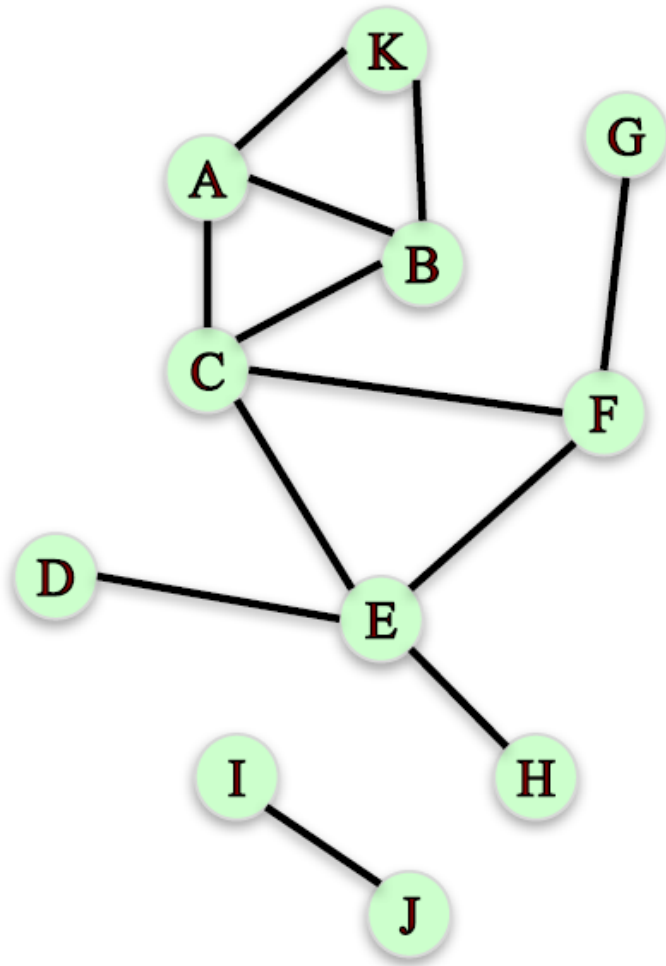
$$\frac{\#\text{ of pairs of C's friends who are friends}}{\#\text{ of pairs of C's friends}}$$

# of C's friends = $d_c = 4$ (the "degree" of C)

$$\#\text{ of pairs of C's friends} = \frac{d_c(d_c-1)}{2} = \frac{12}{2} = 6$$

# of pairs of C's friends who are friends = 2

$$\text{Local clustering coefficent of C} = \frac{2}{6} = \frac{1}{3}$$



Local clustering coefficent of F = $\frac{1}{3}$

Local Clustering Coefficient

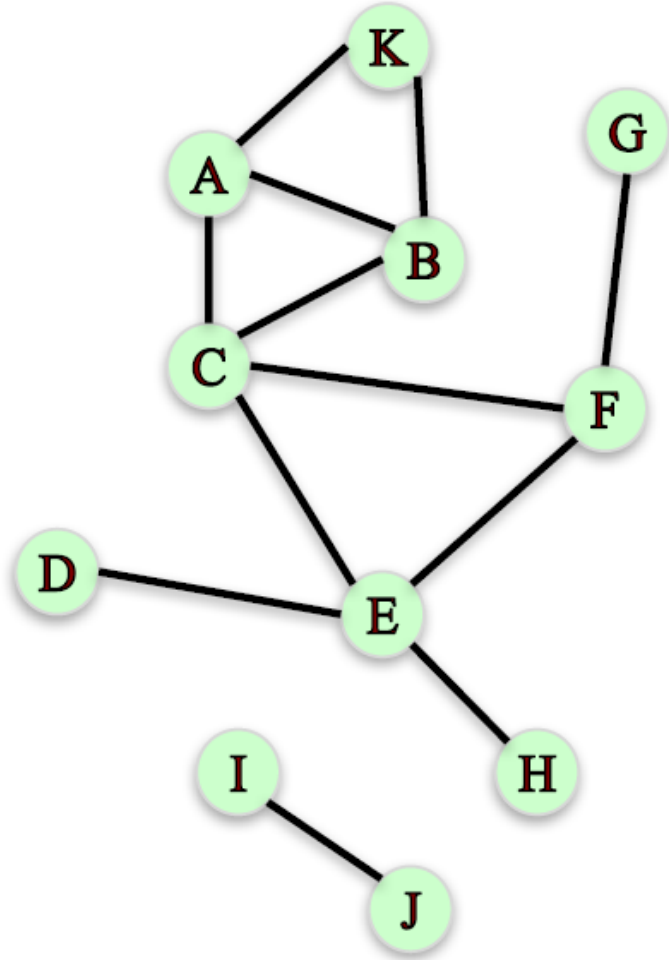
Local clustering coefficient in NetworkX:

```
G = nx.Graph()
G.add_edges_from([('A', 'K'), ('A', 'B'), ('A', 'C'), ('B', 'C'), ('B', 'K'),
('C', 'E'), ('C', 'F'), ('D', 'E'), ('E', 'F'), ('E', 'H'), ('F', 'G'), ('I', 'J')])
```

```
In: nx.clustering(G, 'F')
Out: 0.333333333333333
```

```
In: nx.clustering(G, 'A')
Out: 0.6666666666666666
```

```
In: nx.clustering(G, 'J')
Out: 0.0
```

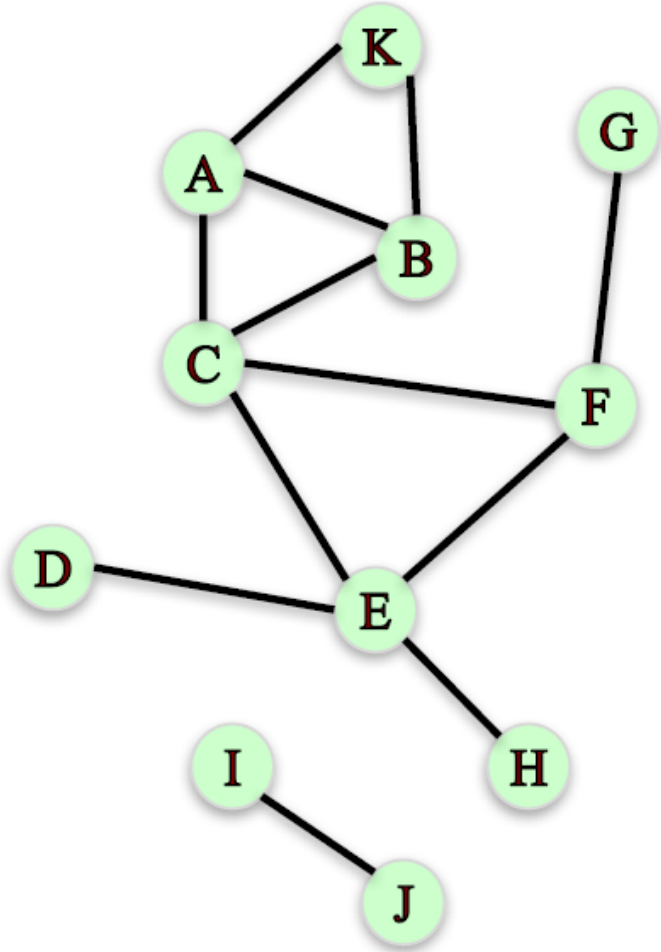


Global Clustering Coefficient

Measuring clustering on the whole network:

Approach 1: Average local clustering coefficient over all nodes in the graph.

In: nx.average_clustering(G)
Out: 0.28787878787878785



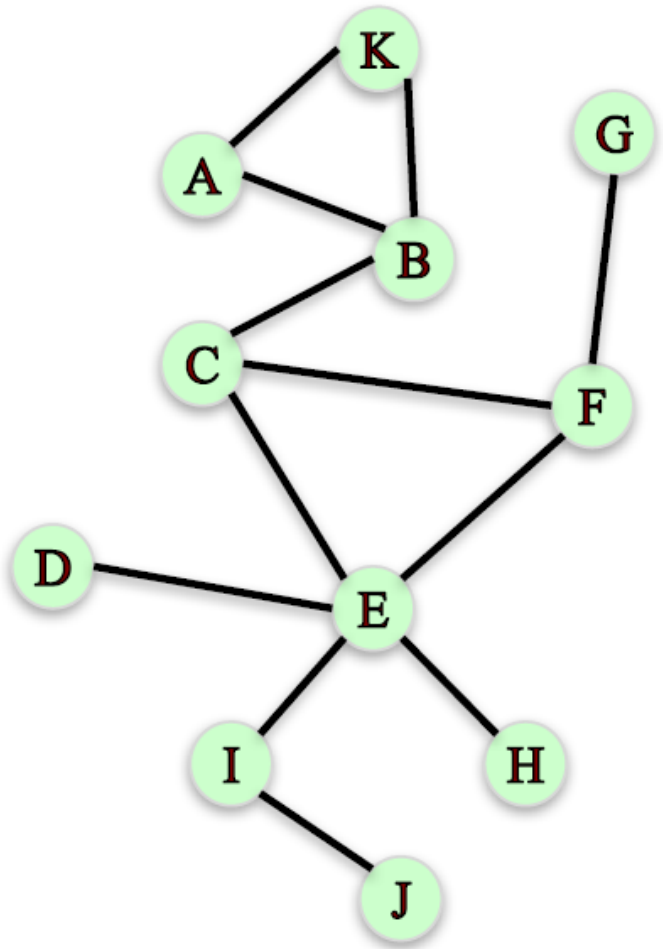
Distance

How "far" is node A from node H?

Are nodes far away or close to each other in this network?

Which nodes are "closest" and "farthest" to other nodes?

We need a sense of distance between nodes to answer these questions



Path: A sequence of nodes connected by an edge.

Find two paths from node G to node C:

G – F – C

G – F – E – C

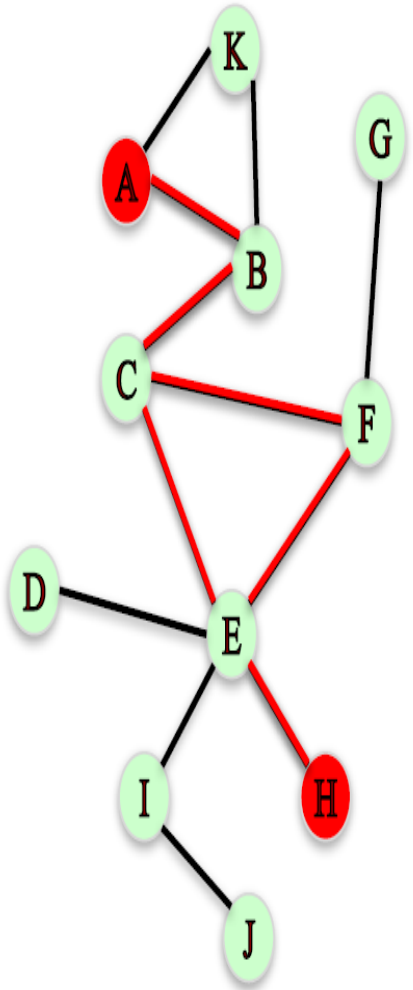
How far is node A from node H?

Path 1: A – B – C – E – H (4 "hops")

Path 2: A – B – C – F – E – H (5 "hops")

Path length: Number of steps it contains from beginning to end.

Path 1 has length 4, Path 2 has length 5

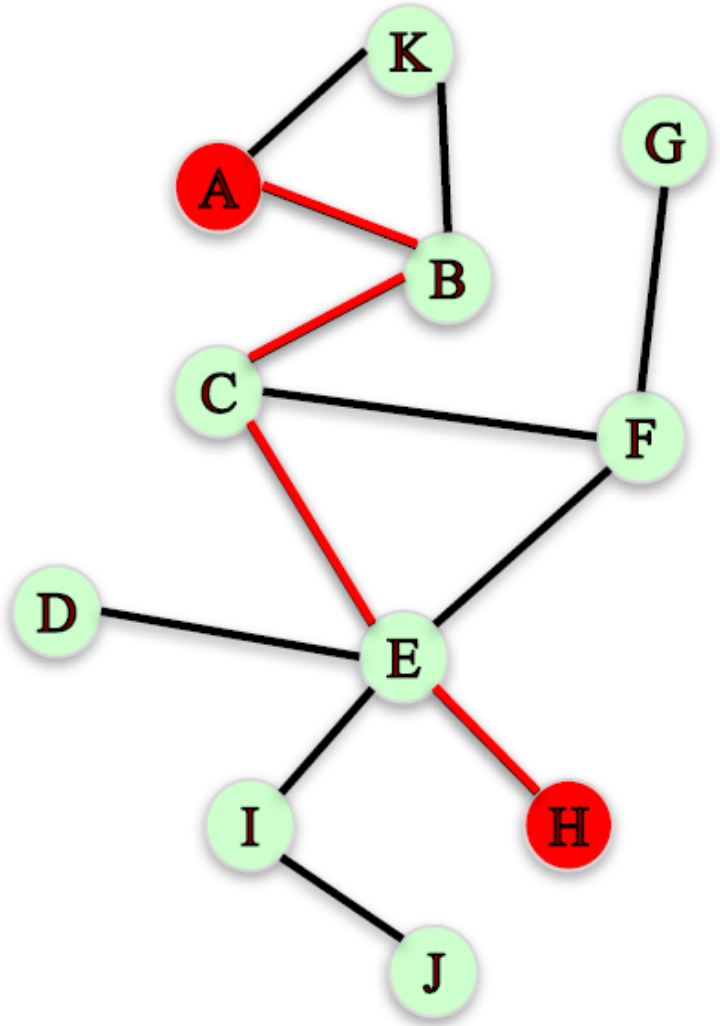


Distance between two nodes: the length of the shortest path between them.

The distance between node A and H is 4

In: nx.shortest_path(G,'A', 'H')
Out: ['A', 'B', 'C', 'E', 'H']

In: nx.shortest_path_length(G,'A', 'H')
Out: 4



Distance Measures

How to characterize the distance between all pairs of nodes in a graph?

Average distance between every pair of nodes.

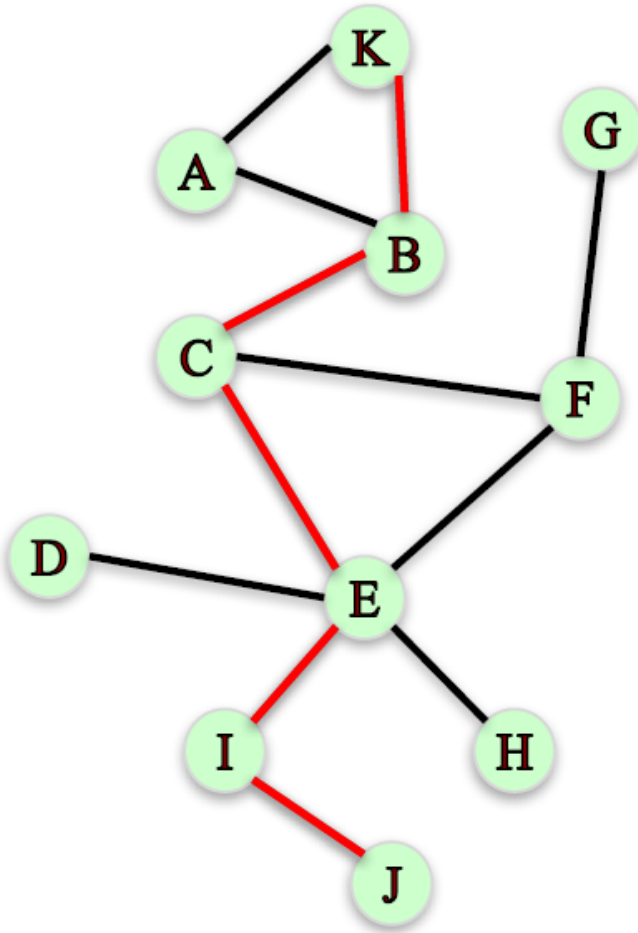
In: nx.average_shortest_path_length(G)

Out: 2.52727272727

Diameter: maximum distance between any pair of nodes.

In: nx.diameter(G)

Out: 5



How to summarize the distances between all pairs of nodes in a graph?

The **Eccentricity** of a node *n* is the largest distance between *n* and all other nodes.

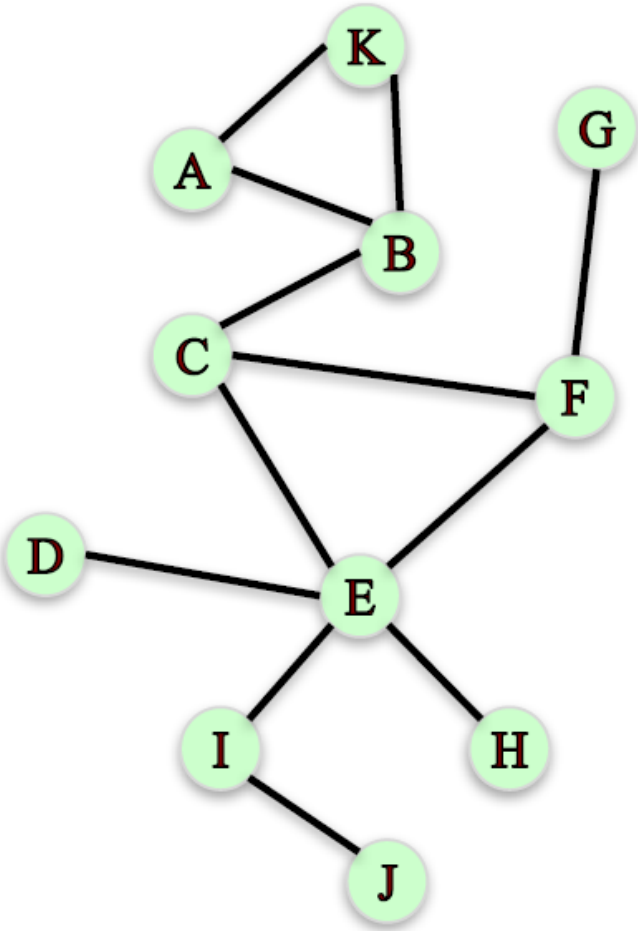
In: nx.eccentricity(G)

Out: {'A': 5, 'B': 4, 'C': 3, 'D': 4, 'E': 3, 'F': 3, 'G': 4, 'H': 4, 'I': 4, 'J': 5, 'K': 5}

The **radius** of a graph is the minimum eccentricity.

In: nx.radius(G)

Out: 3



How to summarize the distances between all pairs of nodes in a graph?

The **Periphery** of a graph is the set of nodes that have eccentricity equal to the diameter.

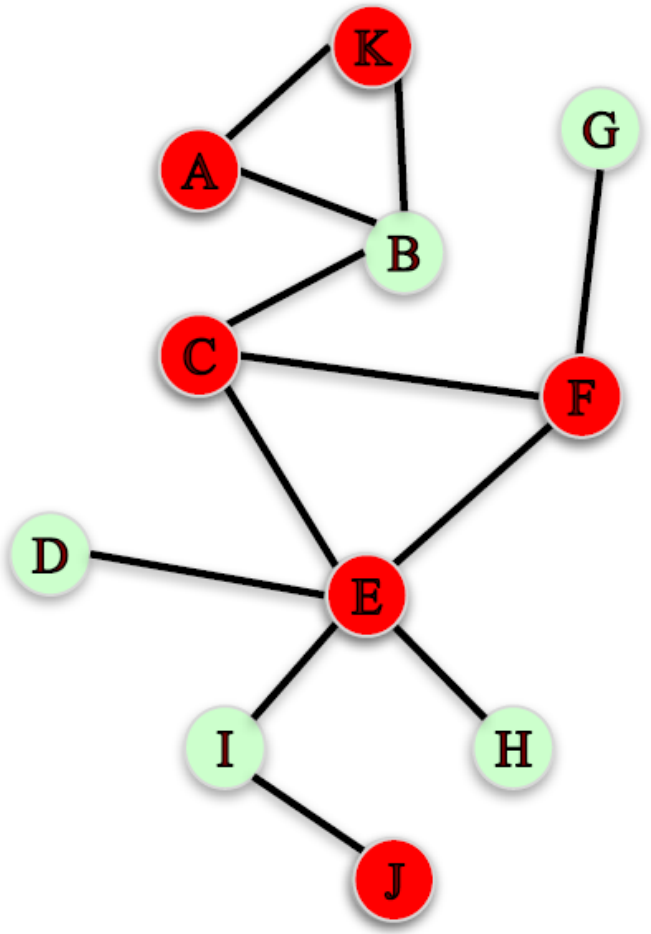
In: nx.periphery(G)

Out: ['A', 'K', 'J']

The **center** of a graph is the set of nodes that have eccentricity equal to the radius.

In: nx.center(G)

Out: ['C', 'E', 'F']



Karate Club Network

```
G = nx.karate_club_graph()
G = nx.convert_node_labels_to_integers(G,first_label=1)
```

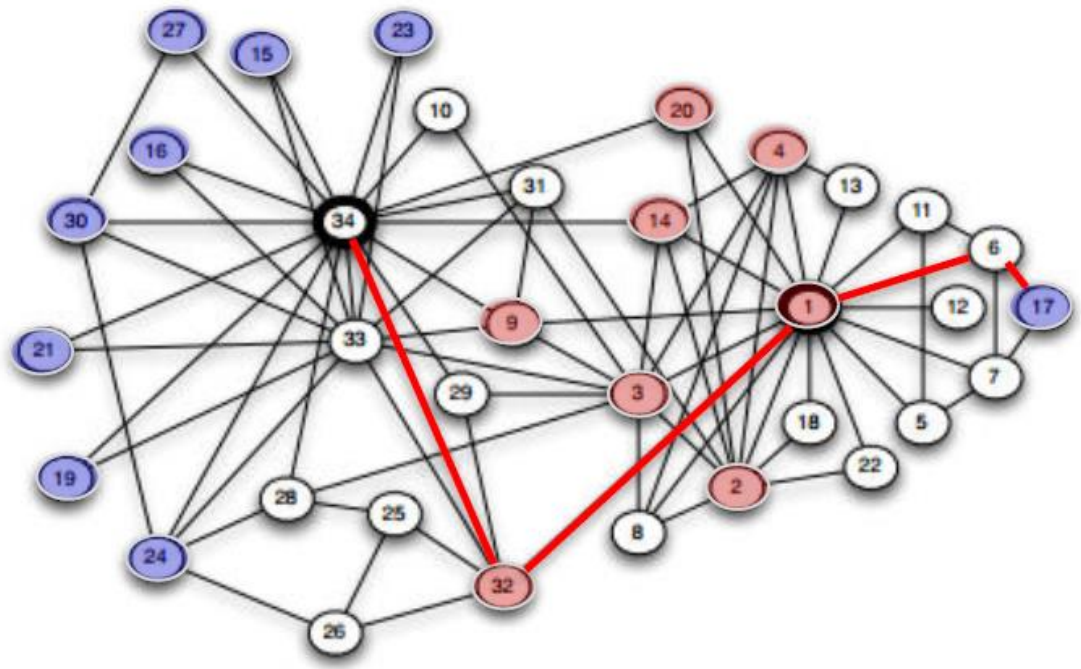
Average shortest path = 2.41

Radius = 3

Diameter = 5

Center = [1, 2, 3, 4, 9, 14, 20, 32]

Periphery: [15, 16, 17, 19, 21, 23, 24, 27, 30]



Friendship network in a 34-person karate club

Node 34 looks pretty "central". However, it has distance 4 to node 17

Connected Graphs

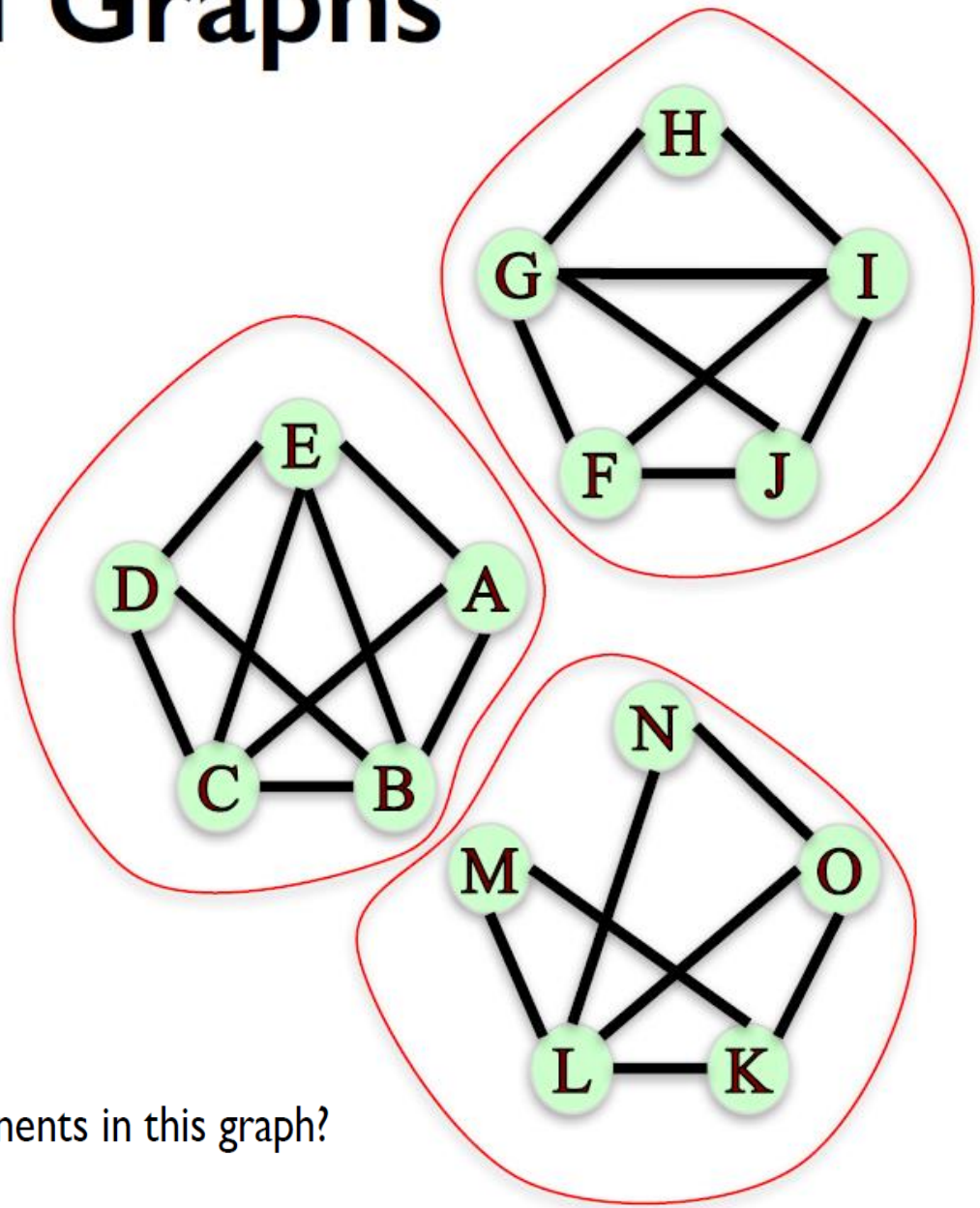
An undirected graph is **connected** if, for every pair nodes, there is a path between them.

In: nx.is_connected(G)

Out: True

However, if we remove edges A—G, A—N, and J—O, the graph becomes disconnected.

There is no path between nodes in the three different "communities".



What are the connected components in this graph?

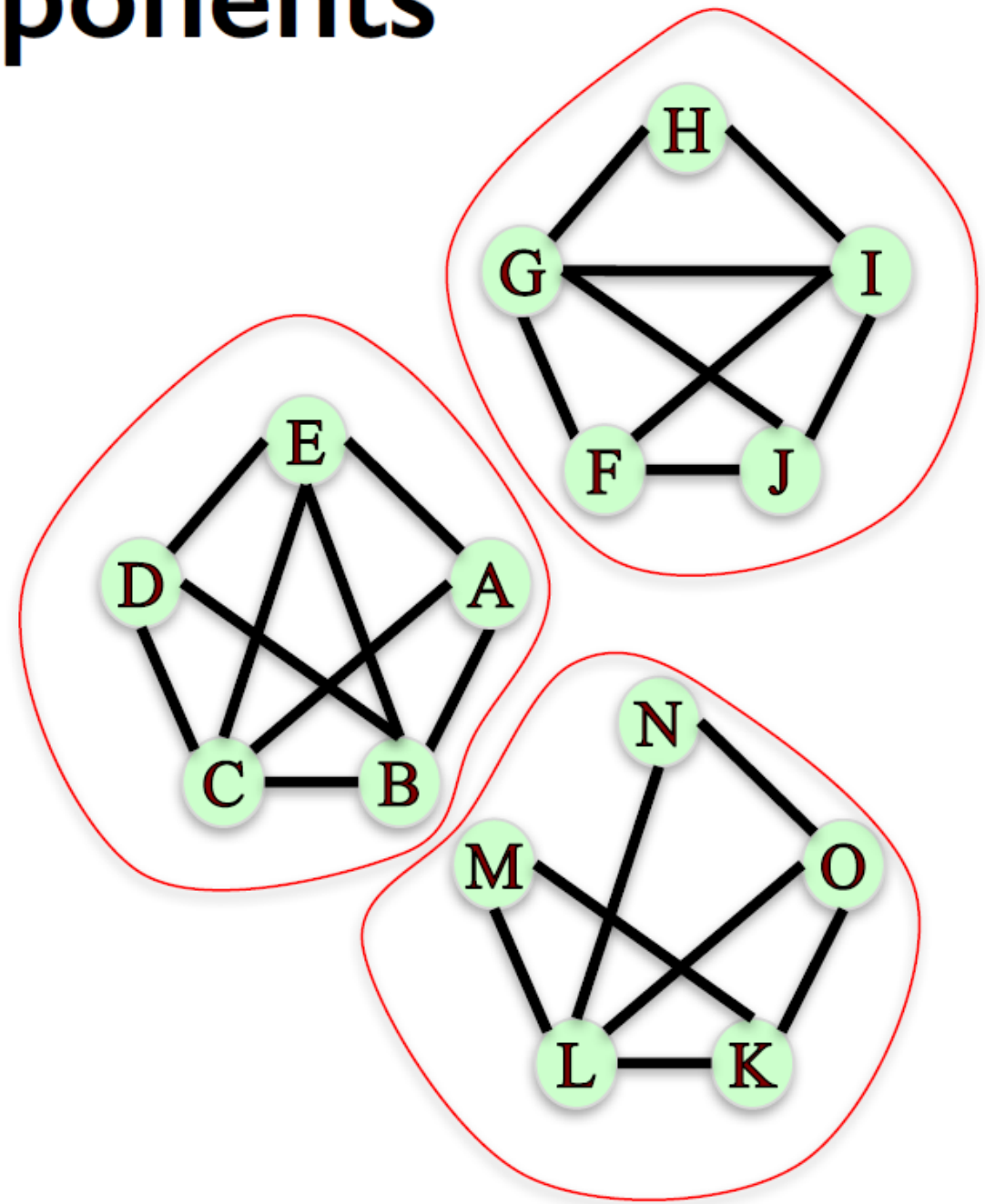
{A, B, C, D, E}, {F, G, H, I, J}, {K, L, M, N, O}

Graph Components

In: nx.number_connected_components(G)
Out: 3

In: sorted(nx.connected_components(G))
Out: [{'A', 'B', 'C', 'D', 'E'},
{'F', 'G', 'H', 'I', 'J'},
{'K', 'L', 'M', 'N', 'O'}]

In: nx.node_connected_component(G, 'M')
Out: {'K', 'L', 'M', 'N', 'O'}

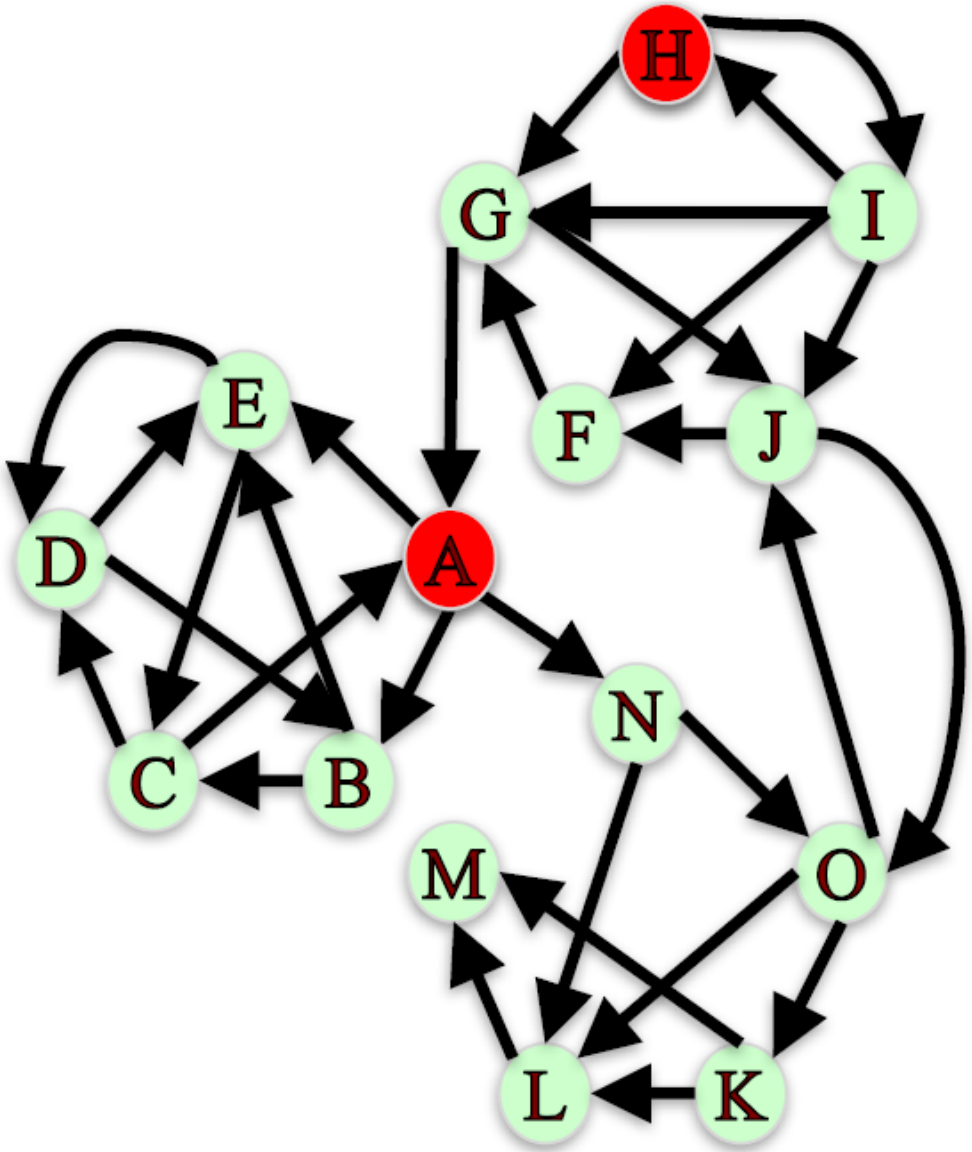


Connectivity in Directed Graphs

A directed graph is **strongly connected** if, for every pair nodes *u* and *v*, there is a directed path from u to v and a directed path from *v* to *u*.

In: nx.is_strongly_connected(G)
Out: False

Note: There is no directed path from A to H

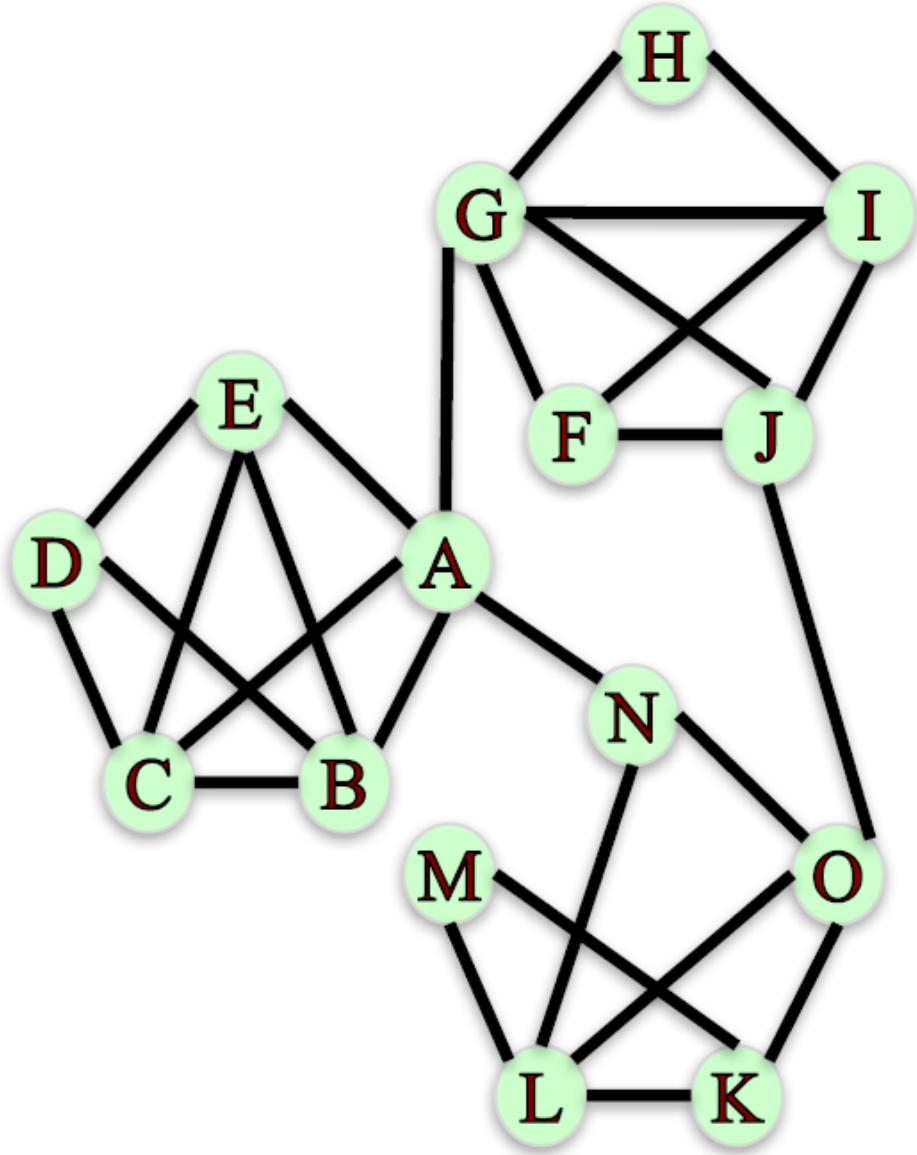


Connectivity in Directed Graphs

A directed graph is **weakly connected** if replacing all directed edges with undirected edges produces a connected undirected graph.

In: nx.is_weakly_connected(G)

Out: True



Connectivity in Directed Graphs

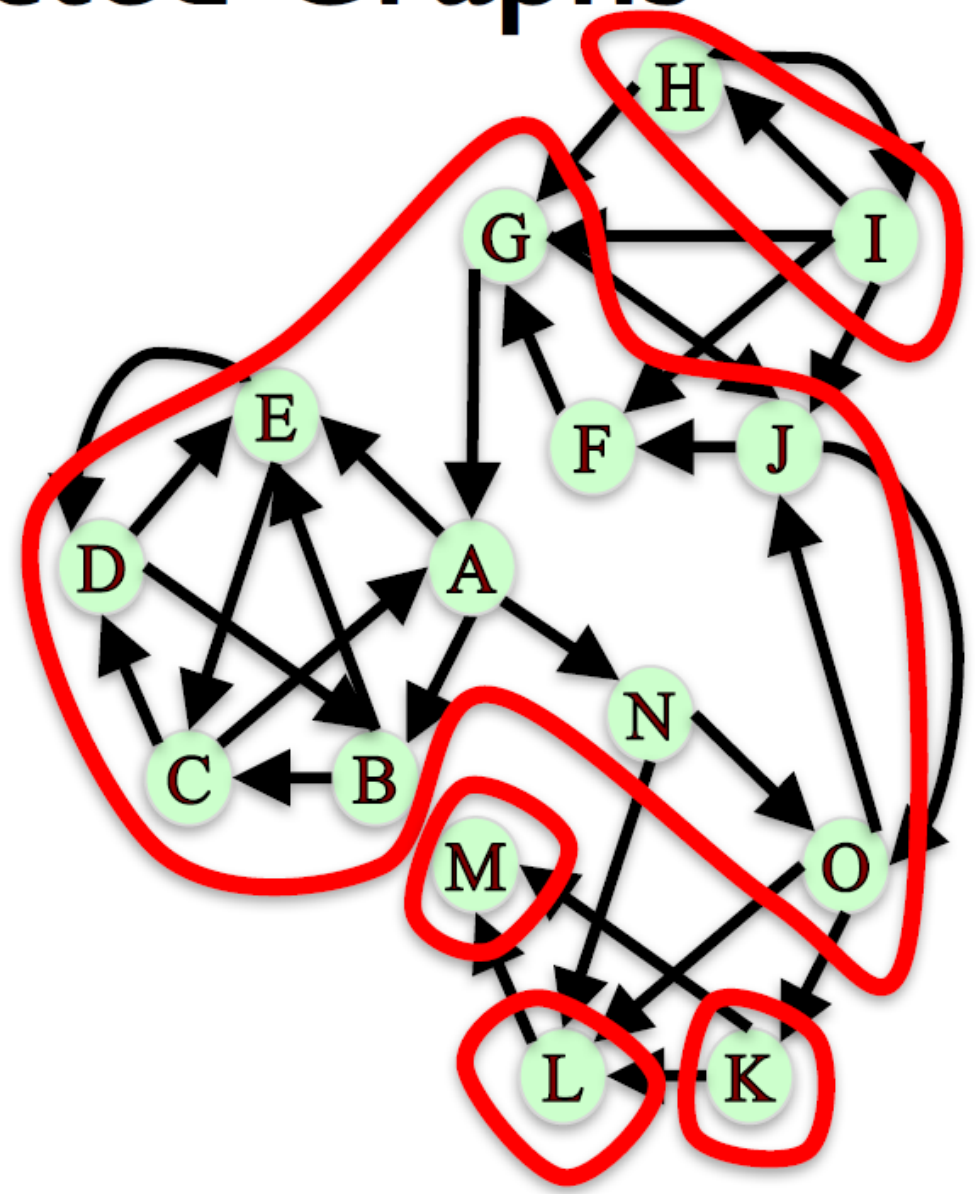
Strongly connected component:
A subset of nodes such as:

i. Every node in the subset has a **directed** path to every other node.
ii. No other node has a **directed** path to every node in the subset.

What are the strongly connected components in this graph?

In: sorted(nx.strongly_connected_components(G))

Out: [{M}, {L}, {K}, {A, B, C, D, E, F, G, J, N, O}, {H, I}]



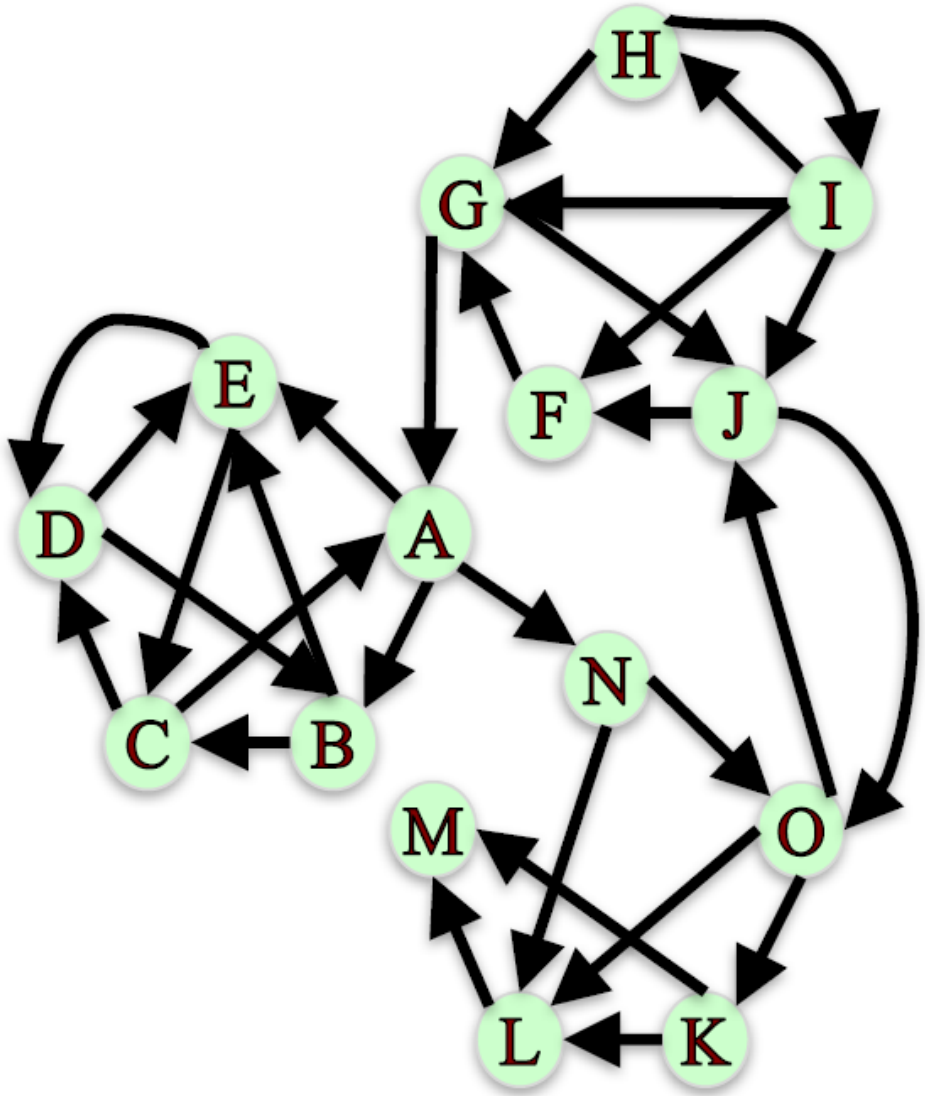
Connectivity in Directed Graphs

Weakly connected component:

The connected components of the graph after replacing all directed edges with undirected edges.

```
In: sorted(nx.weakly_connected_components(G))
Out: [{'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
'M', 'N', 'O'}]
```

Since the graph is weakly connected it only has one weakly connected component.



Connectivity and Robustness in Networks

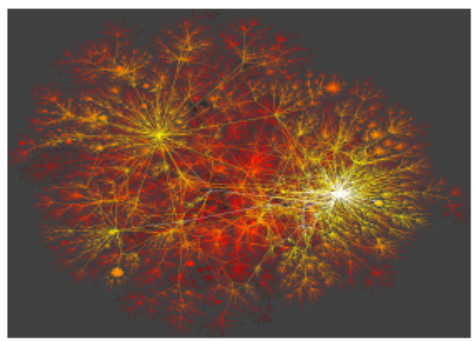
Network robustness: the ability of a network to maintain its general structural properties when it faces failures or attacks.

Type of attacks: removal of nodes or edges.

Structural properties: connectivity.

Examples: airport closures, internet router failures, power line failures.

Network of direct flights around the world [Bio.Diaspora]



Internet Connectivity [K. C. Claffy]

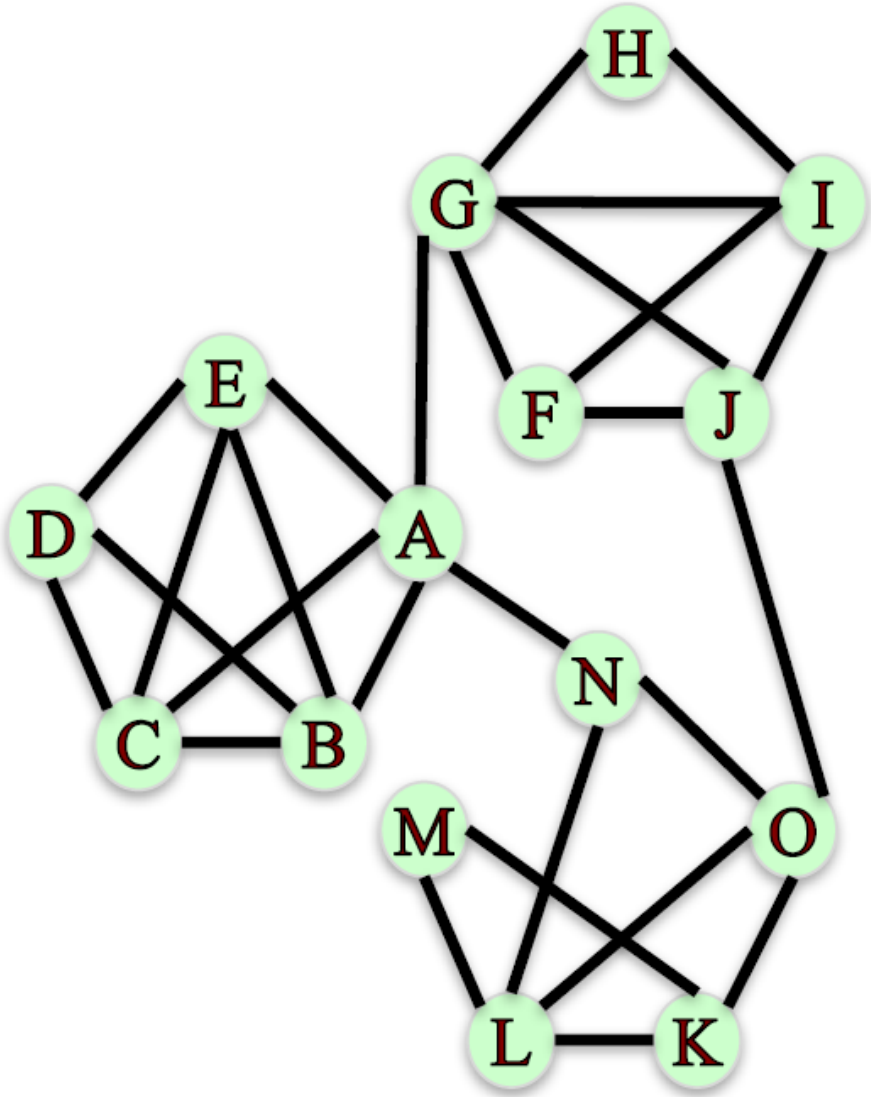
Disconnecting a Graph

What is the smallest number of nodes that can be removed from this graph in order to disconnect it?

```
In: nx.node_connectivity(G_un)
Out: 1
```

Which node?

```
In: nx.minimum_node_cut(G_un)
Out: {'A'}
```



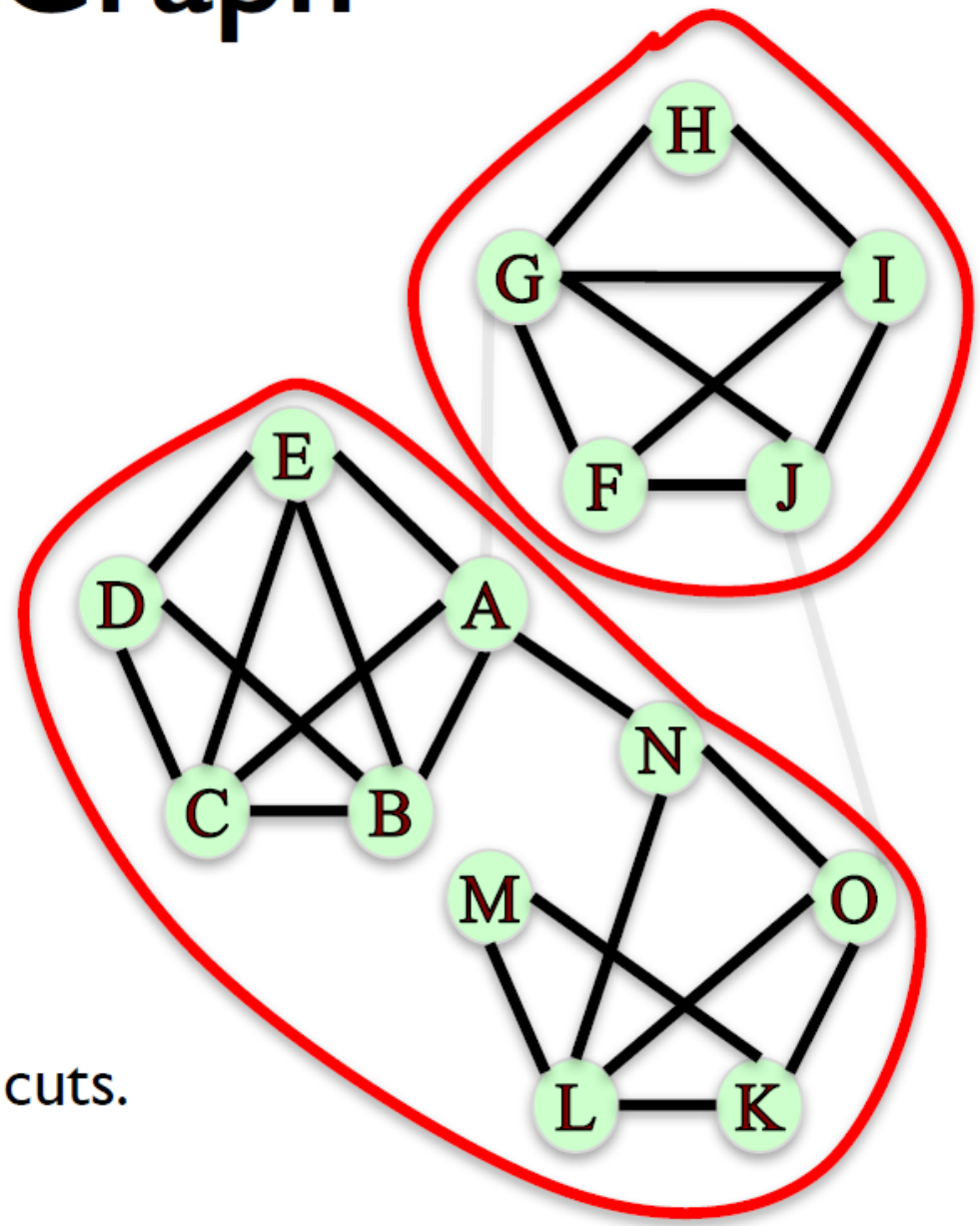
Disconnecting a Graph

What is the smallest number of **edges** that can be removed from this graph in order to disconnect it?

In: nx.edge_connectivity(G_un)
Out: 2

Which edges?
In: nx.minimum_edge_cut(G_un)
Out: {('A', 'G'), ('O', 'J')}

Robust networks have large minimum node and edge cuts.

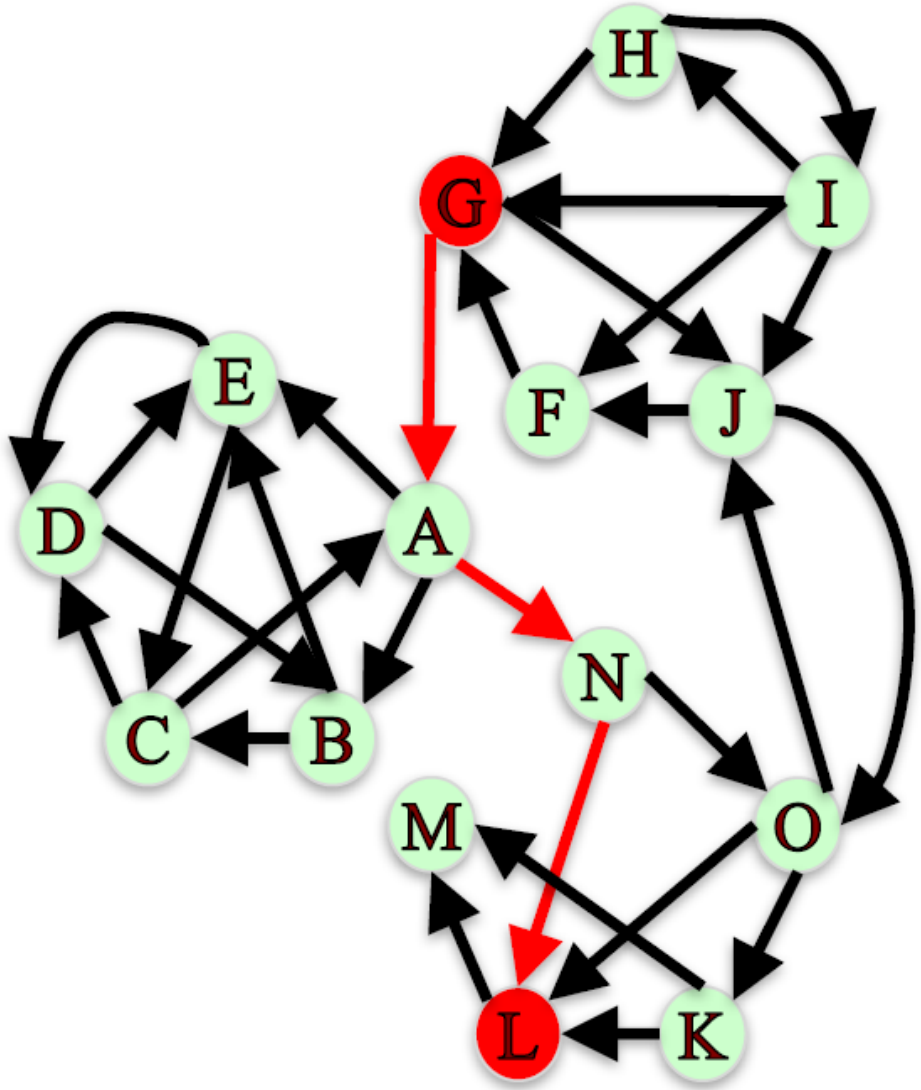


Simple Paths

Imagine node G wants to send a message to node L by passing it along to other nodes in this network.

What options does G have to deliver the message?

```
In: sorted(nx.all_simple_paths(G, 'G', 'L'))
Out: [['G', 'A', 'N', 'L'],
['G', 'A', 'N', 'O', 'K', 'L'],
['G', 'A', 'N', 'O', 'L'],
['G', 'J', 'O', 'K', 'L'],
['G', 'J', 'O', 'L']]
```



Node Connectivity

If we wanted to block the message from G to L by removing **nodes** from the network, how many nodes would we need to remove?

In: nx.node_connectivity(G, 'G', 'L')

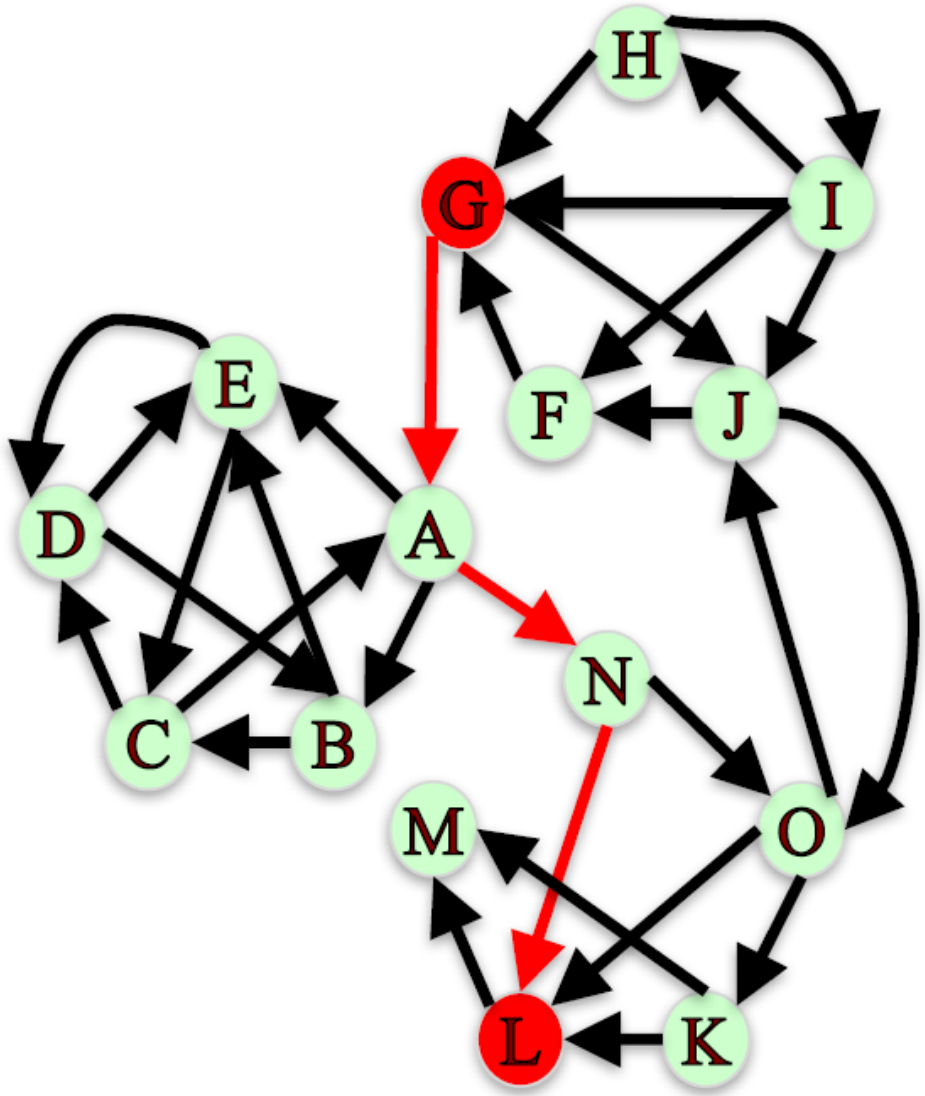
Out: 2

Which nodes?

In: nx.minimum_node_cut(G, 'G', 'L')

Out: {'N', 'O'}

If we only remove node O, message can go on path G → A → N → L.



Edge Connectivity

If we wanted to block the message from G to L by removing **edges** from the network, how many edges would we need to remove?

In: nx.edge_connectivity(G, 'G', 'L')

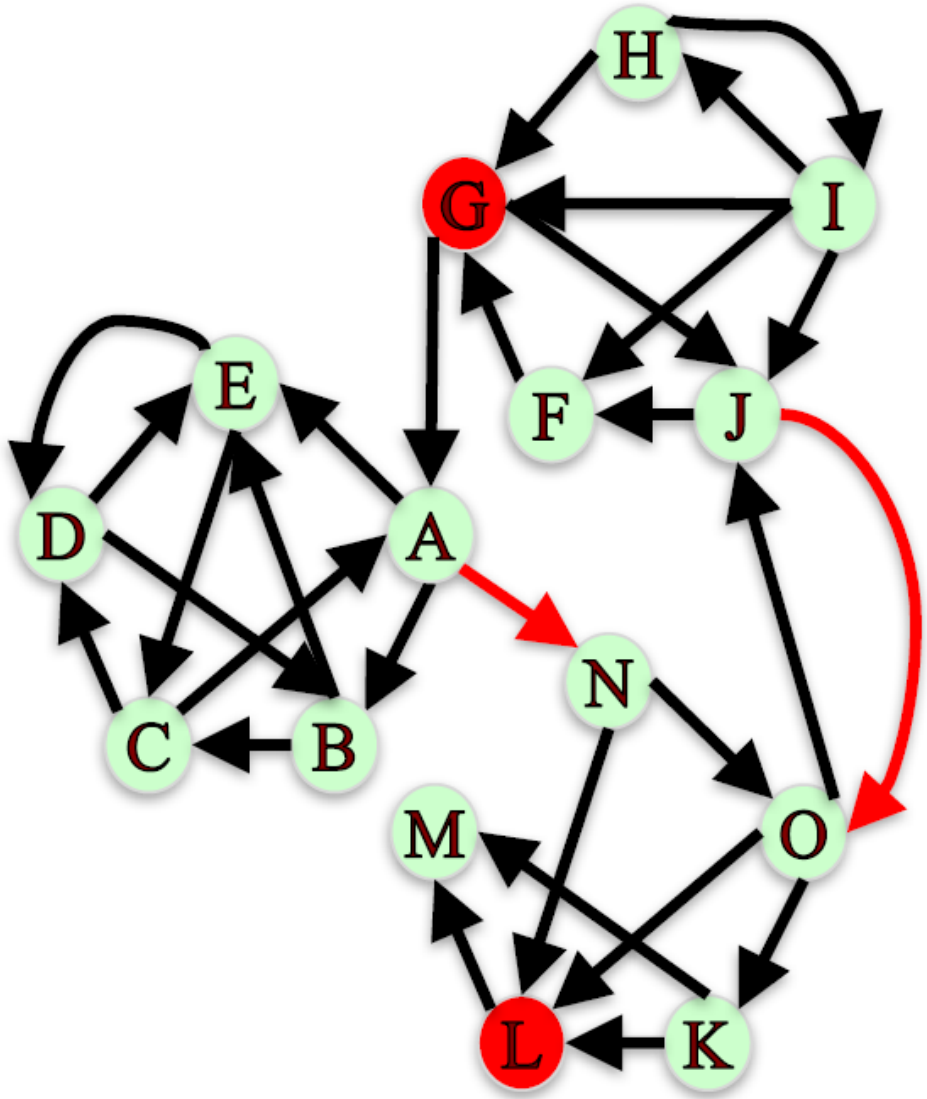
Out: 2

Which edges?

In: nx.minimum_edge_cut(G, 'G', 'L')

Out: {('A', 'N'), ('J', 'O')}

We need to remove A → N an J → O to block messages from G to L.



Network Centrality

Centrality measures identify the most important nodes in a network:

- Influential nodes in a social network.
- Nodes that disseminate information to many nodes or prevent epidemics.
- Hubs in a transportation network.
- Important pages on the Web.
- Nodes that prevent the network from breaking up.

- **Degree centrality**
- **Closeness centrality**
- Betweenness centrality
- Load centrality
- Page Rank
- Katz centrality
- Percolation centrality

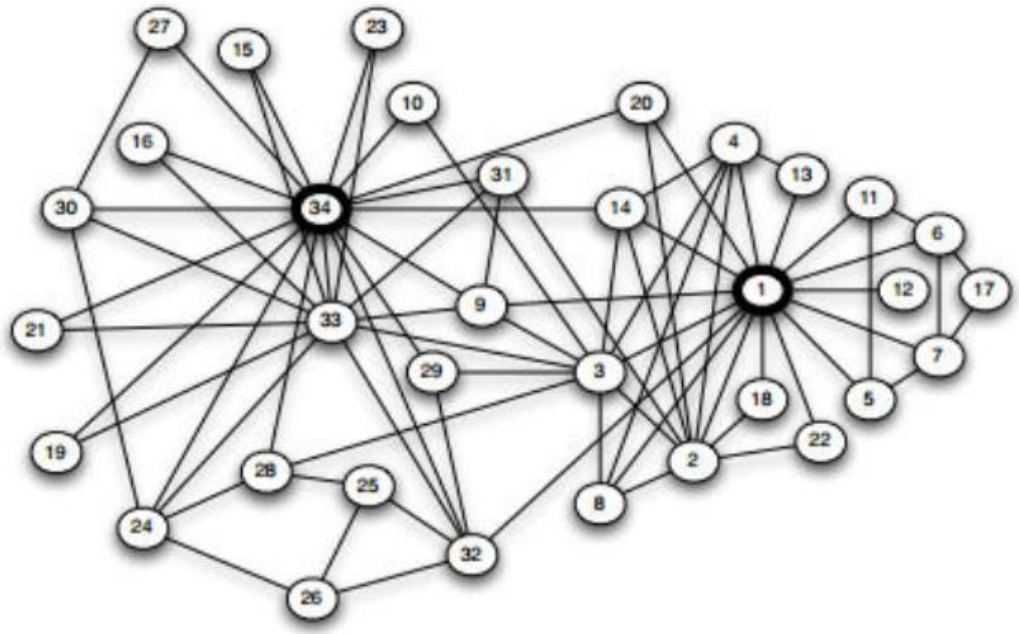
Degree Centrality

Assumption: important nodes have many connections.

The most basic measure of centrality: number of neighbors.

Undirected networks: use degree

Directed networks: use in-degree or out-degree

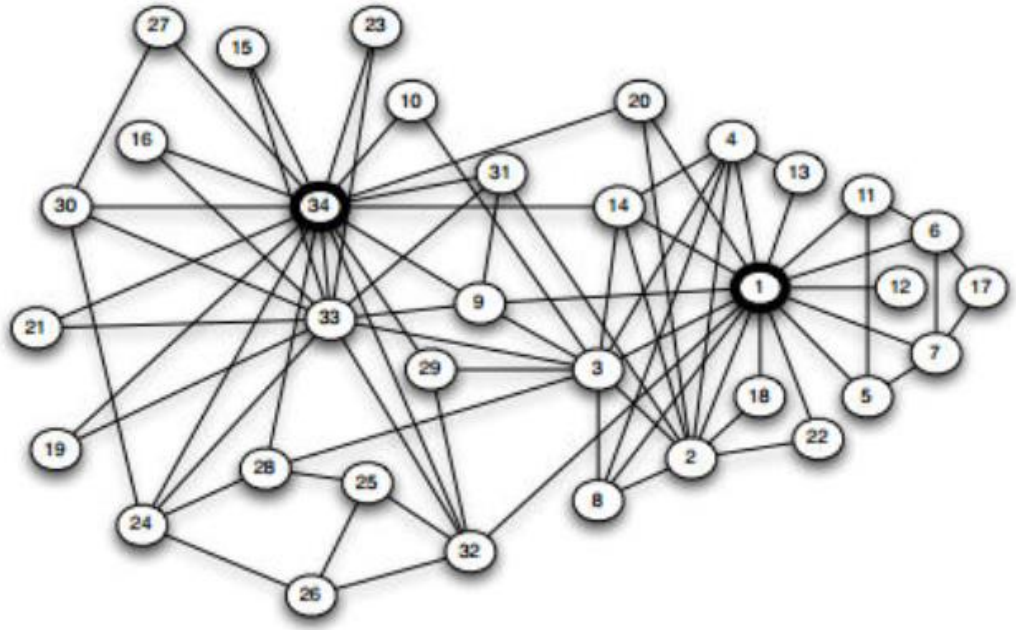


Friendship network in a 34-person karate club
[Zachary 1977]

Degree Centrality – Undirected Networks

$C_{deg}(v) = \frac{d_v}{|N|-1}$, where N is the set of nodes in the network and d_v is the degree of node v.

In: G = nx.karate_club_graph()
In: G = nx.convert_node_labels_to_integers(G,first_label=1)
In: degCent = nx.degree_centrality(G)
In: degCent[34]
Out: 0.515 # 17/33
In: degCent[33]
Out: 0.182 # 6/33



Friendship network in a 34-person karate club [Zachary 1977]

Degree Centrality – Directed Networks

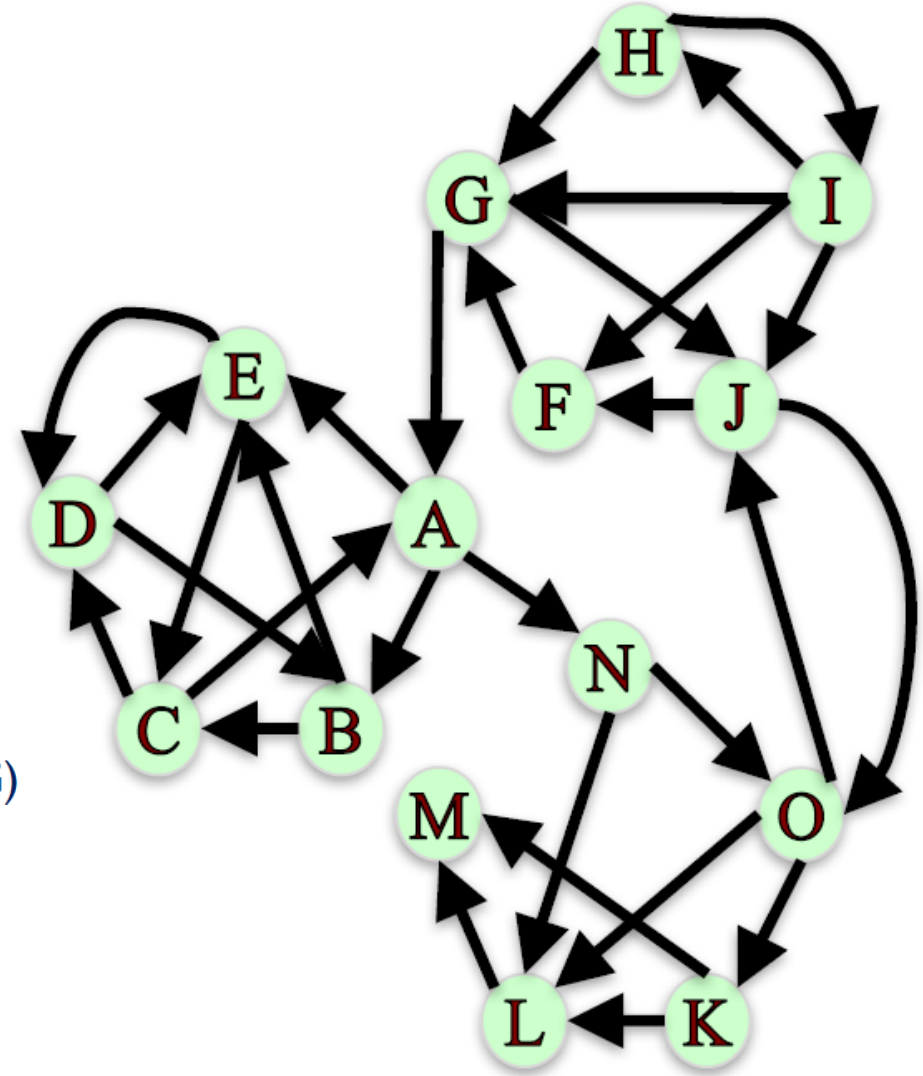
$C_{indeg}(v) = \frac{d_v^{in}}{|N|-1}$, where

N = set of nodes in the network,

d_v^{in} = the in-degree of node v.

In: indegCent = nx.in_degree_centrality(G)
In: indegCent['A']
Out: 0.143 # 2/14
In: indegCent['L']
Out: 0.214 # 3/14

In: outdegCent = nx.out_degree_centrality(G)
In: outdegCent['A']
Out: 0.214 # 3/14
In: outdegCent['L']
Out: 0.071 # 1/14



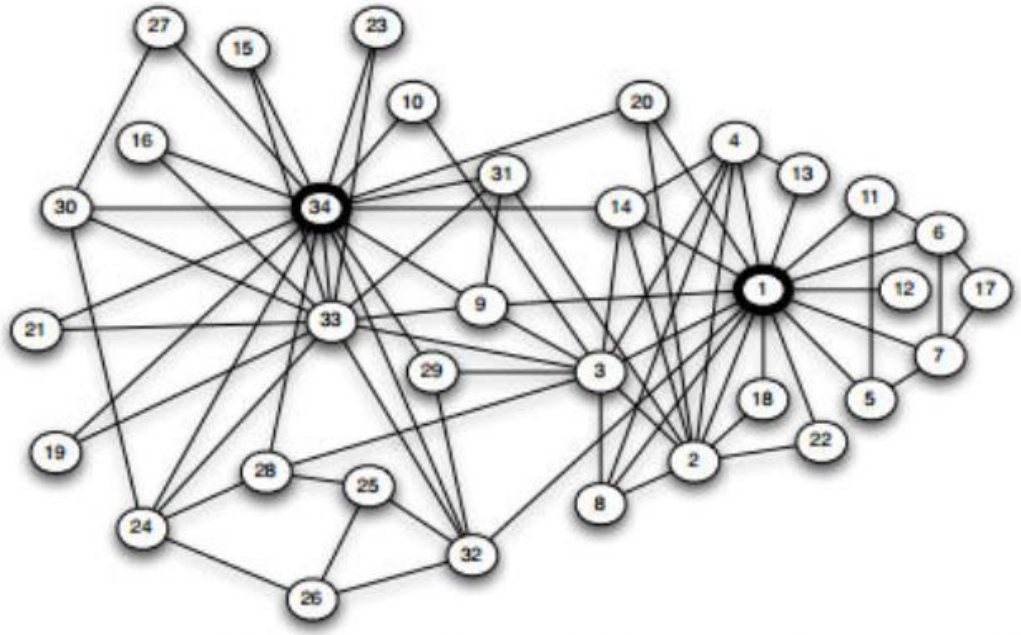
Closeness Centrality

Assumption: important nodes are close to other nodes.

$$C_{close}(v) = \frac{|N|-1}{\sum_{u \in N \setminus \{v\}} d(v,u)}, \text{ where}$$

N = set of nodes in the network,

$d(v,u)$ = length of shortest path from v to u.



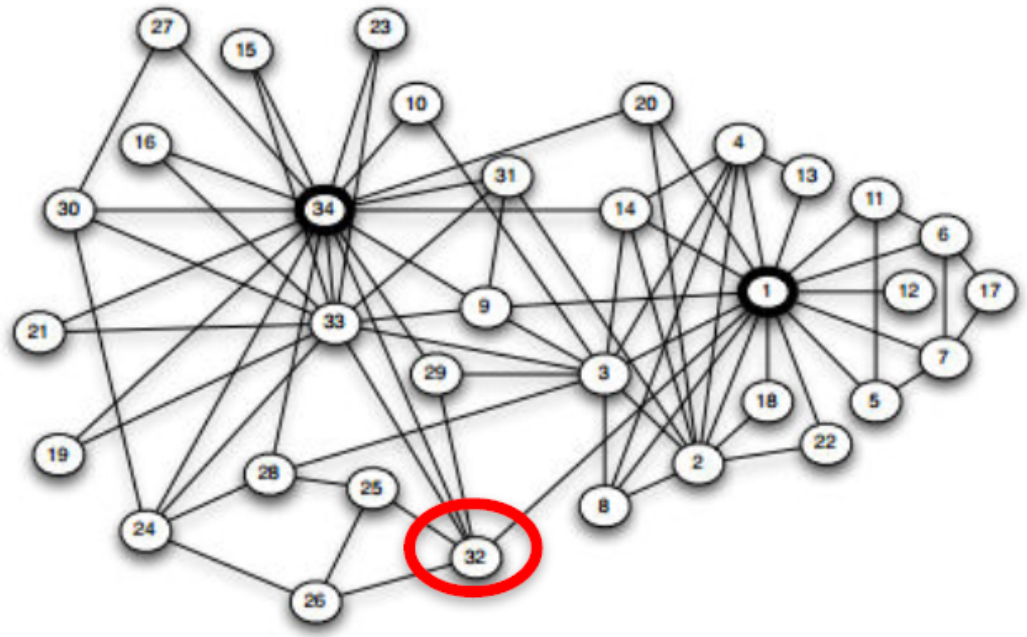
Friendship network in a 34-person karate club
[Zachary 1977]

Closeness Centrality

Assumption: important nodes are close to other nodes.

In: closeCent = nx.closeness_centrality(G)
In: closeCent[32]
Out: 0.541

In: sum(nx.shortest_path_length(G,32).values())
Out: 61
In: (len(G.nodes())-1)/61.
Out: 0.541



Friendship network in a 34-person karate club
[Zachary 1977]

Betweenness Centrality

Assumption: important nodes connect other nodes.

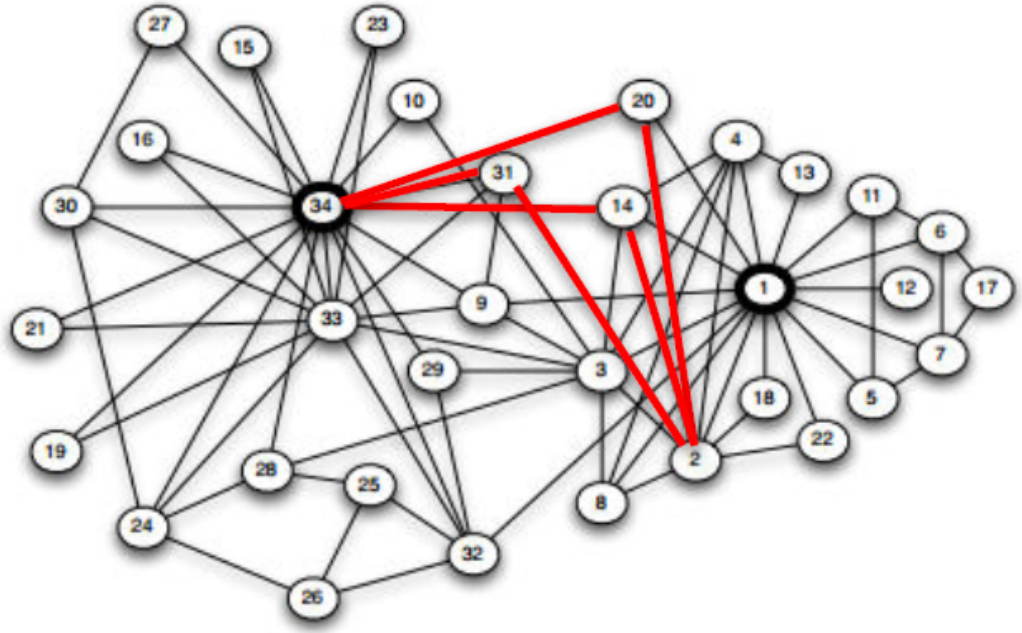
Recall: the distance between two nodes is the length of the shortest path between them.

Ex. The distance between nodes 34 and 2 is 2:

Path 1: 34 – 31 – 2

Path 2: 34 – 14 – 2

Path 3: 34 – 20 – 2



Friendship network in a 34-person karate club [Zachary 1977]

Nodes 31, 14, and 20 are in a shortest path of between nodes 34 and 2.

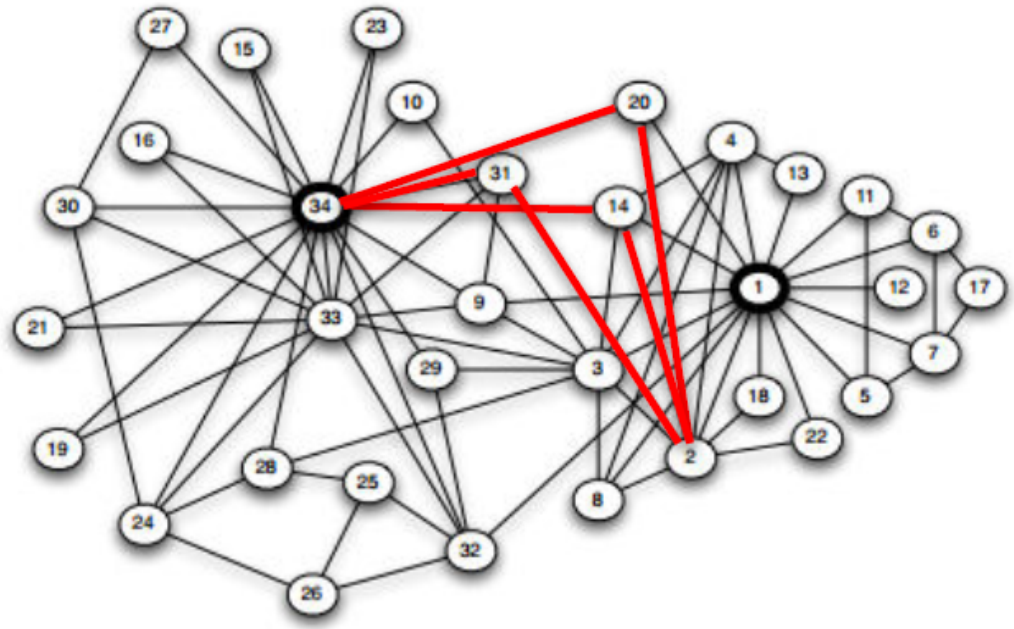
Betweenness Centrality

Assumption: important nodes connect other nodes.

$C_{btw}(v) = \sum_{s,t \in N} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}$, where

$\sigma_{s,t}$ = the number of shortest paths between nodes s and t.

$\sigma_{s,t}(v)$ = the number shortest paths between nodes s and t *that pass through node* v.

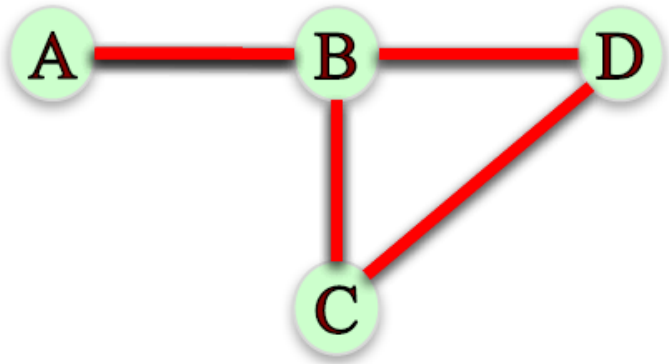


Friendship network in a 34-person karate club
[Zachary 1977]

Betweenness Centrality

Assumption: important nodes connect other nodes.

$$C_{btw}(v) = \sum_{s,t \in N} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}$$



Endpoints: we can either include or exclude node v as node s and t in the computation of $C_{btw}(v)$.

Ex. If we exclude node v, we have:

$$C_{btw}(B) = \frac{\sigma_{A,D}(B)}{\sigma_{A,D}} + \frac{\sigma_{A,C}(B)}{\sigma_{A,C}} + \frac{\sigma_{C,D}(B)}{\sigma_{C,D}} = \frac{1}{1} + \frac{1}{1} + \frac{0}{1} = 2$$

If we include node v, we have:

$$C_{btw}(B) = \frac{\sigma_{A,B}(B)}{\sigma_{A,B}} + \frac{\sigma_{A,C}(B)}{\sigma_{A,C}} + \frac{\sigma_{A,D}(B)}{\sigma_{A,D}} + \frac{\sigma_{B,C}(B)}{\sigma_{B,C}} + \frac{\sigma_{B,D}(B)}{\sigma_{B,D}} + \frac{\sigma_{C,D}(B)}{\sigma_{C,D}} = \frac{1}{1} + \frac{1}{1} + \frac{1}{1} + \frac{1}{1} + \frac{1}{1} + \frac{0}{1} = 5$$

Betweenness Centrality – Normalization

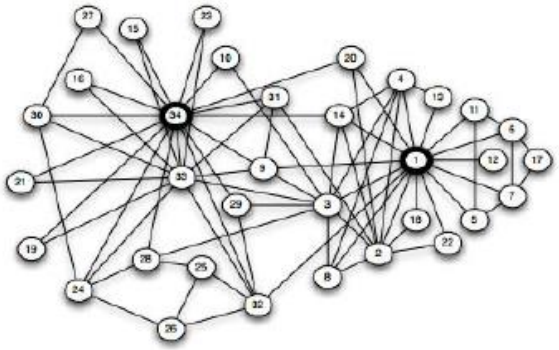
Assumption: important nodes connect other nodes.

$$C_{btw}(v) = \sum_{s,t \in N} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}$$

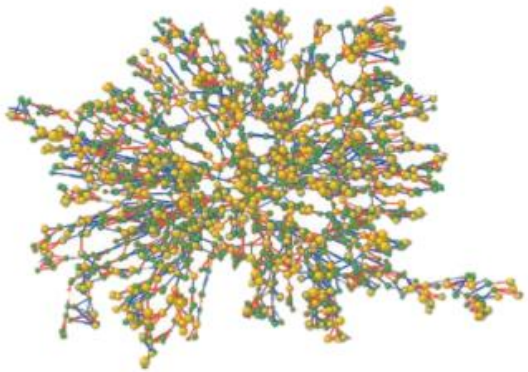
Normalization: betwenness centrality values will be larger in graphs with many nodes. To control for this, we divide centrality values by the number of pairs of nodes in the graph (excluding v):

$\frac{1}{2}(|N|-1)(|N|-2)$ in undirected graphs

$(|N|-1)(|N|-2)$ in directed graphs



Friendship network in a 34-person karate club [Zachary 1977]



Network of friendship, marital tie, and family tie among 2200 people [Christakis & Fowler 2007]

Betweenness Centrality

In: btwnCent = nx.betweenness_centrality(G, **normalized** = True, **endpoints** = False)

In: import operator

In: sorted(btwnCent.items(), key=operator.itemgetter(1), reverse = True)[0:5]

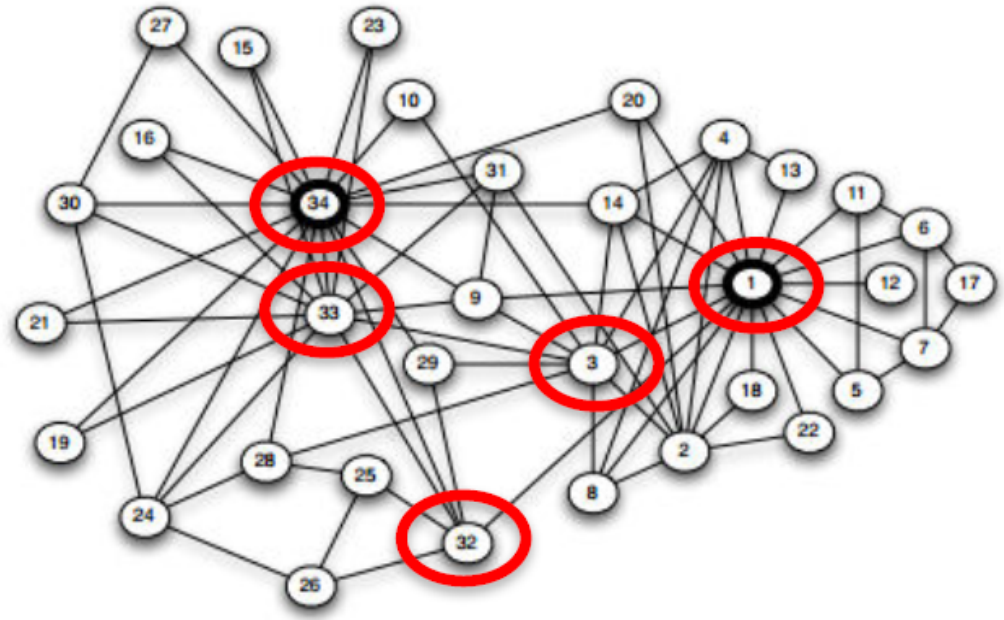
Out: [(1, 0.4376352813852814),

(34, 0.30407497594997596),

(33, 0.14524711399711399),

(3, 0.1436568061568061 8),

(32, 0.13827561327561325)]



Friendship network in a 34-person karate club [Zachary 1977]

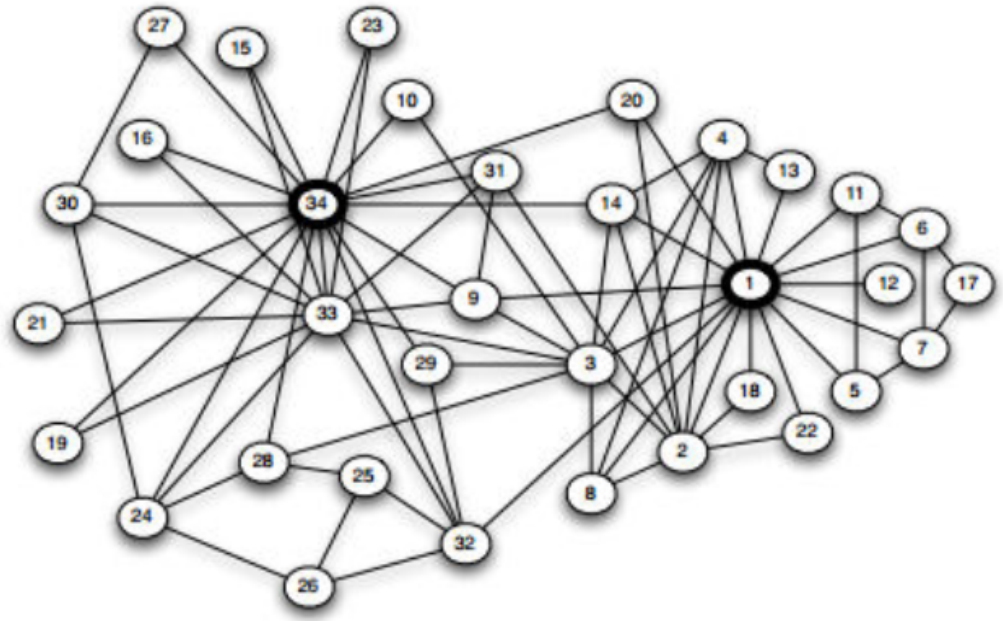
Betweenness Centrality – Edges

We can use betweenness centrality to find important edges instead of nodes:

$C_{btw}(e) = \sum_{s,t \in N} \frac{\sigma_{s,t}(e)}{\sigma_{s,t}}$, where

$\sigma_{s,t}$ = the number of shortest paths between nodes s and t.

$\sigma_{s,t}(e)$ = the number shortest paths between nodes s and t *that pass through edge* e.



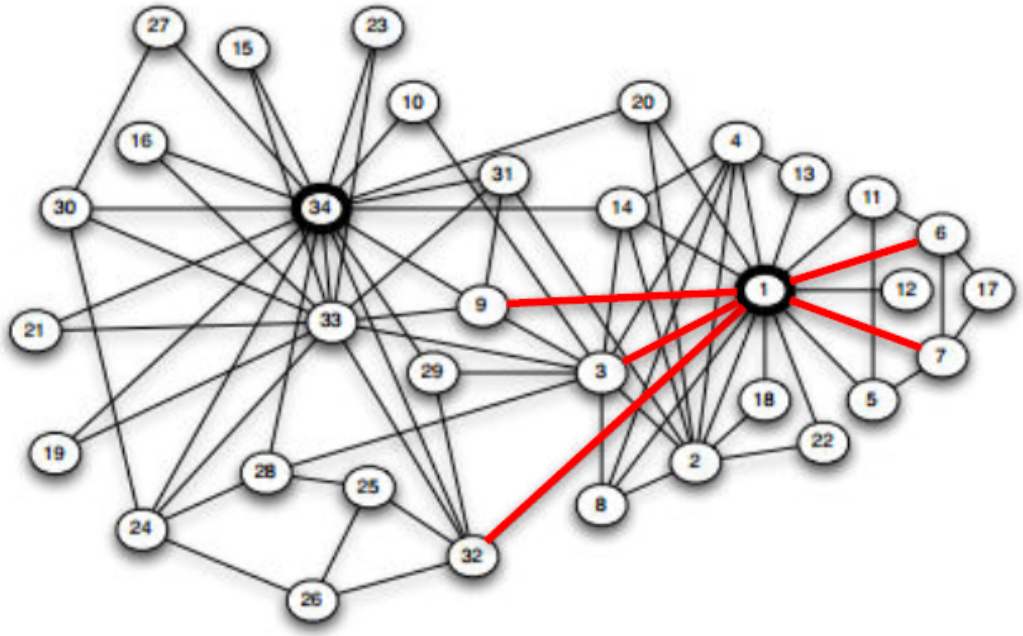
Friendship network in a 34-person karate club [Zachary 1977]

Betweenness Centrality – Edges

In: btwnCent_edge =
nx.edge_betweenness_centrality(G,
normalized=True)

In: sorted(btwnCent_edge.items(),
key=operator.itemgetter(1), reverse = True)[0:5]

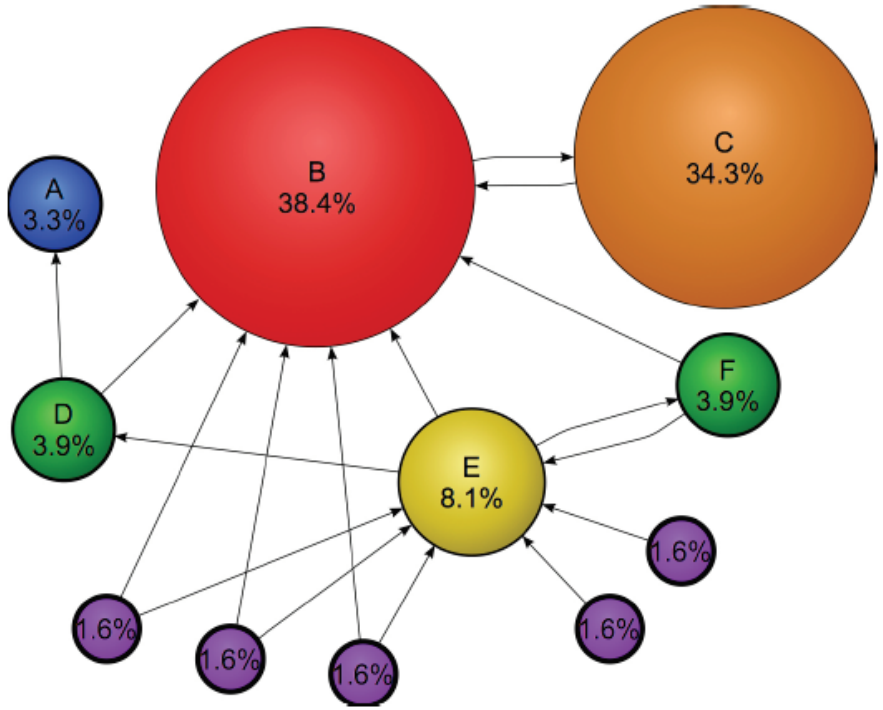
Out: [((1, 32), 0.12725999490705373),
((1, 7), 0.07813428401663694),
((1, 6), 0.07813428401663694),
((1, 3), 0.07778768072885 72),
((1, 9), 0.07423959482783014)]



Friendship network in a 34-person karate club
[Zachary 1977]

PageRank

Developed by Google founders to measure the importance of webpages from the hyperlink network structure.



PageRank assigns a score of importance to each node. Important nodes are those with many in-links from important pages.

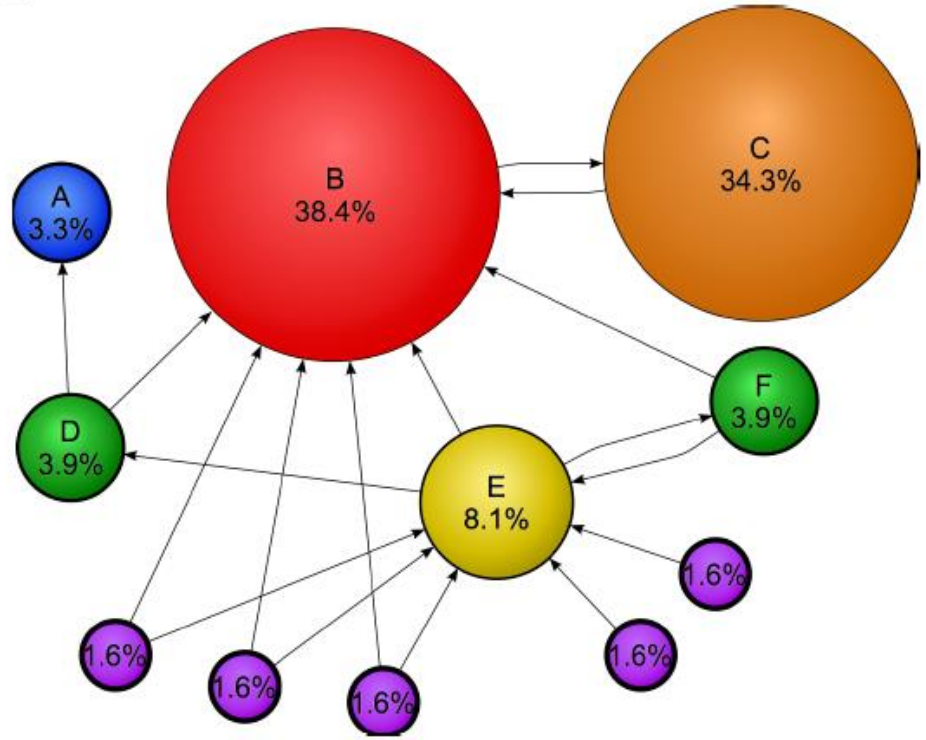
PageRank can be used for any type of network, but it is mainly useful for directed networks.

A node's PageRank depends on the PageRank of other nodes (Circular definition?).

PageRank

n = number of nodes in the network
k = number of steps

1. Assign all nodes a PageRank of $1/n$
2. Perform the *Basic PageRank Update Rule* k times.



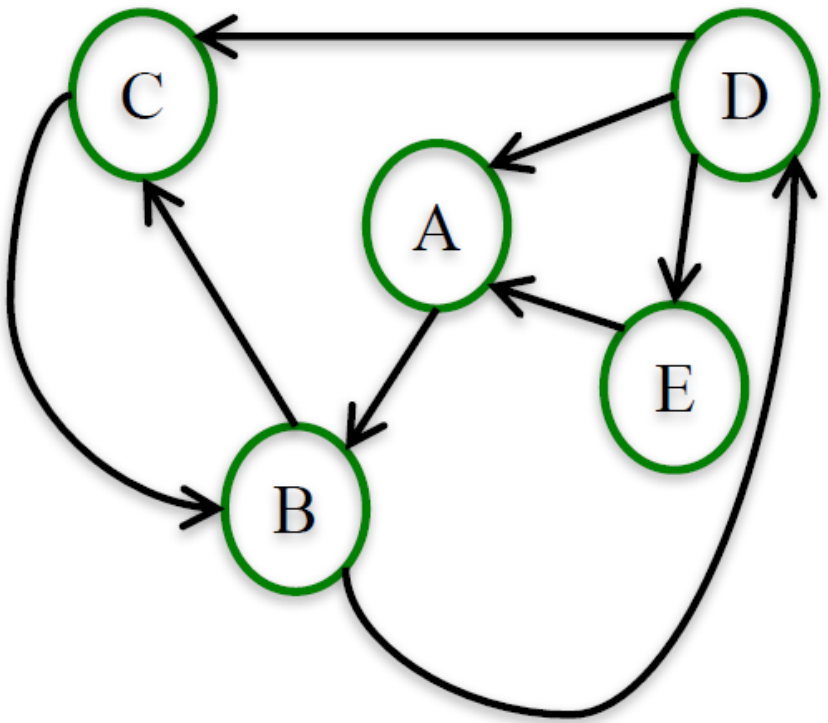
Basic PageRank Update Rule: Each node gives an equal share of its current PageRank to all the nodes it links to.

The new PageRank of each node is the sum of all the PageRank it received from other nodes.

PageRank

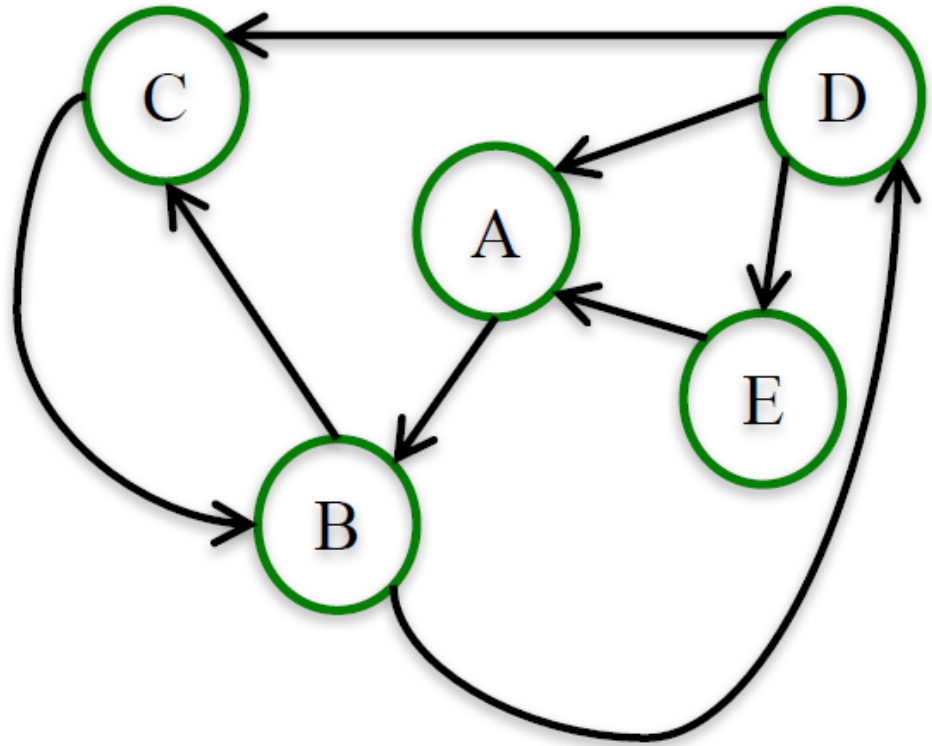
Who should be the most "important" node in this network?

Calculate the PageRank of each node after 2 steps of the procedure (k = 2).



PageRank – Step 1

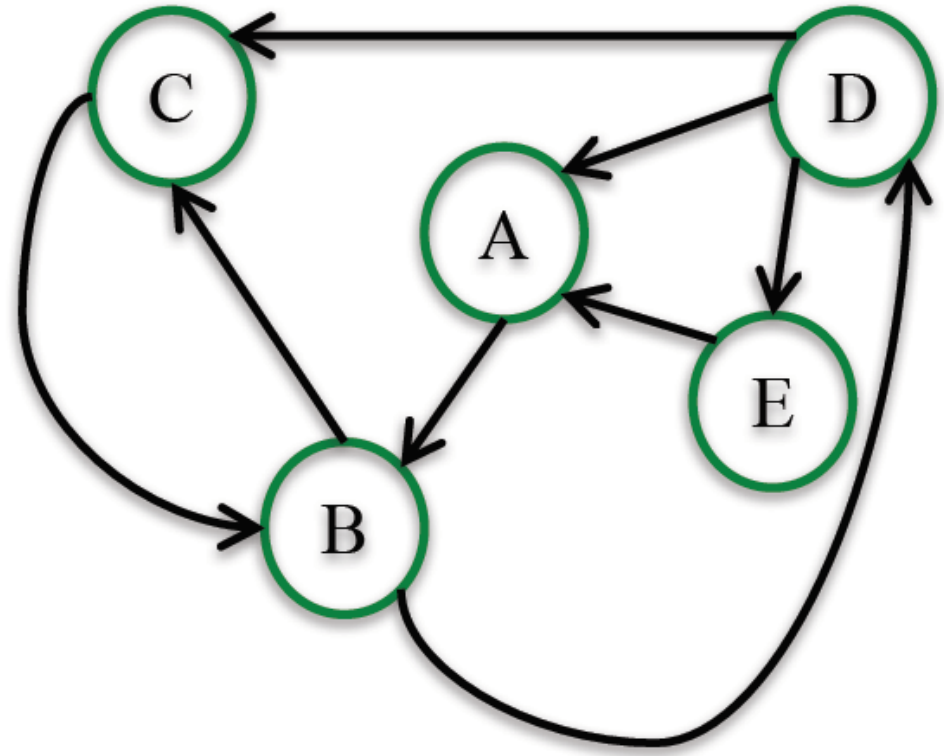
Page Rank (k = 1)					
	A	B	C	D	E
Old	1/5	1/5	1/5	1/5	1/5
New	4/15	2/5	1/6	1/10	1/15



A: (1/3)*(1/5) + 1/5 = 4/15
B: 1/5 + 1/5 = 2/5
C: (1/3)*(1/5) + (1/2)*(1/5) = 5/30 = 1/6
D: (1/2)*(1/5) = 1/10
E: (1/3)*(1/5) = 1/15

PageRank – Step 2

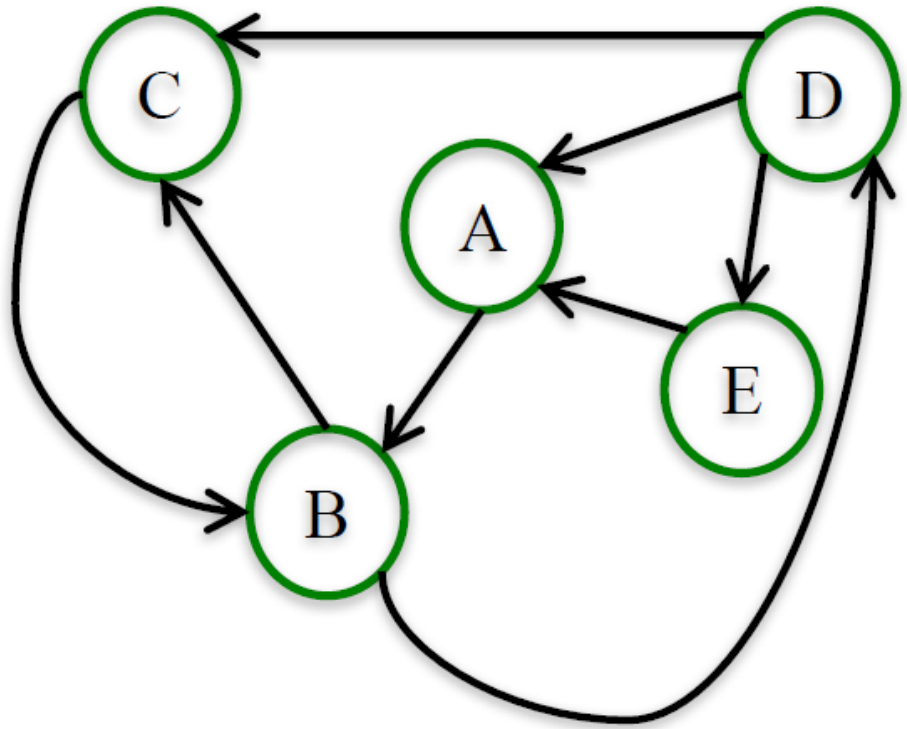
Page Rank (k = 2)					
	A	B	C	D	E
Old	4/15	2/5	1/6	1/10	1/15
New	1/10	13/30	7/30	2/10	1/30



A: (1/3)*(1/10) + 1/15 = 1/10
B: 1/6 + 4/15 = 13/30
C: (1/3)*(1/10) + (1/2)*(2/5) = 7/30
D: (1/2)*(2/5) = 2/10
E: (1/3)*(1/10) = 1/30

PageRank

	Page Rank				
	A	B	C	D	E
k=2	1/10	13/30	7/30	2/10	1/30
k=2	.1	.43	.23	.20	.03
k=3	.1	.33	.28	.22	.06
k=∞	.12	.38	.25	.19	.06

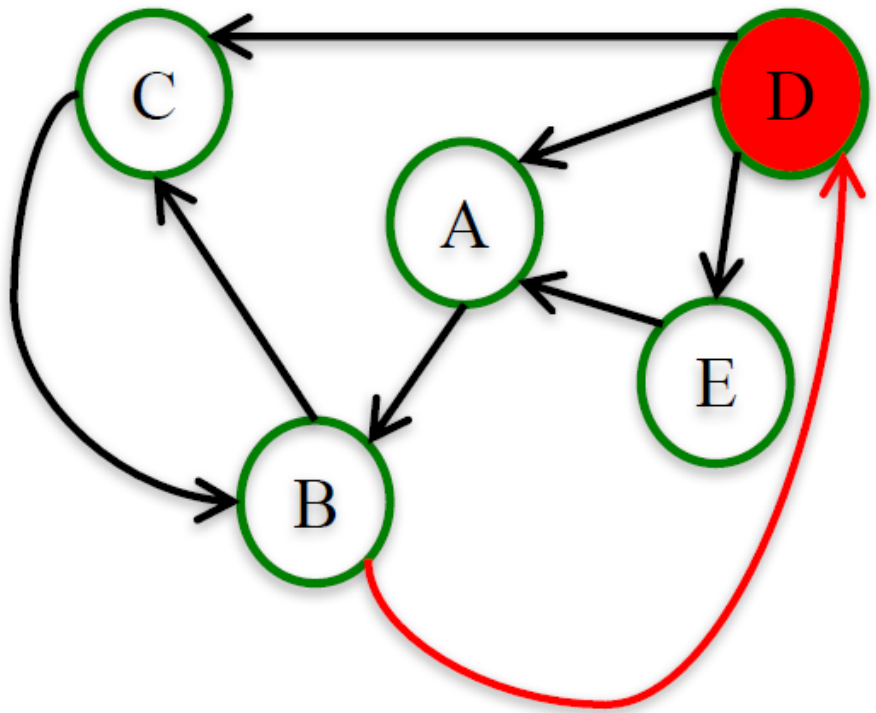


What if continue with k = 4,5,6,...? For most networks, PageRank values converge.

Interpreting PageRank

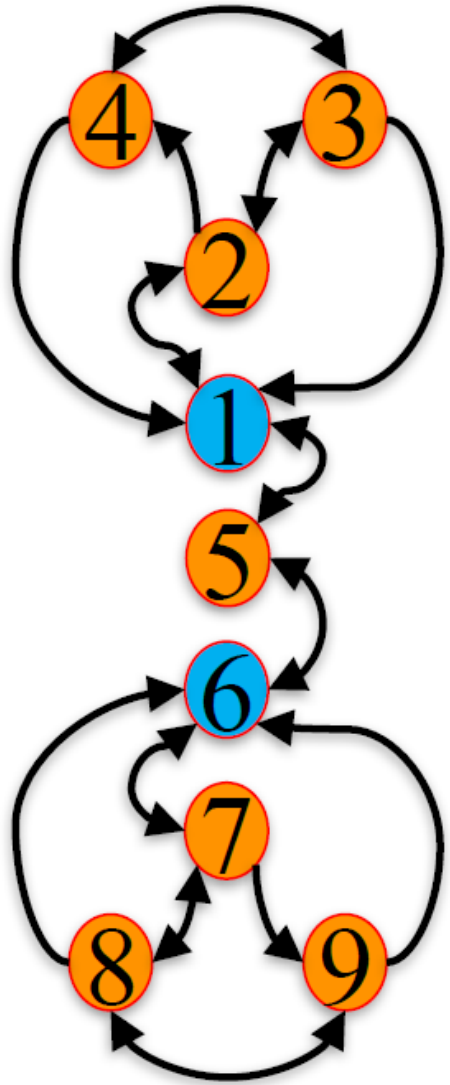
Random walk of k steps: Start on a random node. Then choose an outgoing edge at random and follow it to the next node. Repeat k times.

	Page Rank				
	A	B	C	D	E
k=∞	.12	.38	.25	.19	.06



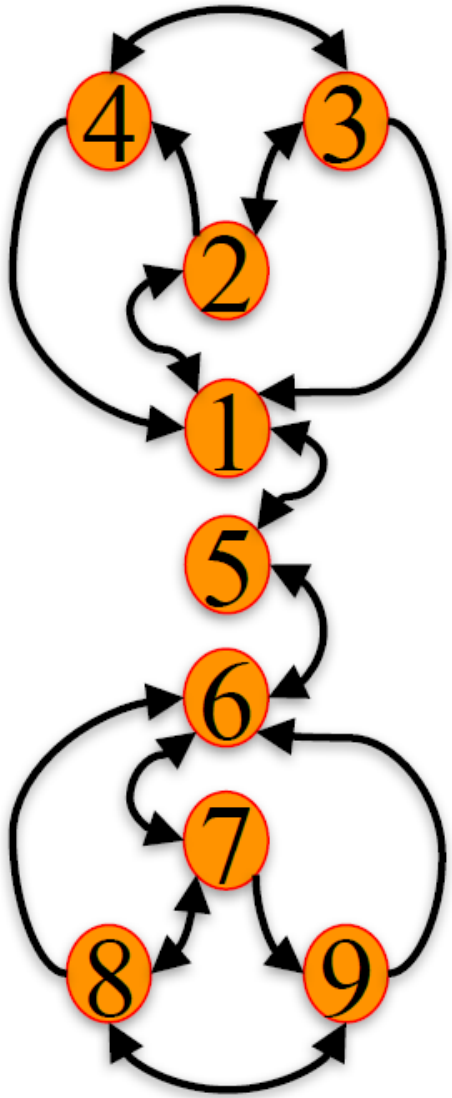
Comparing Centrality Measures

In-Deg	Closeness	Betweenness	PageRank	Auth	Hub
1	5	5	1	1	2
6	1	1	6	6	5
2	6	6	5	4	7
3	2	2	2	9	3
4	3	7	7	3	8
5	7	3	3	8	4
7	8	8	8	2	9
8	4	4	4	7	1
9	9	9	9	5	6



Summary

- In this example, no pair of centrality measures produces the exact same ranking of nodes, but they have some commonalities.
- Centrality measures make different assumptions about what it means to be a "central" node. Thus, they produce different rankings.
- The best centrality measure depends on the context of the network one is analyzing.
- When identifying central nodes, it is usually best to use multiple centrality measures instead of relying on a single one.

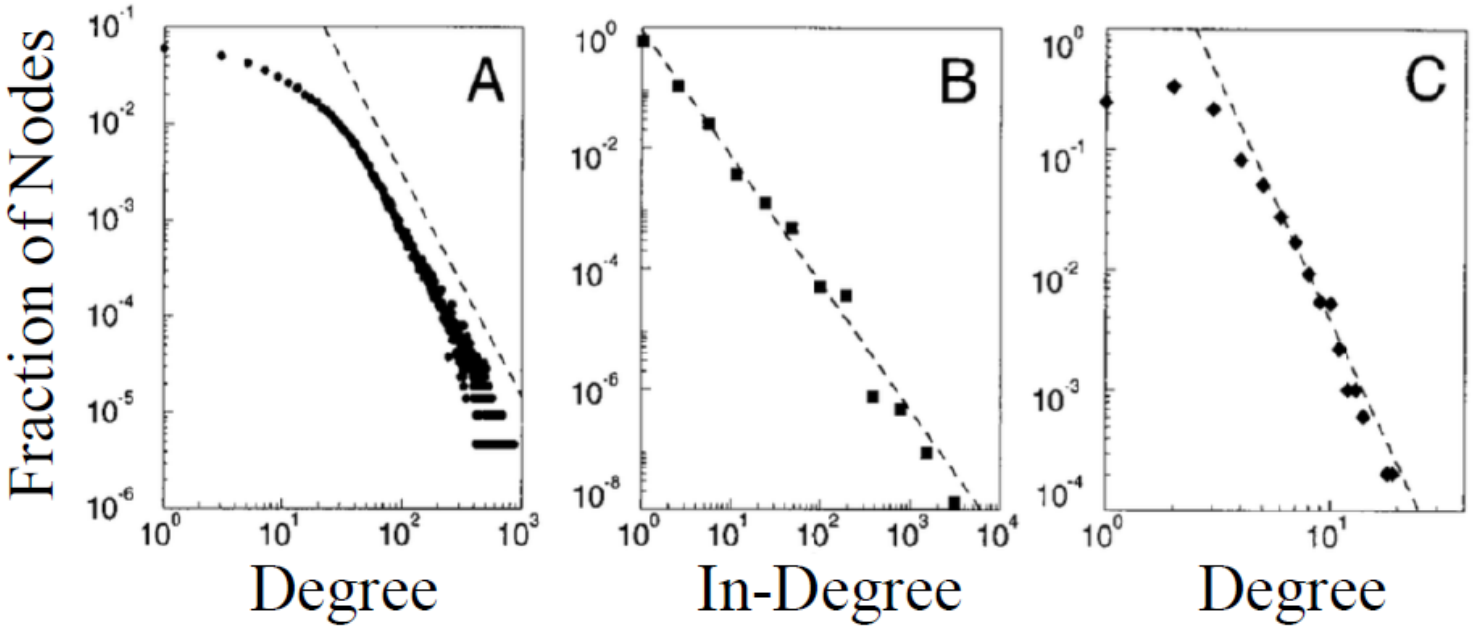


Degree Distributions in Real Networks

A – **Actors**: network of 225,000 actors connected when they appear in a movie together.

B – **The Web**: network of 325,000 documents on the WWW connected by URLs.

C – **US Power Grid**: network of 4,941 generators connected by transmission lines.



[Barabasi-Albert 99]

Degree distribution looks like a straight line when on a log-log scale. **Power law**: $P(k) = Ck^{-\alpha}$, where α and C are constants. α values: A: 2.3, B:2.1, C:4.

Networks with Power Law Degree Distributions are also called Scale-Free Networks.

Structure of a Real-World Network

	Network	Type	n	m	c	S	ℓ	α	C	C_{WS}	r	Ref(s).
Social	Film actors	Undirected	449 913	25 516 482	113.43	0.980	3.48	2.3	0.20	0.78	0.208	20,466
	Company directors	Undirected	7 673	55 392	14.44	0.876	4.60	–	0.59	0.88	0.276	131,369
	Math coauthorship	Undirected	253 339	496 489	3.92	0.822	7.57	–	0.15	0.34	0.120	133,219
	Physics coauthorship	Undirected	52 909	245 300	9.27	0.838	6.19	–	0.45	0.56	0.363	347,349
	Biology coauthorship	Undirected	1 520 251	11 803 064	15.53	0.918	4.92	–	0.088	0.60	0.127	347,349
	Telephone call graph	Undirected	47 000 000	80 000 000	3.16			2.1				10,11
	Email messages	Directed	59 812	86 300	1.44	0.952	4.95	1.5/2.0		0.16		156
	Email address books	Directed	16 881	57 029	3.38	0.590	5.22	–	0.17	0.13	0.092	364
	Student dating	Undirected	573	477	1.66	0.503	16.01	–	0.005	0.001	−0.029	52
	Sexual contacts	Undirected	2 810					3.2				304,305
Information	WWW nd.edu	Directed	269 504	1 497 135	5.55	1.000	11.27	2.1/2.4	0.11	0.29	−0.067	16,41
	WWW AltaVista	Directed	203 549 046	1 466 000 000	7.20	0.914	16.18	2.1/2.7				84
	Citation network	Directed	783 339	6 716 198	8.57			3.0/–				404
	Roget's Thesaurus	Directed	1 022	5 103	4.99	0.977	4.87	–	0.13	0.15	0.157	272
	Word co-occurrence	Undirected	460 902	16 100 000	66.96	1.000		2.7		0.44		146,175
Technological	Internet	Undirected	10 697	31 992	5.98	1.000	3.31	2.5	0.035	0.39	−0.189	102,168
	Power grid	Undirected	4 941	6 594	2.67	1.000	18.99	–	0.10	0.080	0.003	466
	Train routes	Undirected	587	19 603	66.79	1.000	2.16	–		0.69	−0.033	425
	Software packages	Directed	1 439	1 723	1.20	0.998	2.42	1.6/1.4	0.070	0.082	−0.016	352
	Software classes	Directed	1 376	2 213	1.61	1.000	5.40	–	0.033	0.012	−0.119	453
	Electronic circuits	Undirected	24 097	53 248	4.34	1.000	11.05	3.0	0.010	0.030	−0.154	174
	Peer-to-peer network	Undirected	880	1 296	1.47	0.805	4.28	2.1	0.012	0.011	−0.366	6,409
Biological	Metabolic network	Undirected	765	3 686	9.64	0.996	2.56	2.2	0.090	0.67	−0.240	252
	Protein interactions	Undirected	2 115	2 240	2.12	0.689	6.80	2.4	0.072	0.071	−0.156	250
	Marine food web	Directed	134	598	4.46	1.000	2.05	–	0.16	0.23	−0.263	245
	Freshwater food web	Directed	92	997	10.84	1.000	1.90	–	0.20	0.087	−0.326	321
	Neural network	Directed	307	2 359	7.68	0.967	3.97	–	0.18	0.28	−0.226	466,470

Table 10.1: Basic statistics for a number of networks. The properties measured are: type of network, directed or undirected; total number of nodes n; total number of edges m; mean degree c; fraction of nodes in the largest component S (or the largest weakly connected component in the case of a directed network); mean distance between connected node pairs ℓ; exponent α of the degree distribution if the distribution follows a power law (or "–" if not; in/out-degree exponents are given for directed networks); clustering coefficient C from Eq. (7.28); clustering coefficient C_{WS} from the alternative definition of Eq. (7.31); and the degree correlation coefficient r from Eq. (7.64). The last column gives the citation(s) for each network in the References. Blank entries indicate unavailable data.

Component Distribution of a Real-World Network

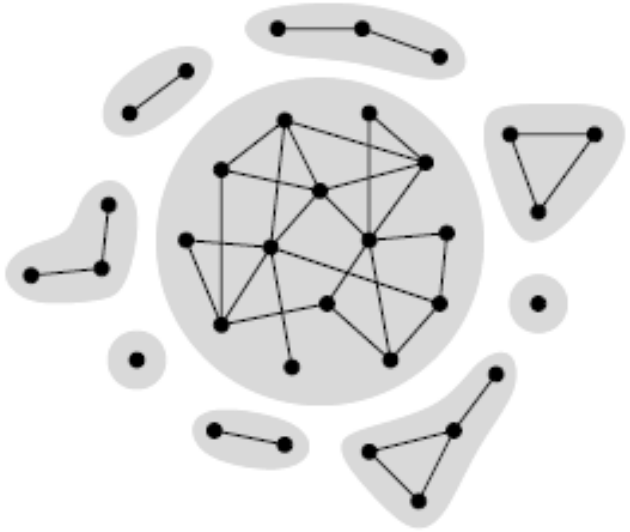


Figure 10.1: Components in an undirected network. In most undirected networks there is a single large component occupying a significant fraction of the network, along with a number of small components, typically consisting of only a handful of nodes each.

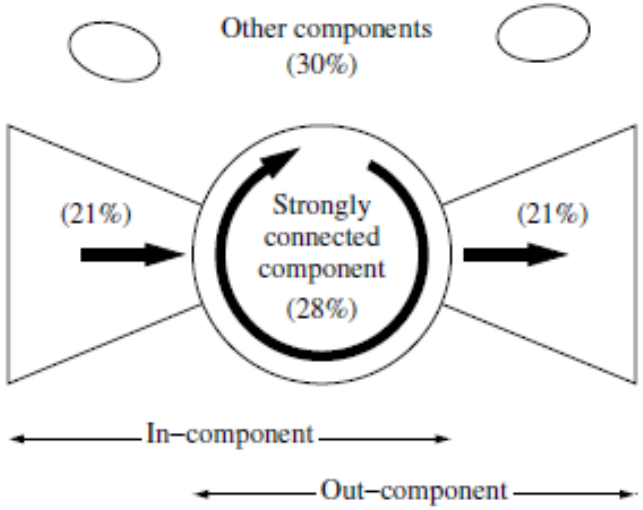


Figure 10.2: The "bow tie" diagram of components in a directed network. The typical directed network consists of one large strongly connected component and many small ones, each with an in-component and an out-component. Note that by definition each in-component includes the corresponding strongly connected component as a subset, as does each out-component. The largest strongly connected component and its in- and out-components typically occupy a significant fraction of the whole network. The percentages shown here are estimates of how much of the network is taken up by each part of the bow tie in the case of the World Wide Web. After Broder *et al.* [84].

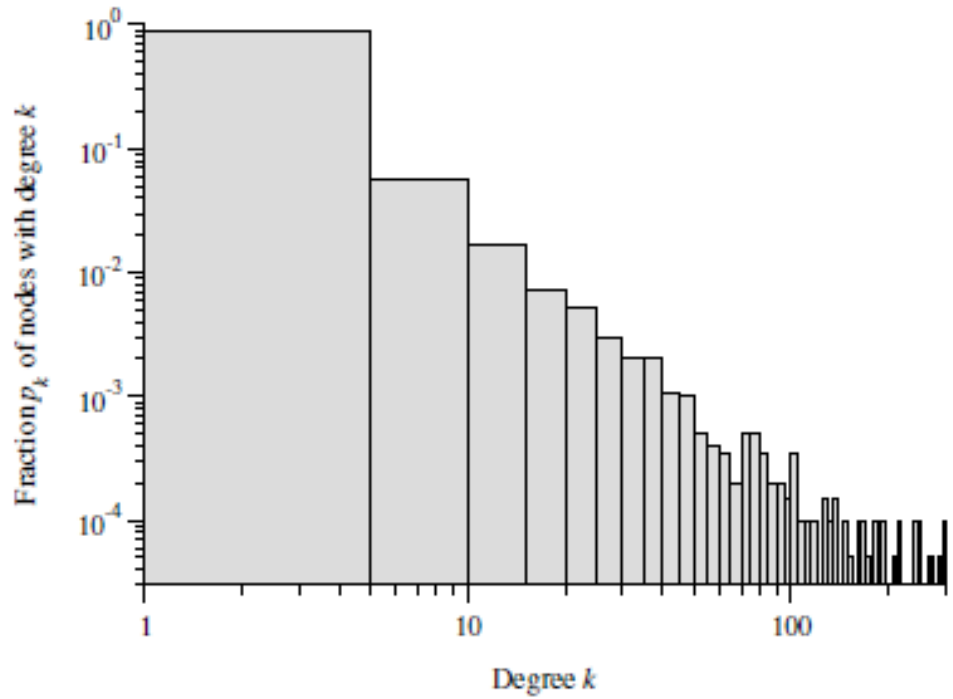


Figure 10.5: The power-law degree distribution of the Internet. Another histogram of the degree distribution of the Internet, plotted this time on logarithmic scales. The approximate straight-line form of the histogram indicates that the degree distribution roughly follows a power law of the form (10.6).

The Small-World Phenomenon

The world is small in the sense that "short" paths exists between almost any two people.

How short are these paths?

How can we measure their length?

Milgram Small World Experiment

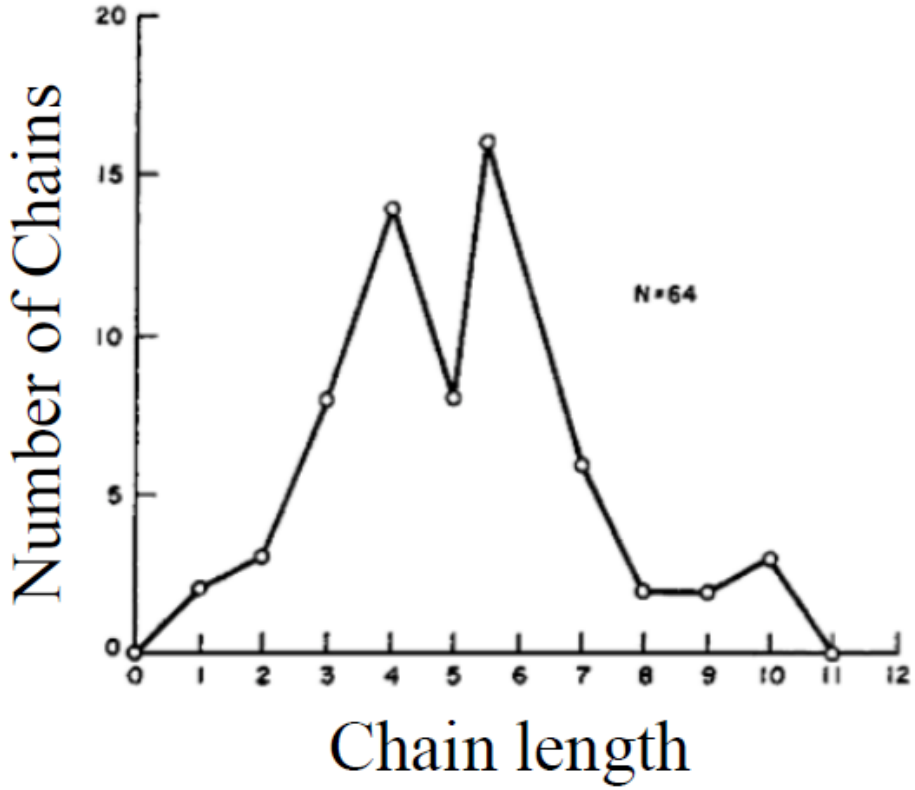
Set up (1960s):

- 296 randomly chosen "starters" asked to forward a letter to a "target" person.
- Target was a stockbroker in Boston.
- Instructions for starter:
  - Send letter to target if you know him on a first name basis.
  - If you do not know target, send letter (and instructions) to someone you know on a first name basis who is more likely to know the target.
- Some information about the target, such as city, and occupation, was provided.

Milgram Small World Experiment

Results:

- 64 out of the 296 letters reached the target.
- Median chain length was 6 (consistent with the phrase "six degrees of separation")

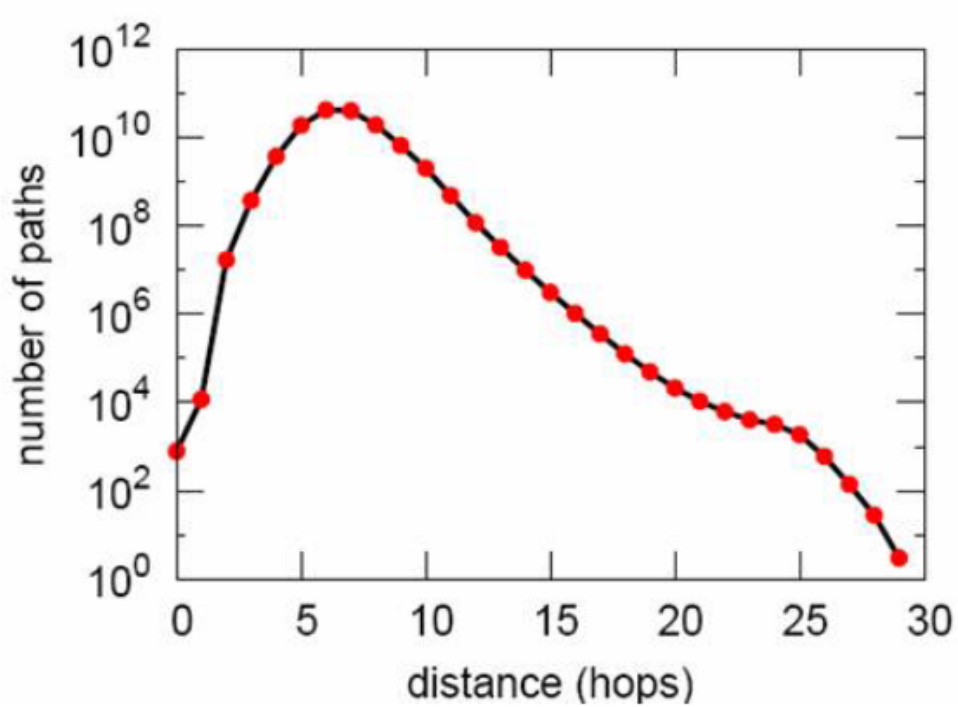


Key points:

- A relatively large percentage (>20%) of letters reached target.
- Paths were relatively short.
- People were able to find these short paths.

Small World of Instant Message

- Nodes: 240 million active users on Microsoft Instant Messenger.
- Edges: Users engaged in two-way communication over a one-month period.
- Estimated median path length of 7.



[Leskovec and Horvitz, 2008]

Network Models

Random Graph

A *random graph* is a model network in which the values of certain properties are fixed, but the network is in other respects random. One of the simplest examples of a random graph is the one where we fix only the number of nodes n and the number of edges m. That is, we take n nodes and place m edges among them at random. More precisely, we choose m distinct pairs of nodes uniformly at random from all possible pairs and connect them with an edge. This model is often referred to by its mathematical name $G(n, m)$.

Another entirely equivalent definition of the model is to say that the network is created by choosing uniformly at random among the set of all simple networks with exactly n nodes and m edges. Since there are $\binom{n}{2}$ pairs of nodes between which we could place an edge, there are $\binom{\binom{n}{2}}{m}$ ways of placing the m edges, and we simply choose any one of these with equal probability.

In $G(n, p)$ we fix not the number but the *probability* of edges between nodes. Again we have n nodes, but now we place an edge between each distinct pair with independent probability p. In this model the number of edges is not fixed. Indeed it is possible that the network could have no edges at all, or it could have edges between every distinct pair of nodes. (For most values of p these are not likely outcomes, but they could happen.)

$$c = (n-1)p. \tag{11.6}$$

In other words, the average number of edges connected to a node is equal to the expected number p between the node and any other node, multiplied by the number $n-1$ of other nodes.

$$p_k = \frac{(n-1)^k}{k!}p^k e^{-c} = \frac{(n-1)^k}{k!}\left(\frac{c}{n-1}\right)^k e^{-c} = e^{-c}\frac{c^k}{k!}, \tag{11.10}$$

in the limit of large n.

Equation (11.10) is the Poisson distribution. In the limit of large n, $G(n, p)$ has a Poisson degree distribution. This is the origin of the name *Poisson random graph* mentioned in Section 11.1, which we will use occasionally to distinguish this model from some of the other random graph models introduced in following chapters, which don't in general have Poisson degree distributions.

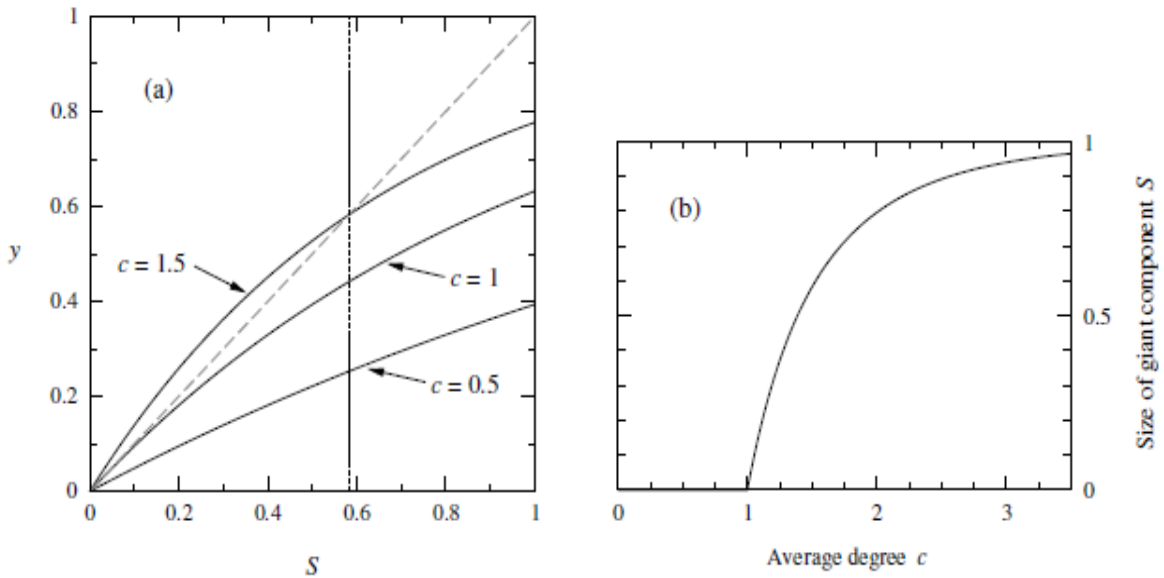


Figure 11.2: Graphical solution for the size of the giant component. (a) The three curves show $y = 1 - e^{-cS}$ for values of c as marked, the diagonal dashed line shows $y = S$, and the intersection gives the solution to Eq. (11.16), $S = 1 - e^{-cS}$. For the bottom curve there is only one intersection, at $S = 0$, so there is no giant component, while for the top curve there is a solution at $S = 0.583\ldots$ (vertical dashed line). The middle curve is precisely at the threshold between the regime where a non-trivial solution for S exists and the regime with only the trivial solution $S = 0$. (b) The resulting solution for the size of the giant component as a function of c.

The Configuration Model

The configuration model is one of the most important theoretical models in the study of networks. For many purposes it strikes an ideal balance between realism and simplicity, and is frequently the first model that network scientists turn to when studying a new question or process. If you are interested in something that happens on networks—the flow of traffic, the spread of a disease, the evolution of a dynamical system—it is often a good first step to see how it behaves on networks generated using the configuration model.

The most widely studied of the generalized random graph models is the *configuration model*.[1] The configuration model is actually a model of a random graph with a given degree *sequence*, rather than degree distribution. That is, the exact degree of each individual node in the network is specified, rather than merely the probability distribution from which those degrees are chosen.

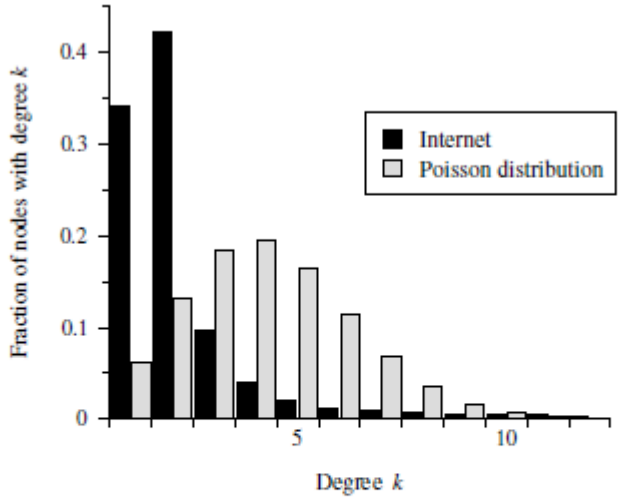


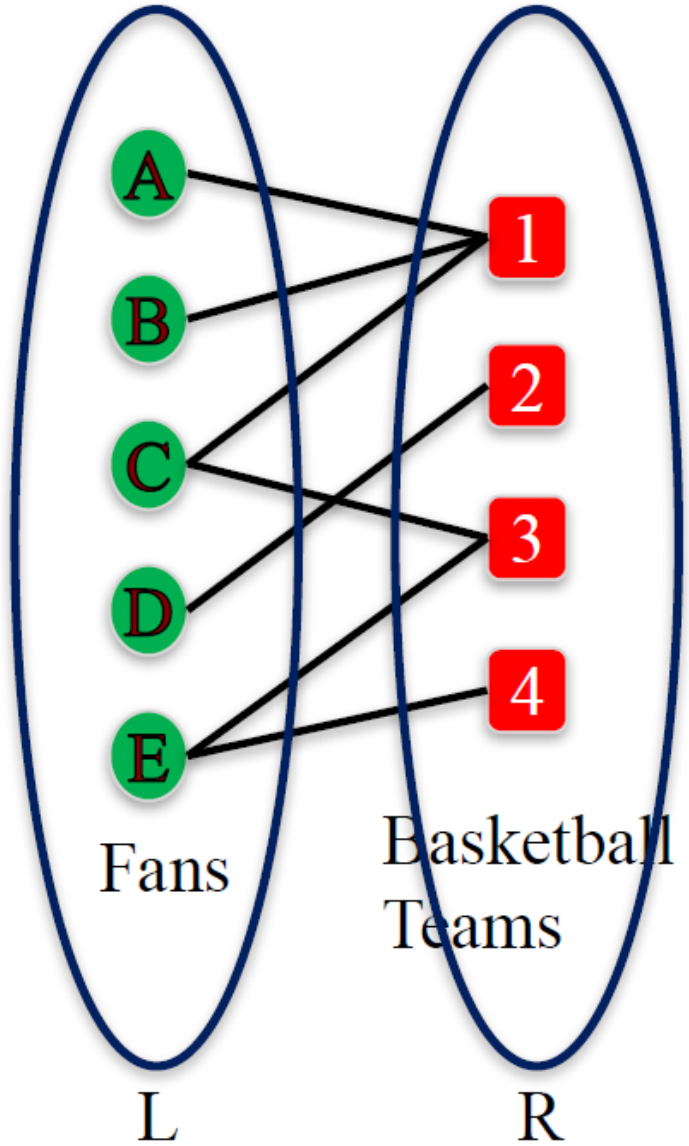
Figure 11.8: Degree distribution of the Internet and a Poisson random graph. The darker bars in this plot show the fraction of nodes with given degrees in a network representation of the Internet at the level of autonomous systems. The lighter bars show the same measure for a random graph with the same average degree as the Internet. Even though the two distributions have the same averages, it is clear that they are entirely different in shape.

Extra Slides

Bipartite Graphs

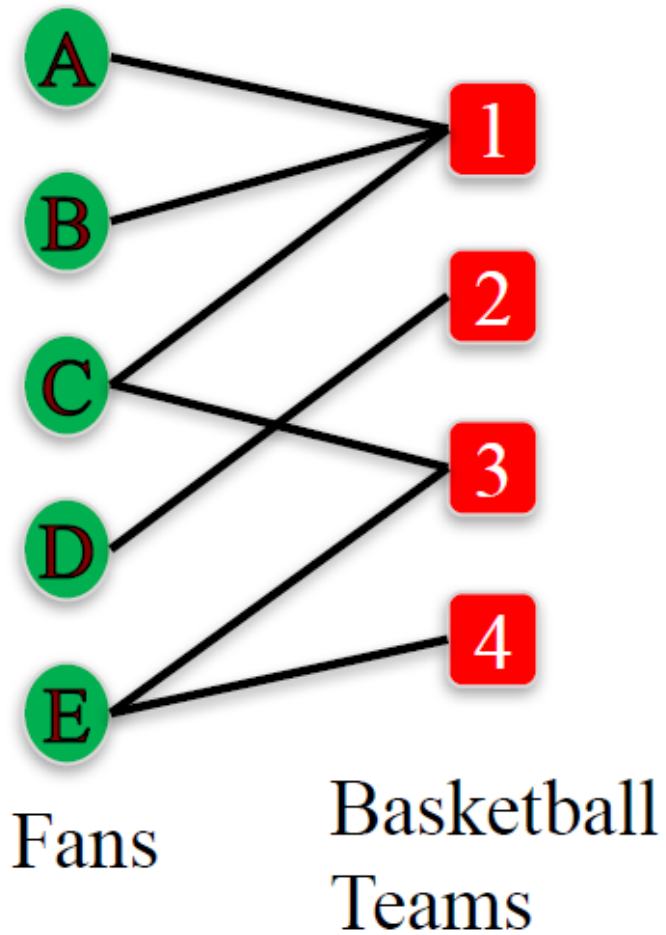
Bipartite Graph: a graph whose nodes can be split into two sets *L* and *R* and every edge connects an node in *L* with a node in *R*.

```
from networkx.algorithms import bipartite
B = nx.Graph() #No separate class for bipartite graphs
B.add_nodes_from(['A','B','C','D', 'E'], bipartite=0) #label one set of nodes 0
B.add_nodes_from([1,2,3,4], bipartite=1) #label other set of nodes 1
B.add_edges_from([('A',1), ('B',1), ('C',1), ('C',3), ('D',2), ('E',3), ('E', 4)])
```



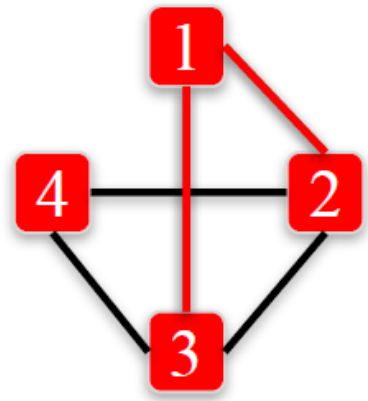
Checking if a graph is bipartite:

```
In: bipartite.is_bipartite(B) # Check if B is bipartite
Out: True

In: B.add_edge('A', 'B')
In: bipartite.is_bipartite(B)
Out: False

B.remove_edge('A', 'B')
```

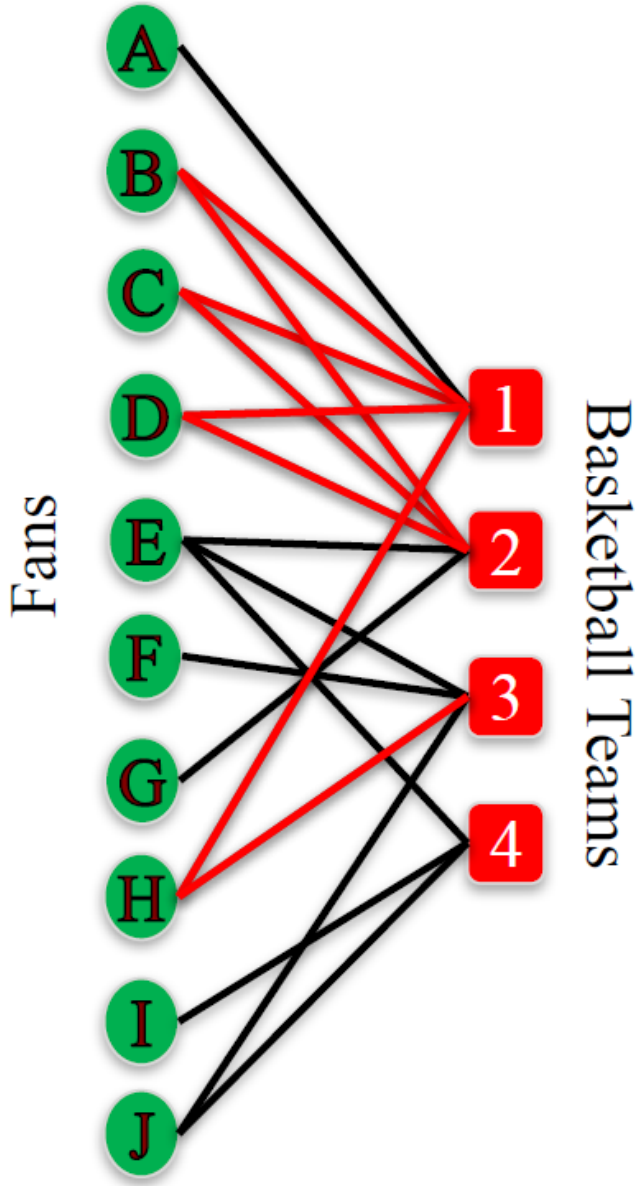


Projected Graphs

```
B = nx.Graph()
B.add_edges_from([('A',1), ('B',1),
('C',1),('D',1),('H',1), ('B', 2), ('C', 2), ('D',
2),('E', 2), ('G', 2), ('E', 3), ('F', 3), ('H', 3),
('J', 3), ('E', 4), ('I', 4), ('J', 4) ])

X = set([1,2,3,4])
P = bipartite.projected_graph(B, X)
```



Network of teams who have a fan common

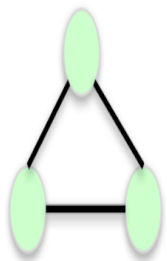


We need weights on the edges!

Measuring clustering on the whole network (Approach 2):

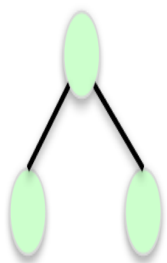
Percentage of "open triads" that are triangles in a network.

Triangles:



Transitivity = $\frac{3*\text{Number of closed triads}}{\text{Number of open triads}}$

Open triads:



Measuring clustering on the whole network:

Transitivity: Ratio of number of triangles and number of "open triads" in a network.

In: nx.transitivity(G)
Out: 0.409090909091

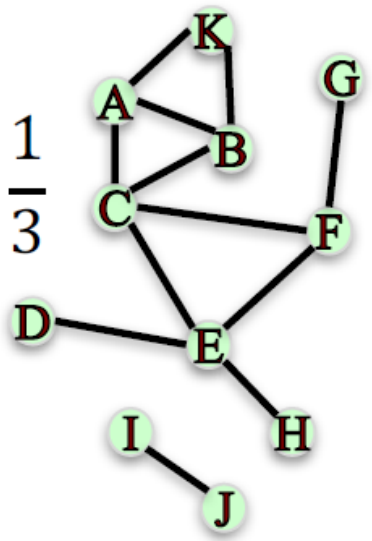
Summary

Clustering coefficient measures the degree to which nodes in a network tend to "cluster" or form triangles.

Local Clustering Coefficient

Fraction of pairs of the node's friends that are friends with each other.

LCC of C = $\frac{2}{6} = \frac{1}{3}$



Global Clustering Coefficient

Average Local Clustering Coefficient

nx.average_clustering(G)

Transitivity

Ratio of number of triangles and number of "open triads".

Puts larger weight on high degree nodes.

nx.transitivity(G)

Disconnected Nodes

How to measure the closeness centrality of a node when it cannot reach all other nodes?

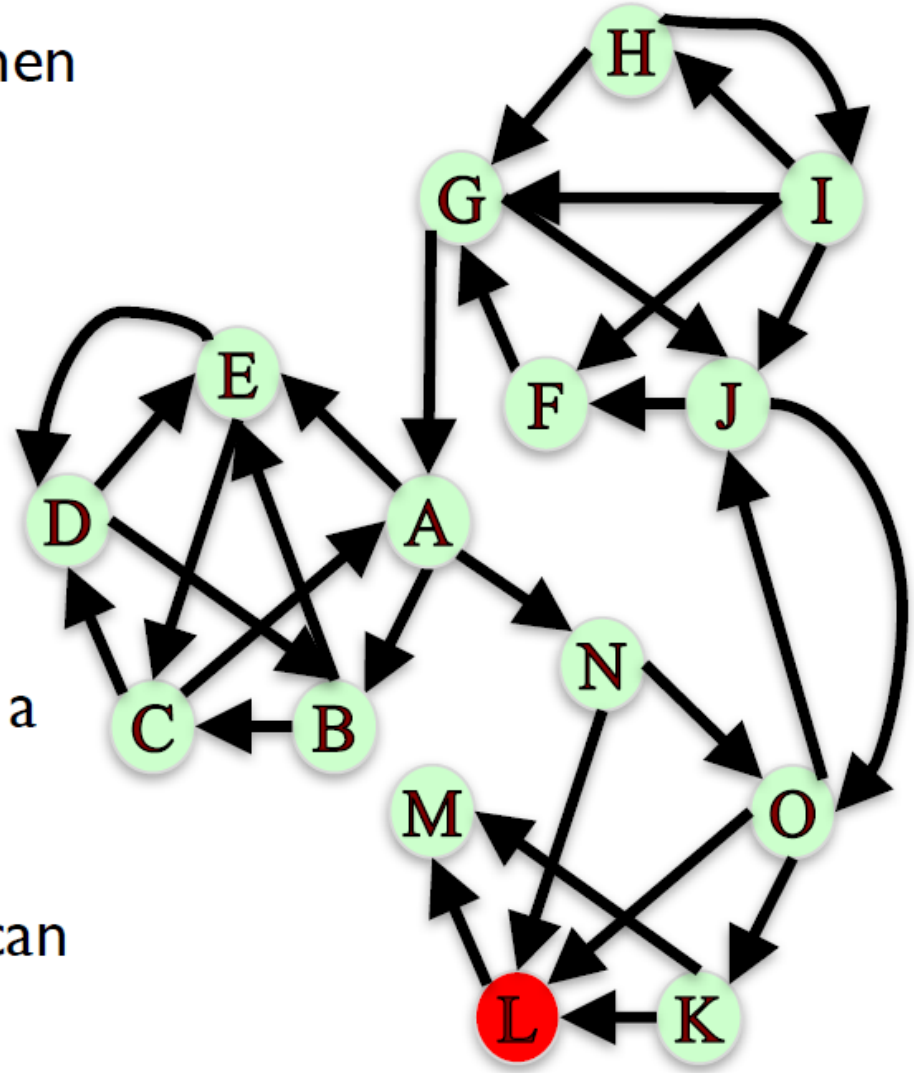
What is the closeness centrality of node L?

Option 1: Consider only nodes that L can reach:

$C_{close}(L) = \frac{|R(L)|}{\sum_{u \in R(L)} d(L,u)}$, where $R(L)$ is the set of nodes L can reach.

$C_{close}(L) = \frac{1}{1} = 1$, since L can only reach M and it has a shortest path of length 1.

Problem: centrality of 1 is too high for a node than can only reach one other node!



Disconnected Nodes

How to measure the closeness centrality of a node when it cannot reach all other nodes?

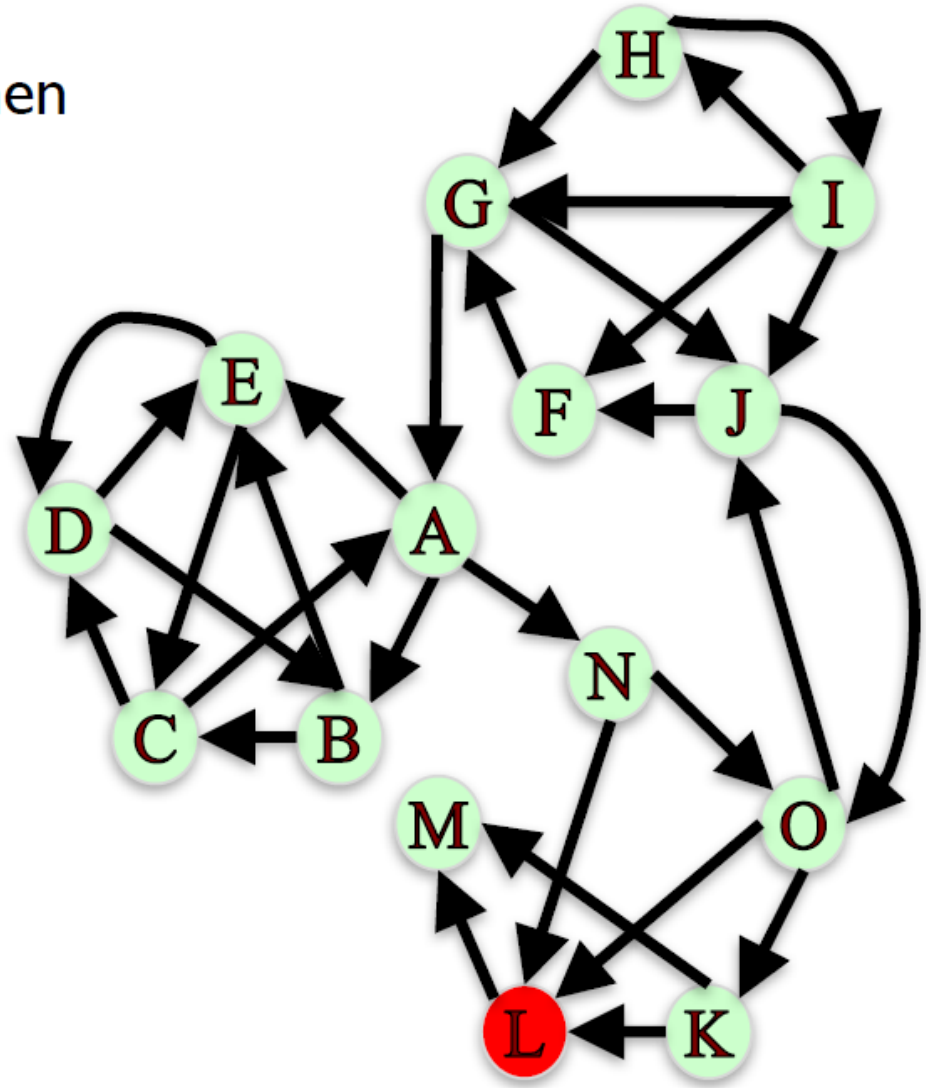
What is the closeness centrality of node L?

Option 2: Consider only nodes that L can reach and normalize by the fraction of nodes L can reach:

$$C_{close}(L) = \left[\frac{|R(L)|}{|N-1|}\right] \frac{|R(L)|}{\sum_{u \in R(L)} d(L,u)}$$

$$C_{close}(L) = \left[\frac{1}{14}\right] \frac{1}{1} = 0.071$$

Note that this definition matches our definition of closeness centrality when a graph is connected since $R(L) = N - 1$



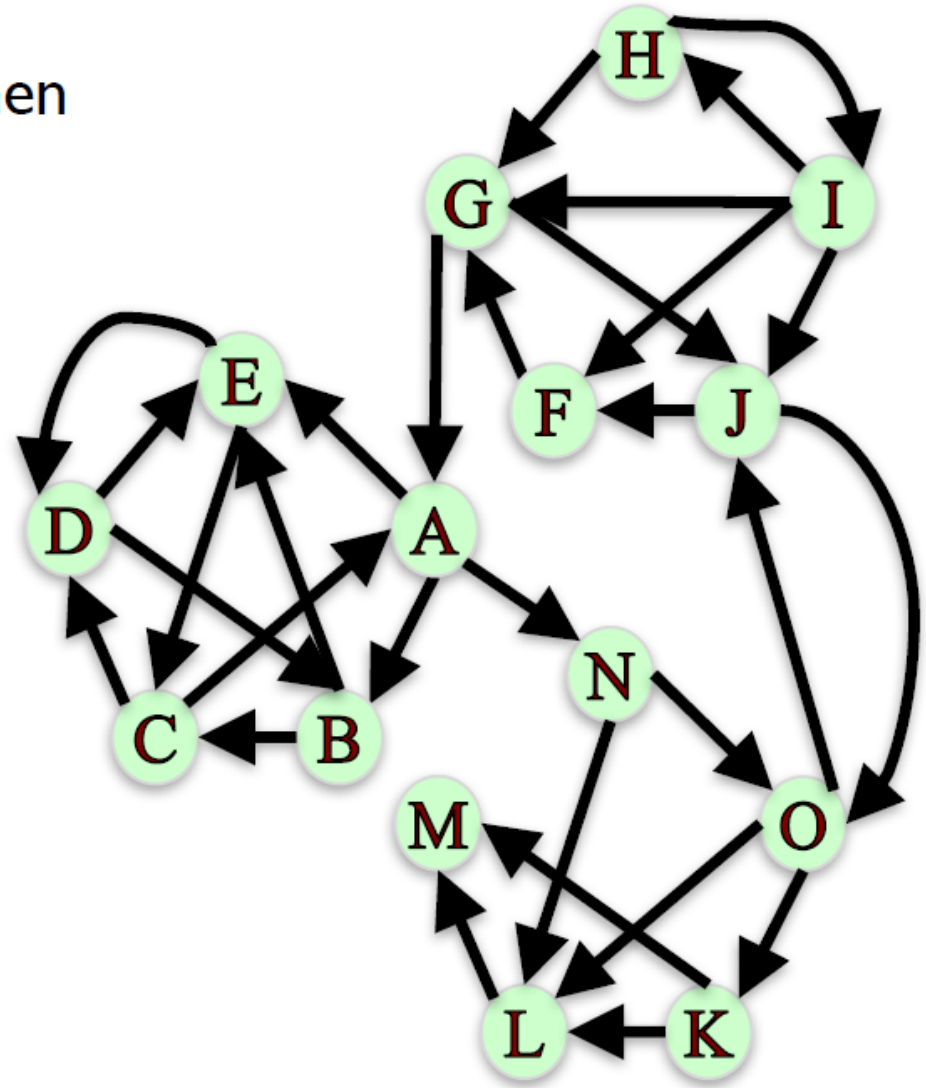
Disconnected Nodes

How to measure the closeness centrality of a node when it cannot reach all other nodes?

What is the closeness centrality of node L?

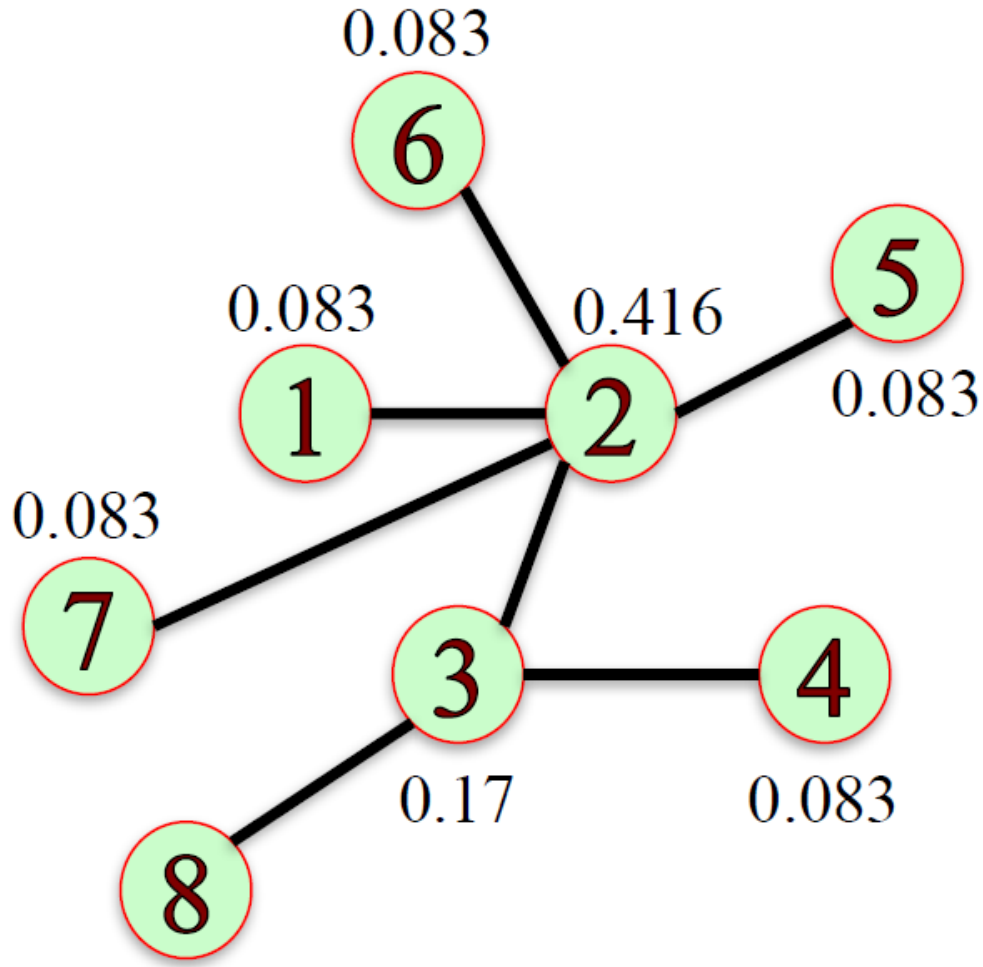
In: closeCent = nx.closeness_centrality(G, normalized = False)
In: closeCent['L']
Out: 1

In: closeCent = nx.closeness_centrality(G, normalized = True)
In: closeCent['L']
Out: 0.071



Preferential Attachment Model

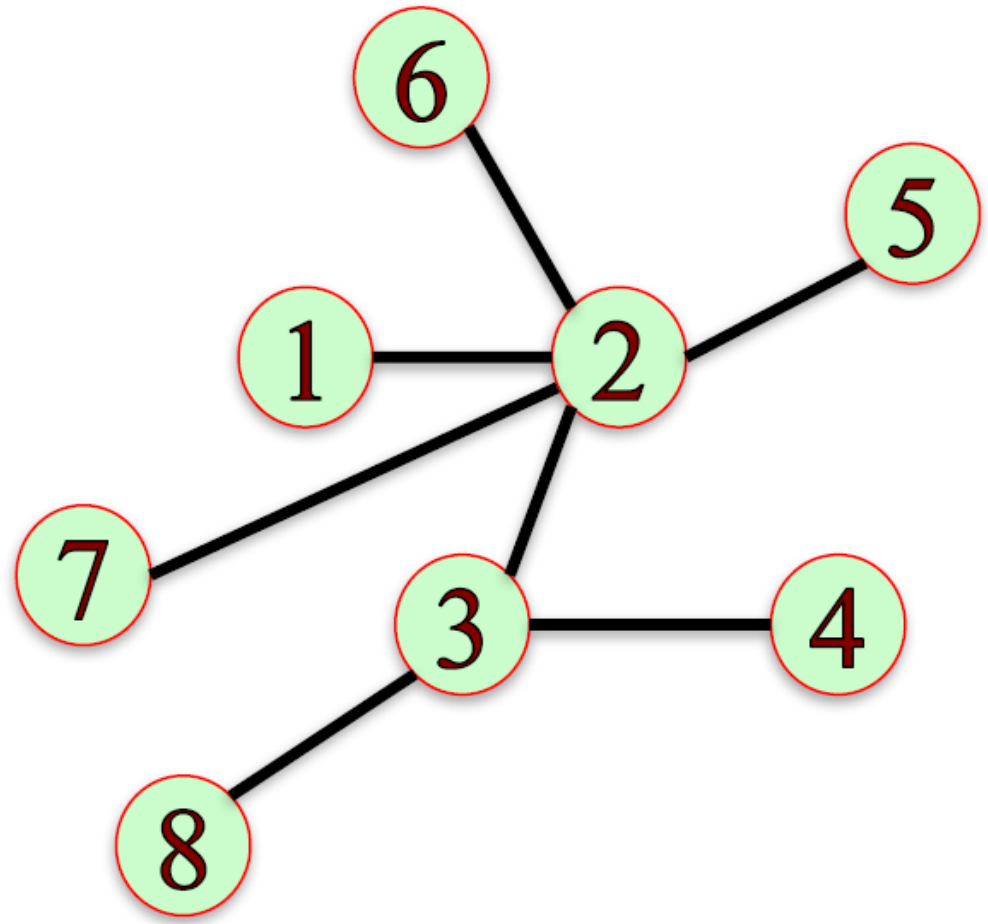
- Start with two nodes connected by an edge.
- At each time step, add a new node with an edge connecting it to an existing node.
- Choose the node to connect to at random with probability proportional to each node's degree.
- The probability of connecting to a node u of degree k_u is $k_u / \sum_j k_j$.



Preferential Attachment Model

As the number of nodes increases, the degree distribution of the network under the preferential attachment model approaches the power law $P(k) = Ck^{-3}$ with constant C.

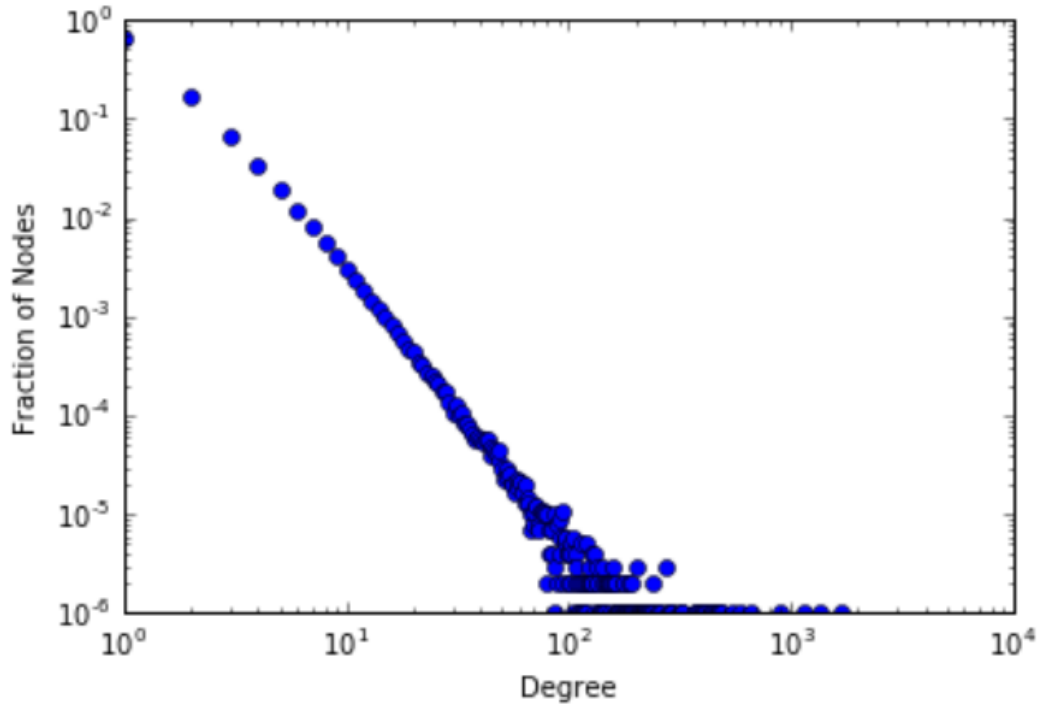
The preferential attachment model produces networks with degree distributions similar to real networks.



Preferential Attachment in NetworkX

barabasi_albert_graph(n, m) returns a network with *n* nodes. Each new node attaches to *m* existing nodes according to the Preferential Attachment model.

```
G = nx.barabasi_albert_graph(1000000,1)
degrees = G.degree()
degree_values = sorted(set(degrees.values()))
histogram =
[list(degrees.values().count(i))/float(nx.number_of_node
s(G)) for i in degree_values]

plt.plot(degree_values,histogram, 'o')
plt.xlabel('Degree')
plt.ylabel('Fraction of Nodes')
plt.xscale('log')
plt.yscale('log')
plt.show()
```



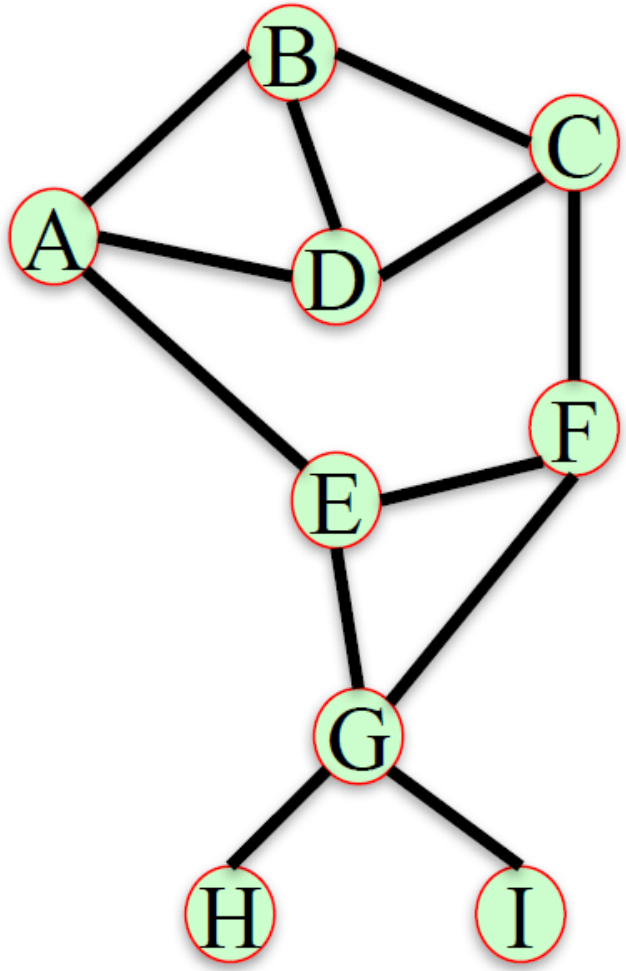
Link Prediction

What new edges are likely to form in this network?

Given a pair of nodes, how to assess whether they are likely to connect?

Triadic closure: the tendency for people who share connections in a social network to become connected.

Measure 1: number of common neighbors.



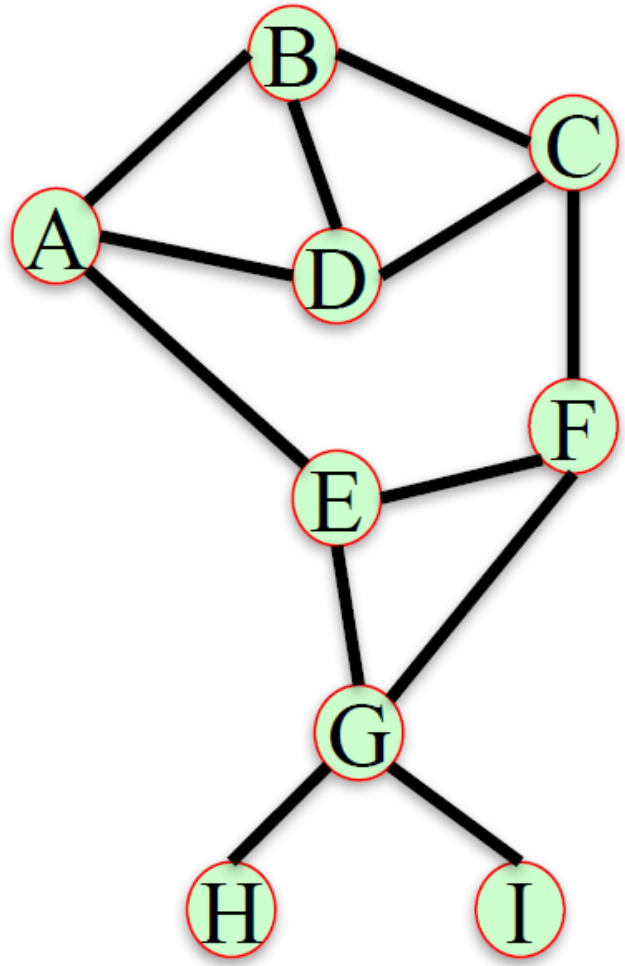
Measure 1: Common Neighbors

The number of common neighbors of nodes X and Y is

$$\text{comm\_neigh}(X, Y) = |N(X) \cap N(Y)|,$$

where $N(X)$ is the set of neighbors of node X

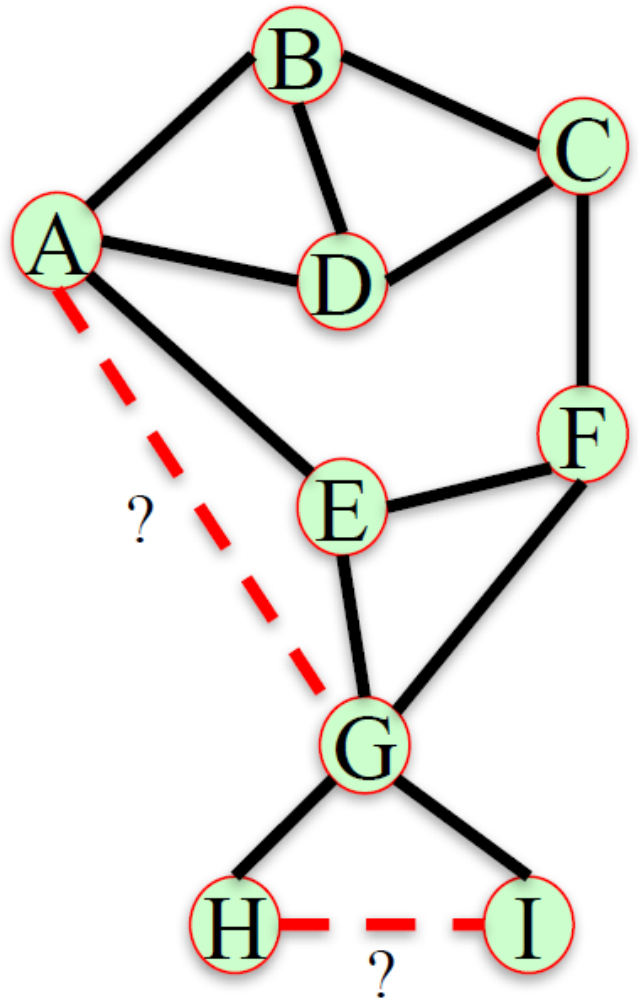
$$\text{comm\_neigh}(A, C) = |\{B, D\}| = 2$$



In: common_neigh = [(e[0], e[1], len(list(**nx.common_neighbors**(G, e[0], e[1])))) for e in nx.**non_edges**(G)]

In: sorted(common_neigh,key=operator.itemgetter(2), reverse = True
In: print (common_neigh)

Out: [('A', 'C', 2), **('A', 'G', 1)**, ('A', 'F', 1), ('C', 'E', 1), ('C', 'G', 1), ('B', 'E', 1), ('B', 'F', 1), ('E', 'I', 1), ('E', 'H', 1), ('E', 'D', 1), ('D', 'F', 1), ('F', 'I', 1), ('F', 'H', 1), **('I', 'H', 1)**, ('A', 'I', 0), ('A', 'H', 0), ('C', 'I', 0), ('C', 'H', 0), ('B', 'I', 0), ('B', 'H', 0), ('B', 'G', 0), ('D', 'I', 0), ('D', 'H', 0), ('D', 'G', 0)]



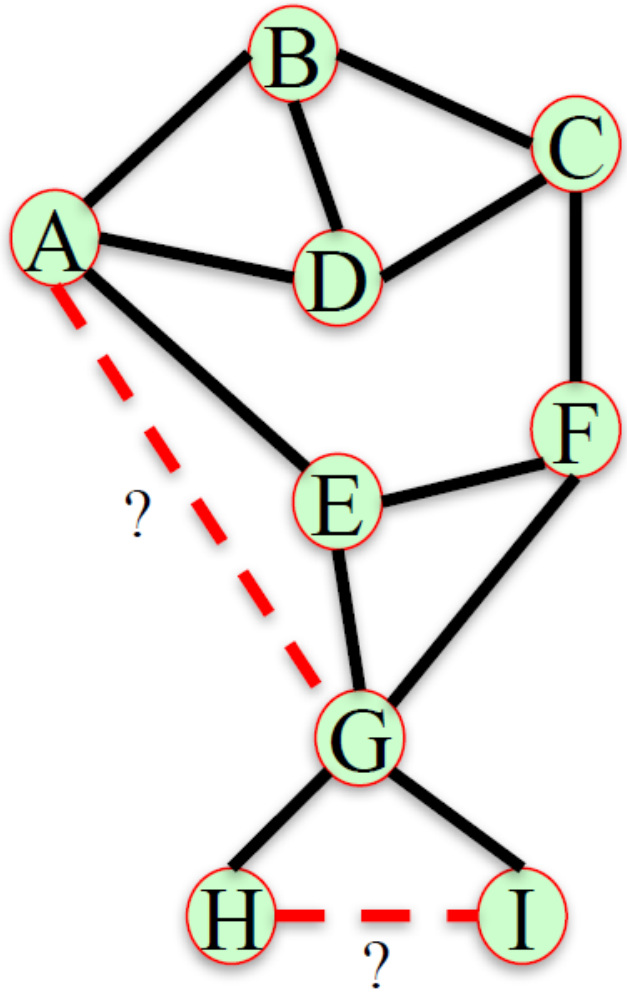
Measure 2: Jaccard Coefficient

Number of common neighbors normalized by the total number of neighbors.

The Jaccard coefficient of nodes X and Y is

$$\text{jacc\_coeff}(X, Y) = \frac{|N(X) \cap N(Y)|}{|N(X) \cup N(Y)|}$$

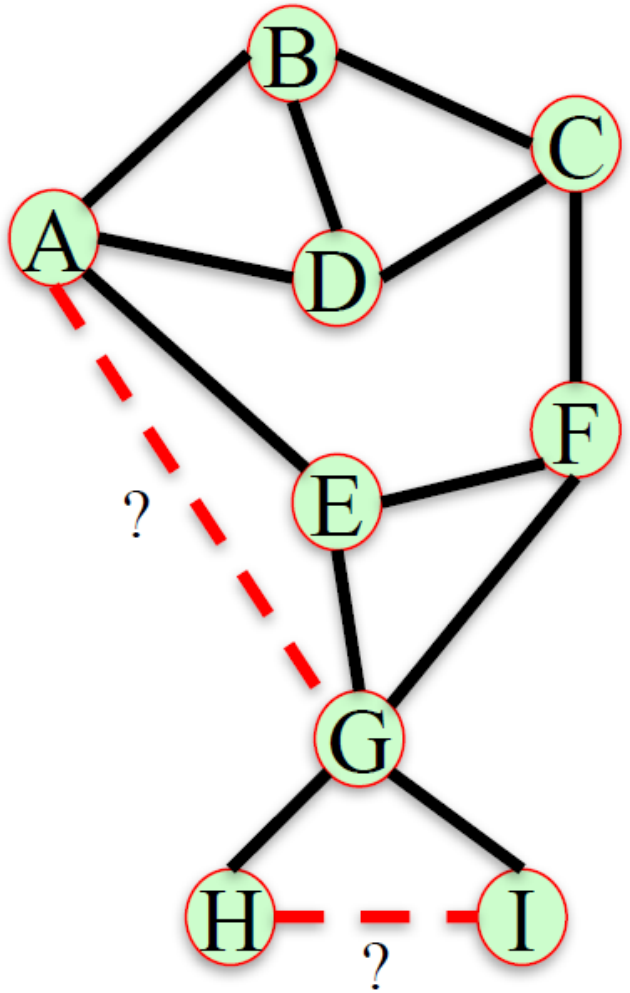
$$\text{jacc\_coeff}(A, C) = \frac{|\{B, D\}|}{|\{B, D, E, F\}|} = \frac{2}{4} = \frac{1}{2}$$



Measure 2: Jaccard Coefficient

Number of common neighbors normalized by the total number of neighbors.

In: L = list(nx.jaccard_coefficient(G))
In: L.sort(key=operator.itemgetter(2), reverse = True)
In: print(L)
Out: [**('I', 'H', 1.0)**, ('A', 'C', 0.5), ('E', 'I', 0.333333333333333), ('E', 'H', 0.333333333333333), ('F', 'I', 0.333333333333333), ('F', 'H', 0.333333333333333), ('A', 'F', 0.2), ('C', 'E', 0.2), ('B', 'E', 0.2), ('B', 'F', 0.2), ('E', 'D', 0.2), ('D', 'F', 0.2), **('A', 'G', 0.1666666666666666)**, ('C', 'G', 0.1666666666666666), ('A', 'I', 0.0), ('A', 'H', 0.0), ('C', 'I', 0.0), ('C', 'H', 0.0), ('B', 'I', 0.0), ('B', 'H', 0.0), ('B', 'G', 0.0), ('D', 'I', 0.0), ('D', 'H', 0.0), ('D', 'G', 0.0)]



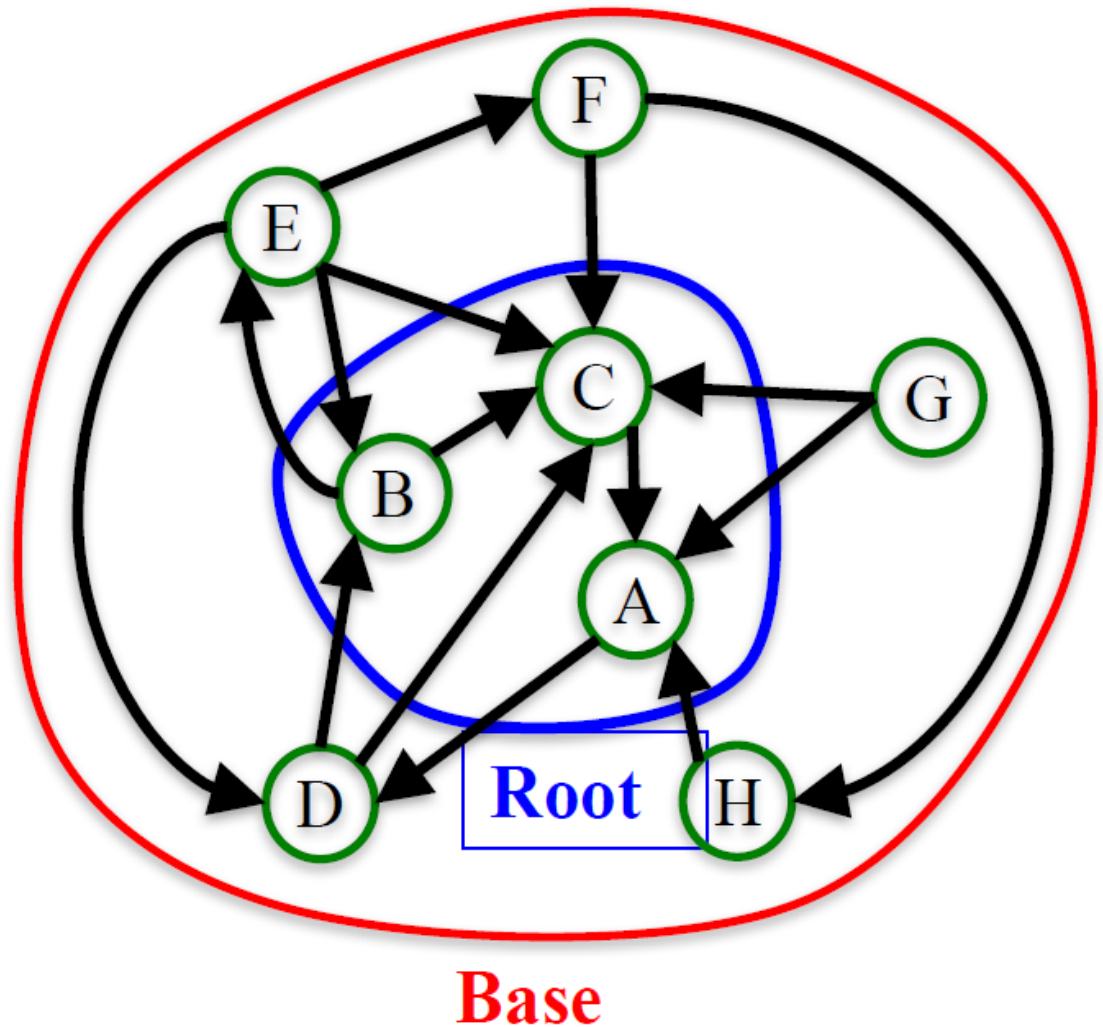
Summary

- Link prediction problem: Given a network, predict which edges will be formed in the future.
- 5 basic measures:
  - Number of Common Neighbors
  - Jaccard Coefficient
  - Resource Allocation Index
  - Adamic-Adar Index
  - Preferential Attachment Score
- 2 measures that require community information:
  - Common Neighbor Soundarajan-Hopcroft Score
  - Resource Allocation Soundarajan-Hopcroft Score

Hubs and Authorities

Given a query to a search engine:

- **Root**: set of highly relevant web pages (e.g. pages that contain the query string) – potential *authorities*.
- Find all pages that link to a page in root – potential *hubs*.
- **Base**: root nodes and any node that links to a node in root.
- Consider all edges connecting nodes in the base set.



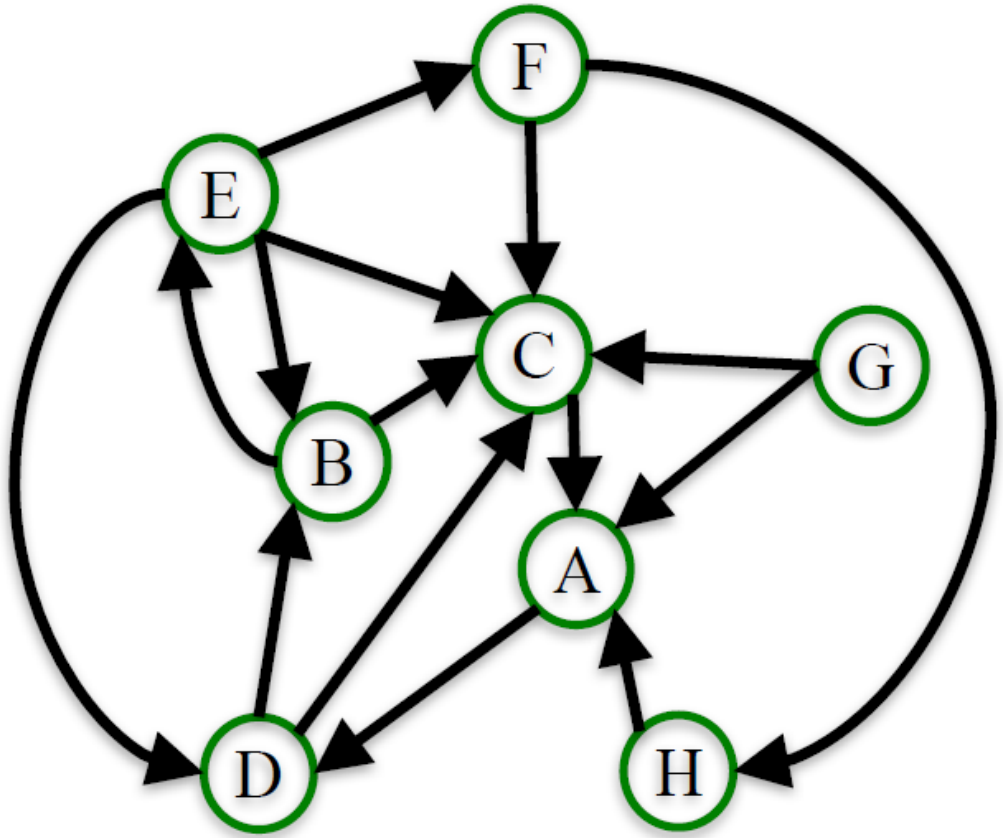
HITS Algorithm

Computing k iterations of the HITS algorithm to assign an *authority score* and *hub score* to each node.

1. Assign each node an authority and hub score of 1.
2. Apply the ***Authority Update Rule***: each node's *authority* score is the sum of *hub* scores of each node that *points to it*.
3. Apply the ***Hub Update Rule***: each node's hub score is the sum of authority scores of each node that *it points to*.
4. **Nomalize** Authority and Hub scores: $\text{auth}(j) = \frac{\text{auth}(j)}{\sum_{i \in N} \text{auth}(i)}$
5. Repeat k times.

HITS Algorithm Example

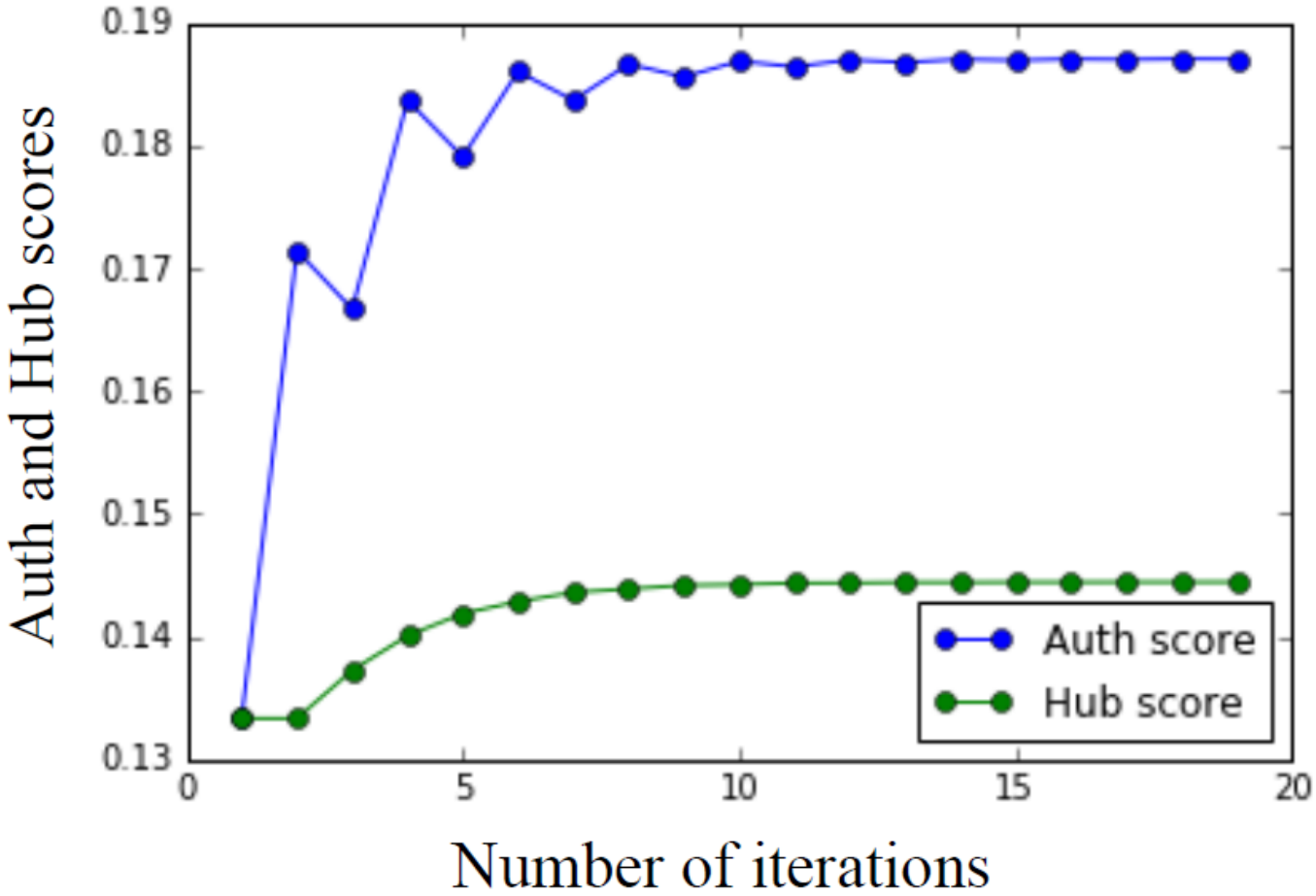
	Old Auth	Old Hub	New Auth	New Hub
A	1	1	3	1
B	1	1	2	2
C	1	1	5	1
D	1	1	2	2
E	1	1	1	4
F	1	1	1	2
G	1	1	0	2
H	1	1	1	1

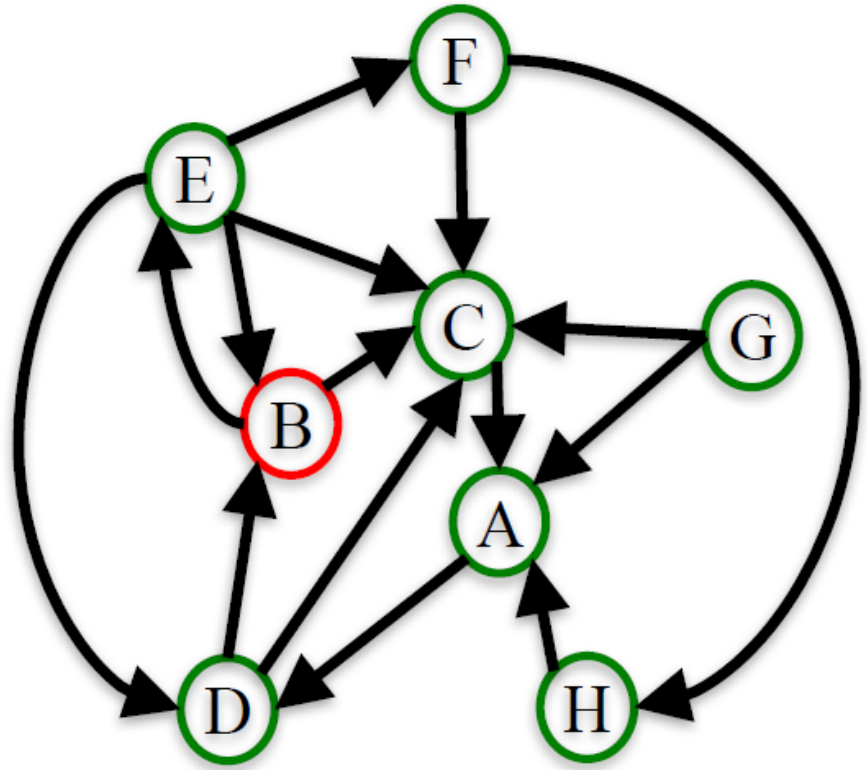


Normalize:

$\sum_{i \in N} \text{auth}(i) = 15 \quad \sum_{i \in N} \text{hub}(i) = 15$

HITS Algorithm Convergence

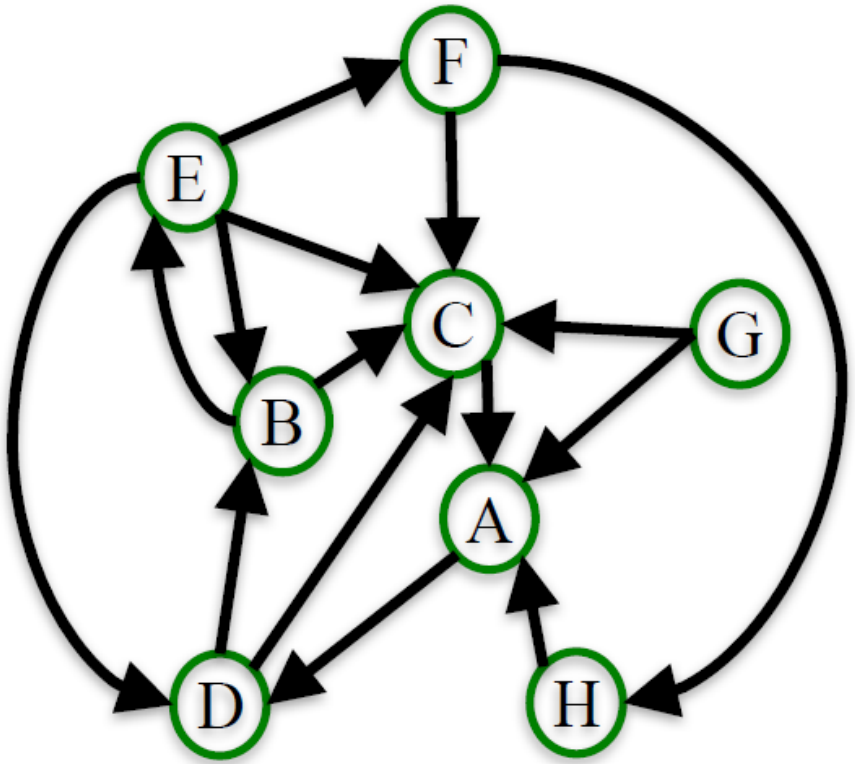




HITS Algorithm NetworkX

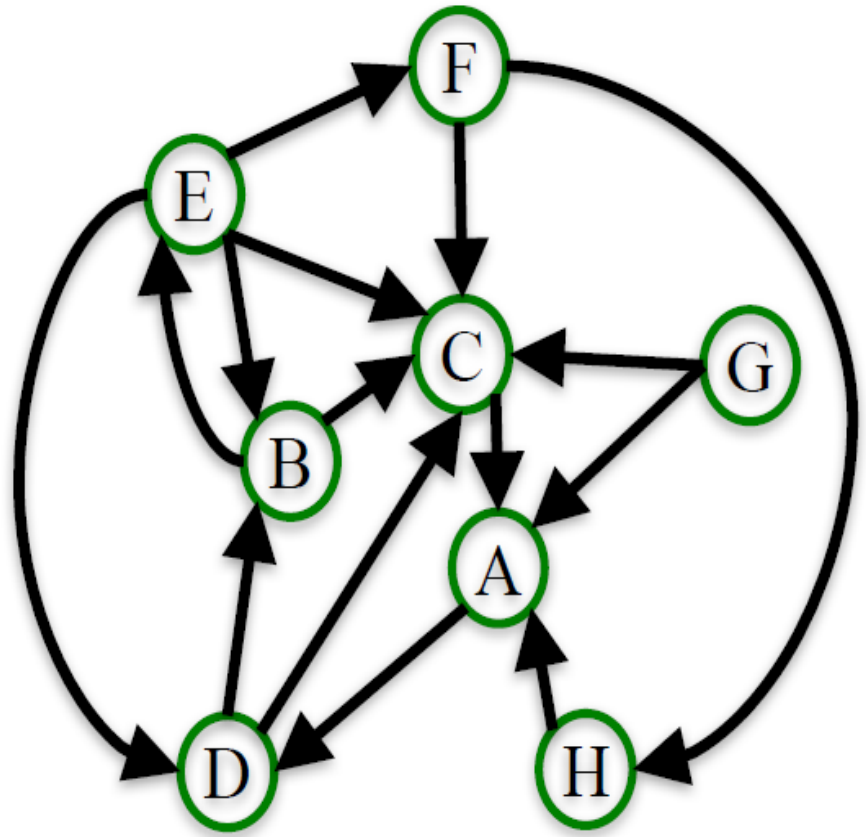
You can use NetworkX funtion hits(G) to compute the hub and authority scores of network G.

hits(G) outputs two dictionaries, keyed by node, with the hub and authority scores of the nodes.



Summary

- The HITS algorithm starts by constructing a *root set* of relevant web pages and expanding it to a *base set.*
- HITS then assigns an authority and hub score to each node in the network.
- Nodes that have incoming edges from *good hubs* are *good authorities*, and nodes that have outgoing edges to *good authorities* are *good hubs*.



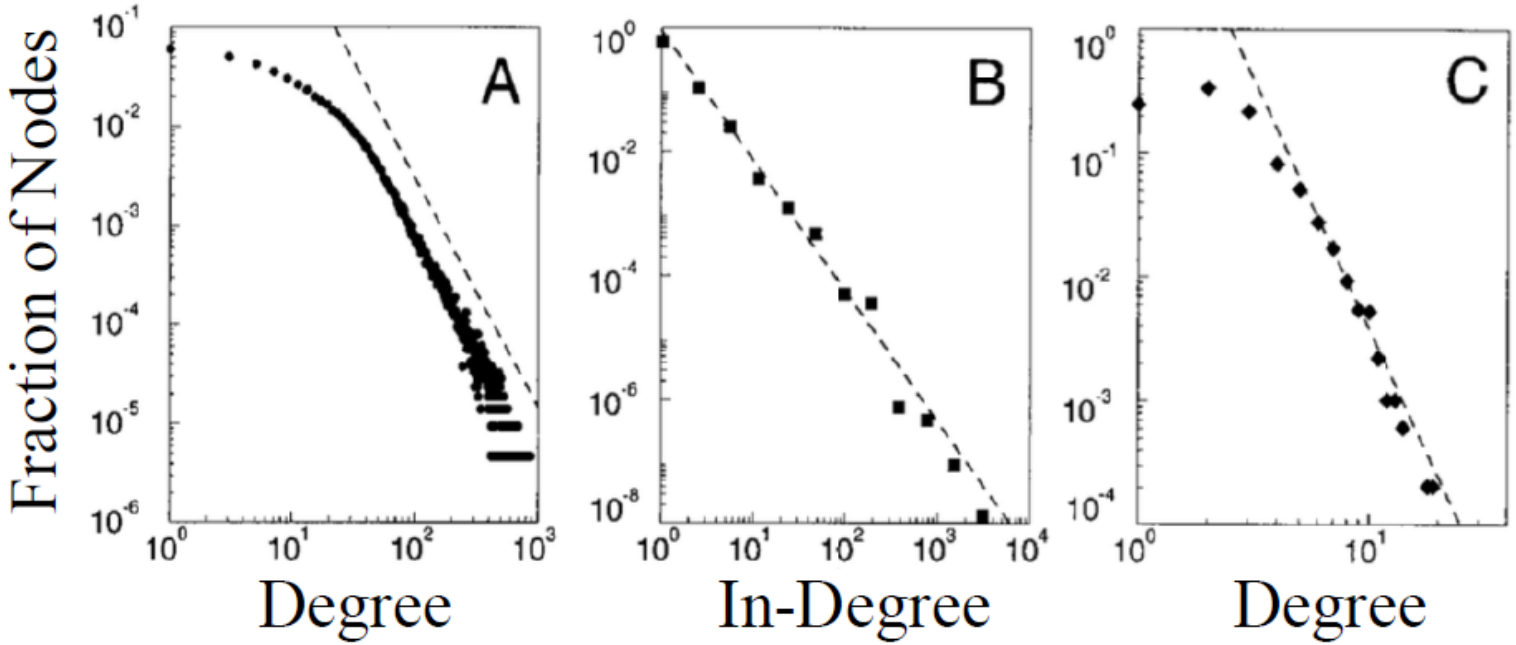
- Authority and hub scores converge for most networks.
- You can use NetworkX function hits(G) to compute the hub and authority scores of network G

Modeling Networks

Networks with power law distribution have many nodes with small degree and a few nodes with very large degree.

What could explain power law degree distribution we observe in many networks?

Can we find a set of basic assumptions that explain this phenomenon?



[Barabasi-Albert 99]

Degree distribution looks like a straight line when on a log-log scale. **Power law**: $P(k) = Ck^{-\alpha}$, where α and C are constants. α values: A: 2.3, B:2.1, C:4.