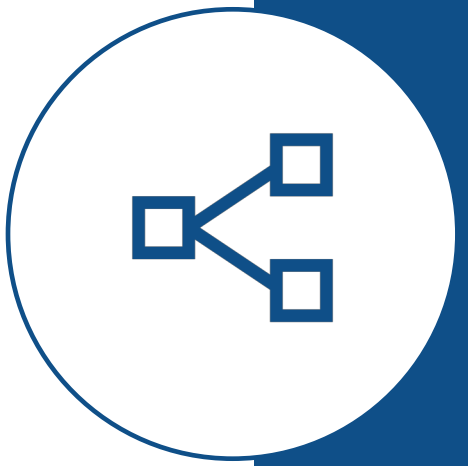
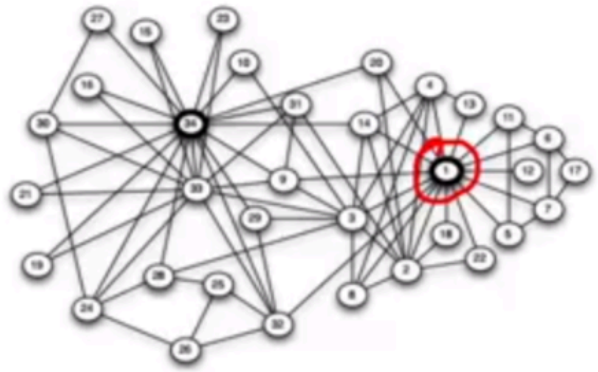
CS 193.1: Social Computing

Social Network Analysis

Social network analysis (SNA)

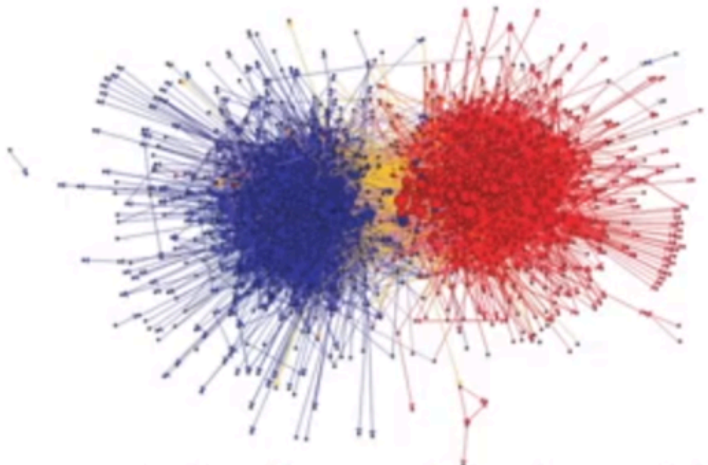
- A **social network** is a grouping of social entities with connections to one another that are significant and consequential.
- **SNA** is a systematic analysis of linkages between these entities.





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

Photos courtesy of University of Michigan.



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

What can we ask?

- Is a rumor likely to spread in a social network?
- Who are the most influential people in an organization?
- How diverse are friend groups in a university?

NetworkX

```
pip3 install networkx
```

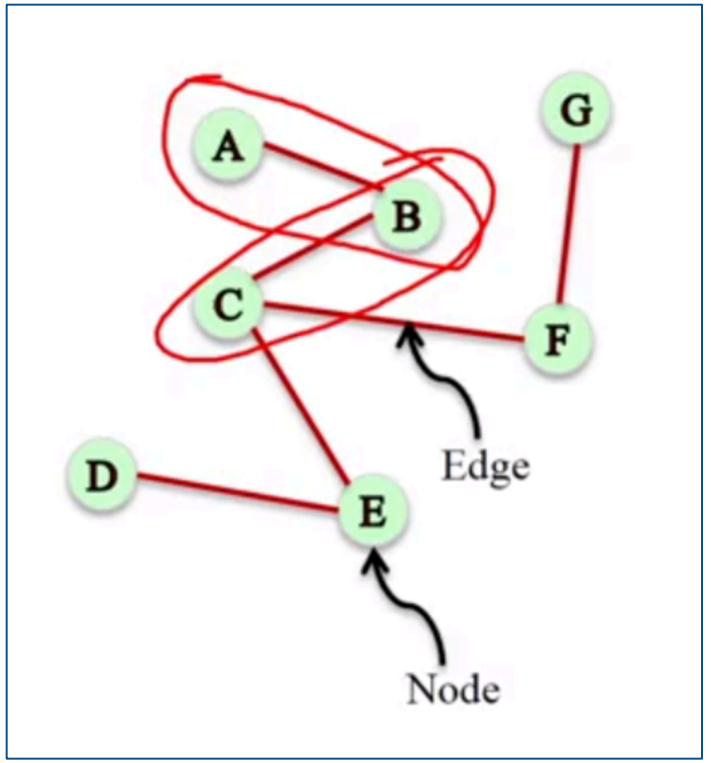
Network terminology

How do we define networks?

Undirected Graph

- **Network** (or graph): representation of nodes connected by edges.
- **Node** (or vertices): items or actors
- **Edges** (or links): connections between nodes.
- **Degree**: number of edges a node has.

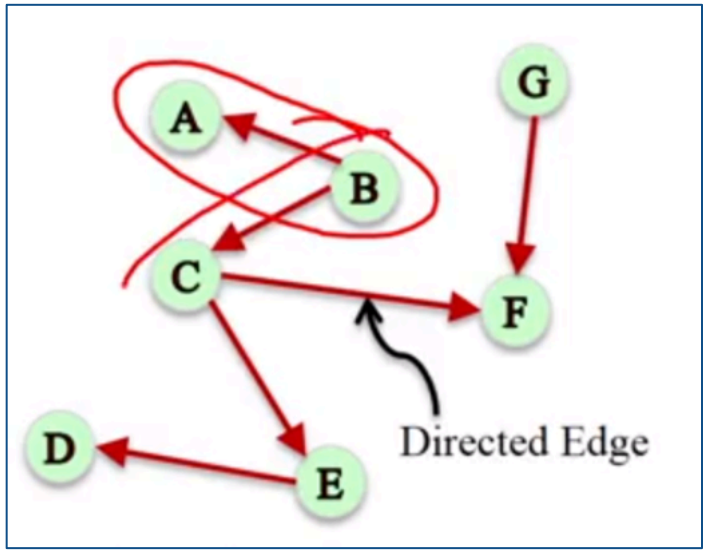
```
import networkx as nx

// Undirected graph
G = nx.Graph()
G.add_edge('A', 'B’)
G.add_edge('B', 'C')
```



Directed graph

- **Undirected graph**: edges have no direction
- **Directed graph**: edges have direction

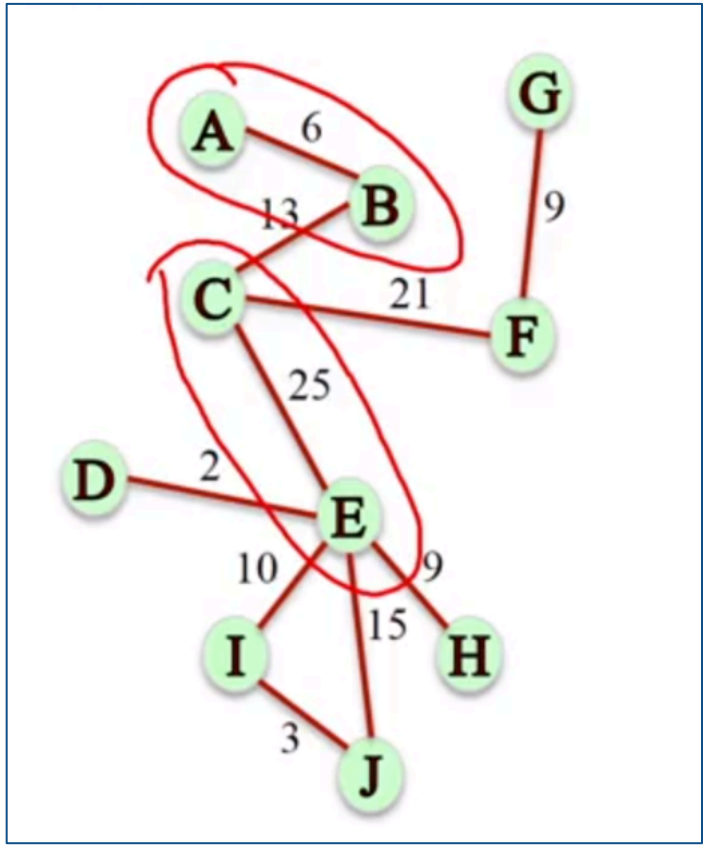
```
import networkx as nx

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



Weighted network

- **Weighted network**: edges are usually assigned a numerical weight.
  - Can be assigned with other attributes as well.

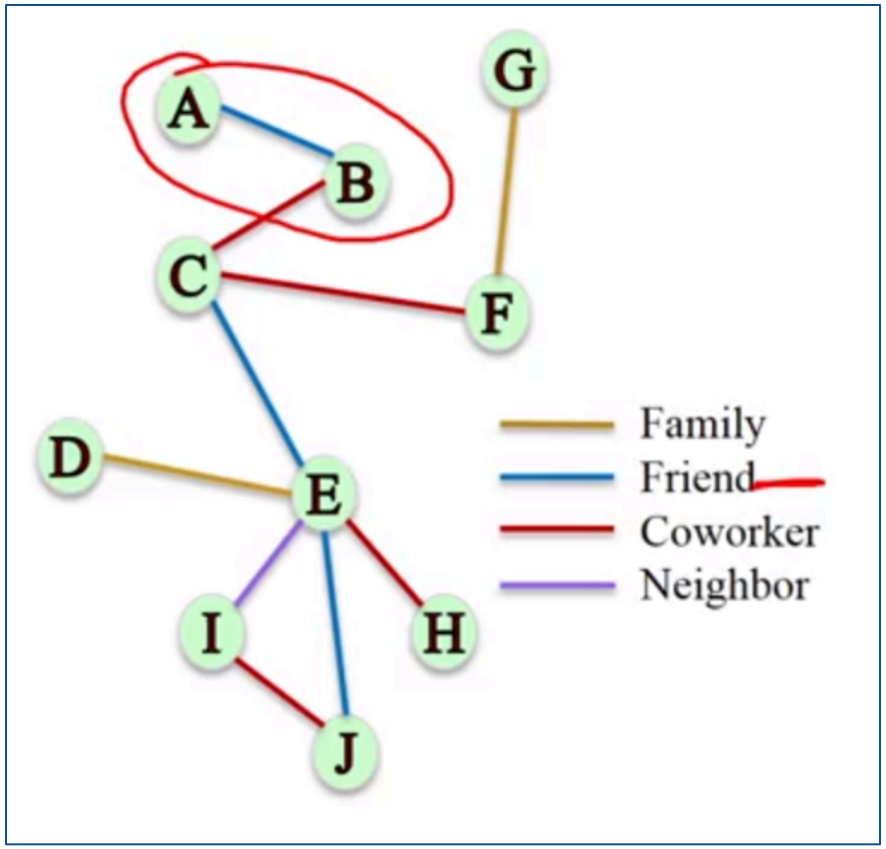
```
import networkx as nx

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



Weighted network

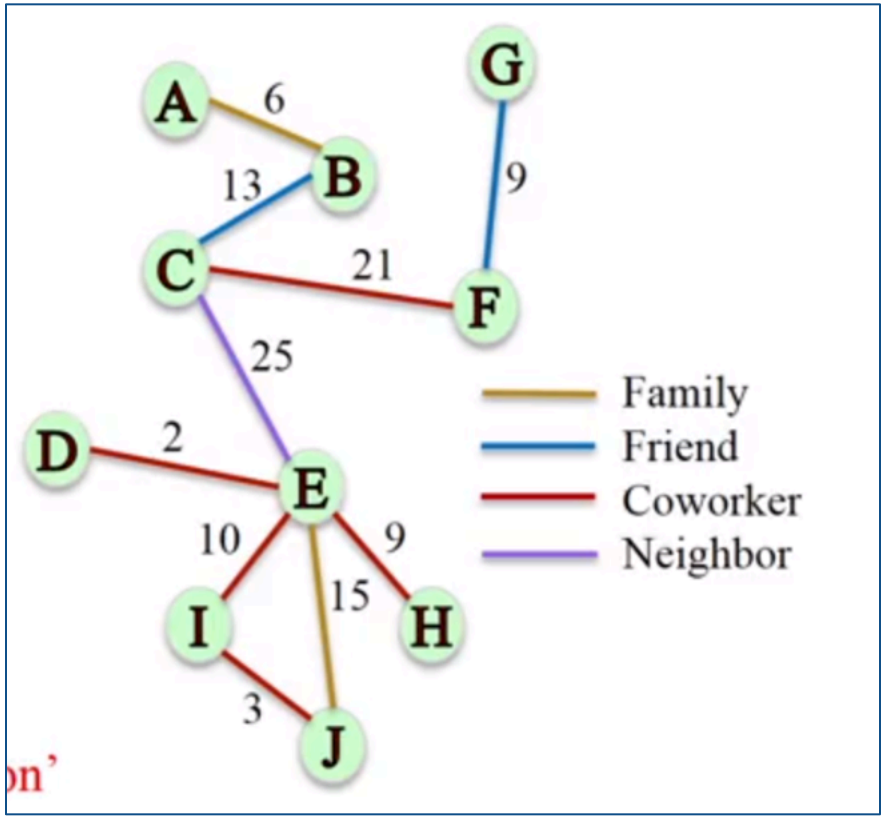
- A weighted network can be assigned with other attributes as well, like strings.

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



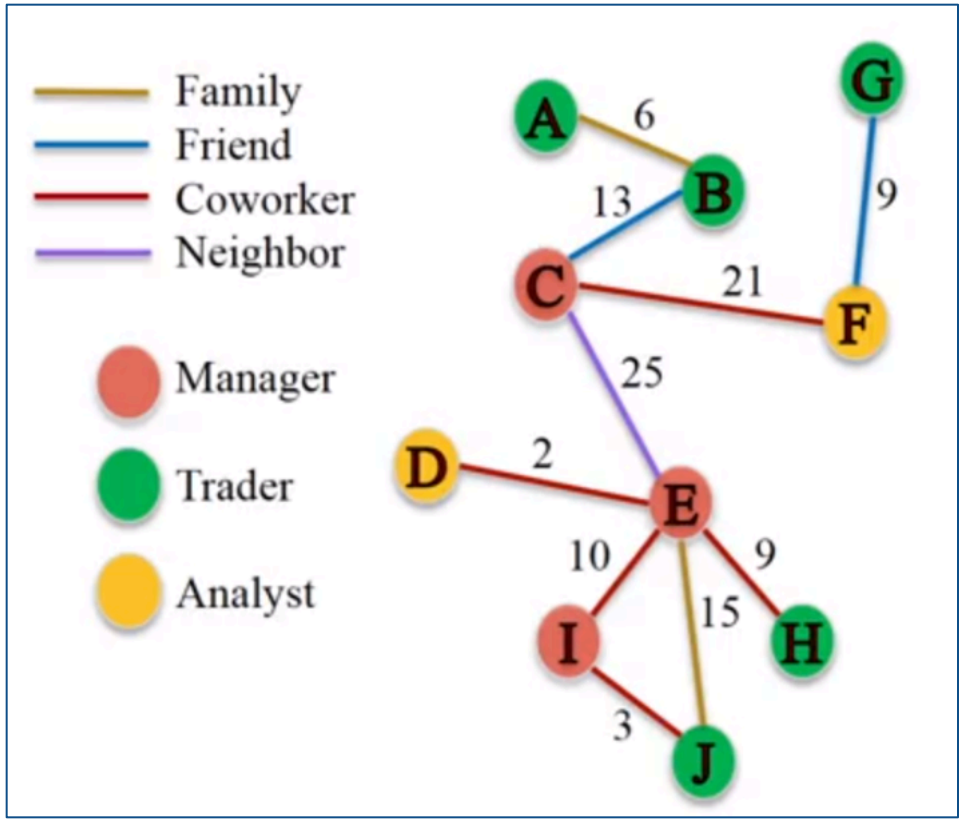
Edge attributes

```
G = nx.Graph()
G.add_edge('A', 'B', weight=6, relation='family')
G.add_edge('B', 'C', weight=3, relation='friend')

G.edges() #Get all edges
G.edges(data=True) #Get all edge attributes
#Get all edges with relation attributes
G.edges(data='relation')

#Get attributes of specific edge
G.get_edge_data('A', 'B')
#Get specific attribute value of an edge
G.get_edge_data('A', 'B')['relation']
```



Node attributes

```
G = nx.Graph()
G.add_edge('A', 'B', weight=6, relation='family')
G.add_edge('B', 'C', weight=3, relation='friend')

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

# Access node data
nodes = G.nodes()
data = G.nodes(data=True)
role = G.nodes(data='role')
```



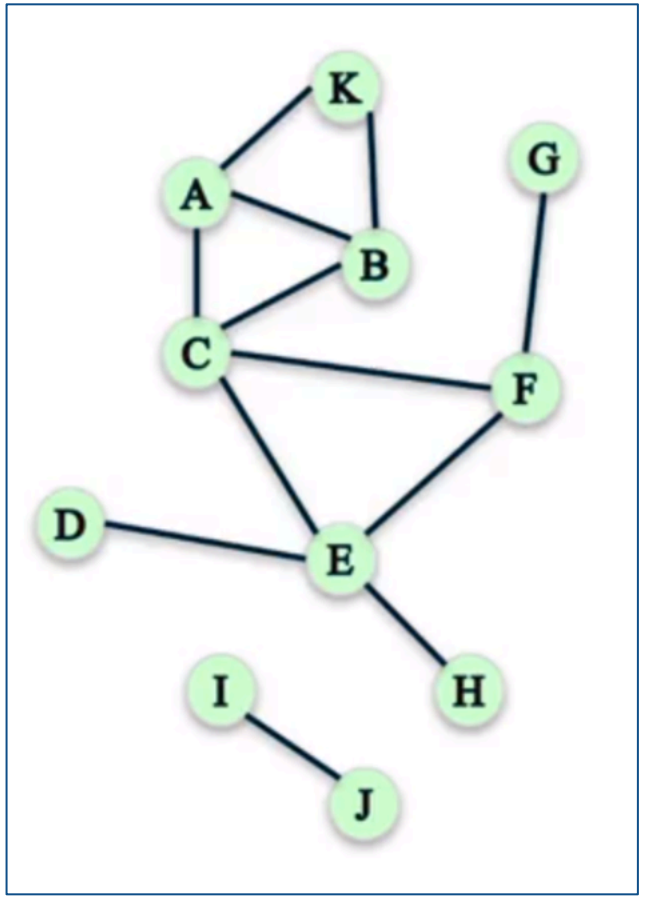
Clustering coefficient

How do we measure grouping?

Local Clustering Coefficient

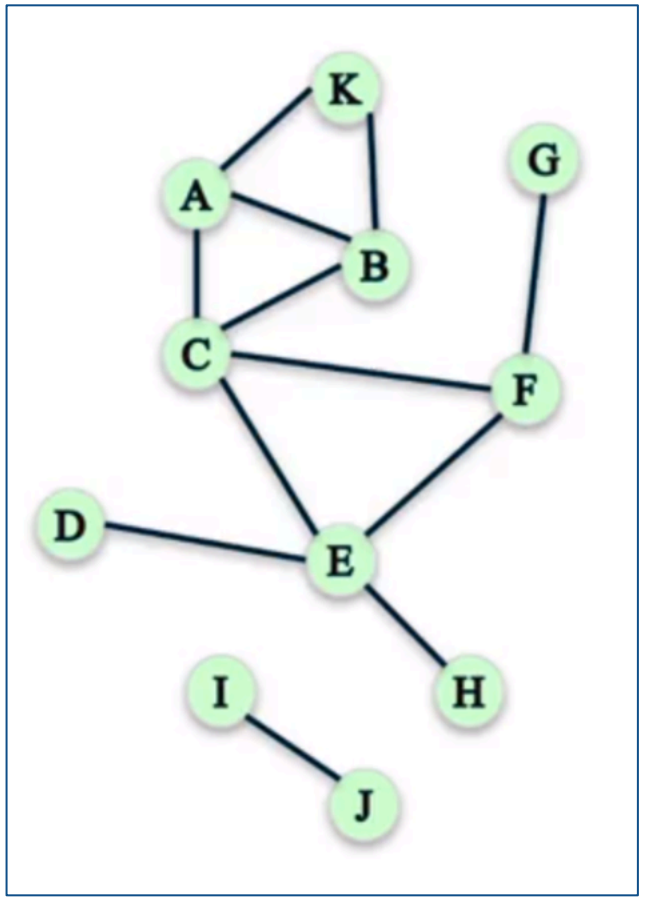
- **LCC**: percentage of a node's neighbors that and also each other's neighbors.
  - d_C: C's neighbors = 4
  - p_C: pairs of C's neighbors that are each other's neighbors = 2
  - n_C: pairs of C's neighbors = $d_C(d_C - 1)/2 = 6$
  - $LCC_C = p_C/n_C = 2/6 = 33.33\%$

```
G = nx.Graph()
G.add_edge_from([('C','E'),('C','F'),('E','F'),(
'F','G')])
nx.clustering(G, 'F')
# 0.3333
```



Global Clustering Coefficient and Transitivity

- **GCC**: clustering over the whole network.
- **Transitivity**: ratio between triangles open triads.
  - $transitivity_G$ = 3*triangles / open triads




```
nx.average_clustering(G)
# 0.2878

nx.transitivity(G)
# 0.4091
```

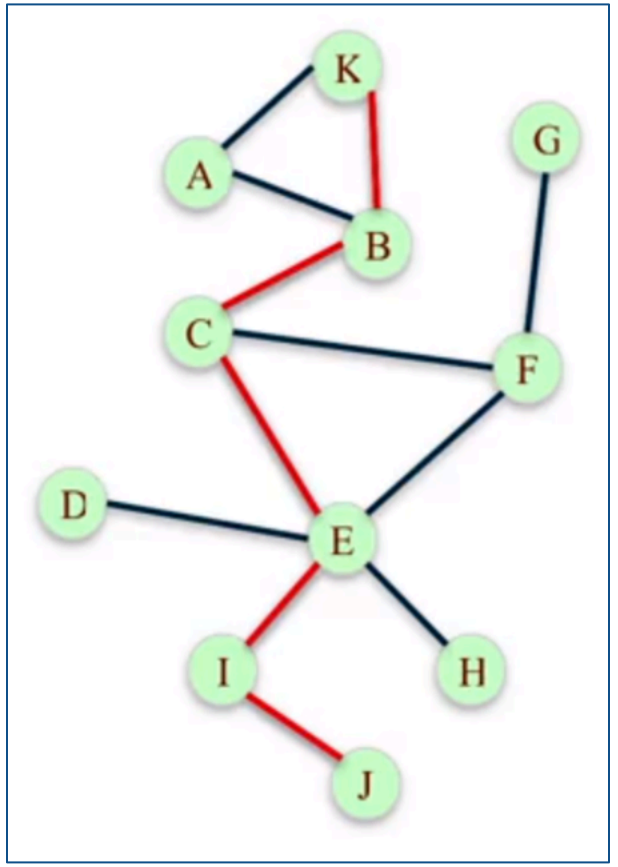
Distance measures

How do we measure distance?

Distance measures

- **Average distance**: between all nodes.
- **Diameter**: max distance between nodes.
- **Eccentricity**: largest distance from one node to all other nodes.
- **Radius**: minimum eccentricity.

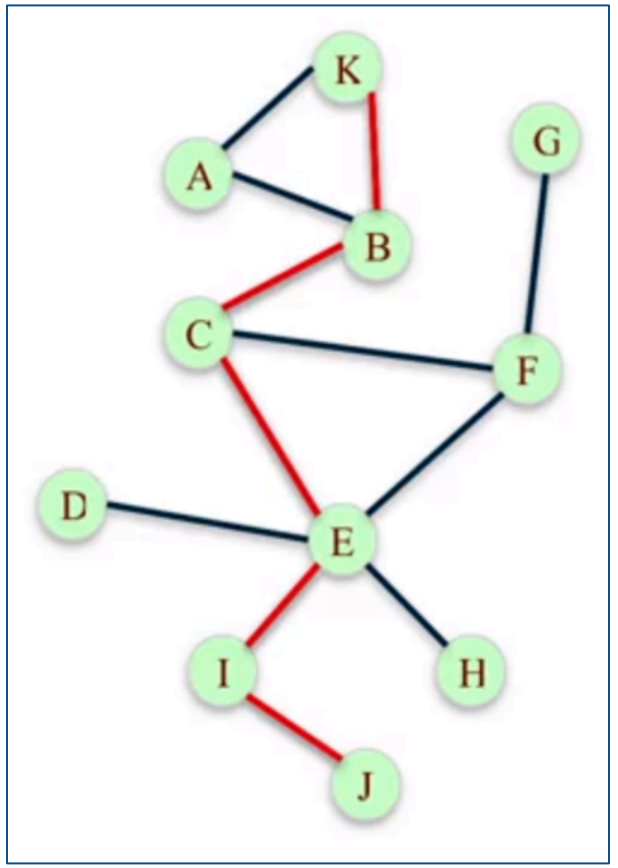
```
nx.average_shortest_path_ength(G)
# 2.5272
nx.diameter(G)
# 5
nx.eccentricity(G)
# {‘A’:5,‘B’:4,‘C’:3,‘D’:4, …}
nx.radius(G)
# 4
```



Distance measures

- **Periphery**: nodes with eccentricity equal to the diameter.
- **Center**: nodes with eccentricity equal to the radius.

```
nx.periphery(G)
# ['A','K','J']

nx.center(G)
# ['C','E','F']
```



Centrality Measures

How do we measure importance?

Degree centrality (DC)

- Assumption: important nodes **have many connections**.
  - Knowledgeable people with many connections
- For undirected graph: edges of a node.
- For digraph: in-degree or out-degree of a node.

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

N = set of nodes in the network,

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

```
nx.degree_centrality(G)
nx.in_degree_centrality(G)
nx.out_degree_centrality(G)
```

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

Closeness centrality (CC)

- Average closeness of a node to others.
- Assumption: important nodes **are close to other nodes**.

```
nx.closeness_centrality(G)
```

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

N = set of nodes in the network,

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

Betweenness centrality (BC)

- Assumption: important nodes **connect to other nodes**.
  - Knowledgeable people with many connections
- Take shortest path between nodes.
  - High presence in shortest paths means high betweenness

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

```
nx.betweenness_centrality(G)
```

Next time…

- Social network analysis
- EX3: SNA with NetworkX
- Hands-on Exam