Network Elements

Network Science (I606)

The plan

- We have seen a bunch of interesting networks arising in many different domains
- Now let us learn the language of networks
  - The components: nodes, links
  - Types of networks and representations
  - Features of nodes and links

We want to be able to talk about...

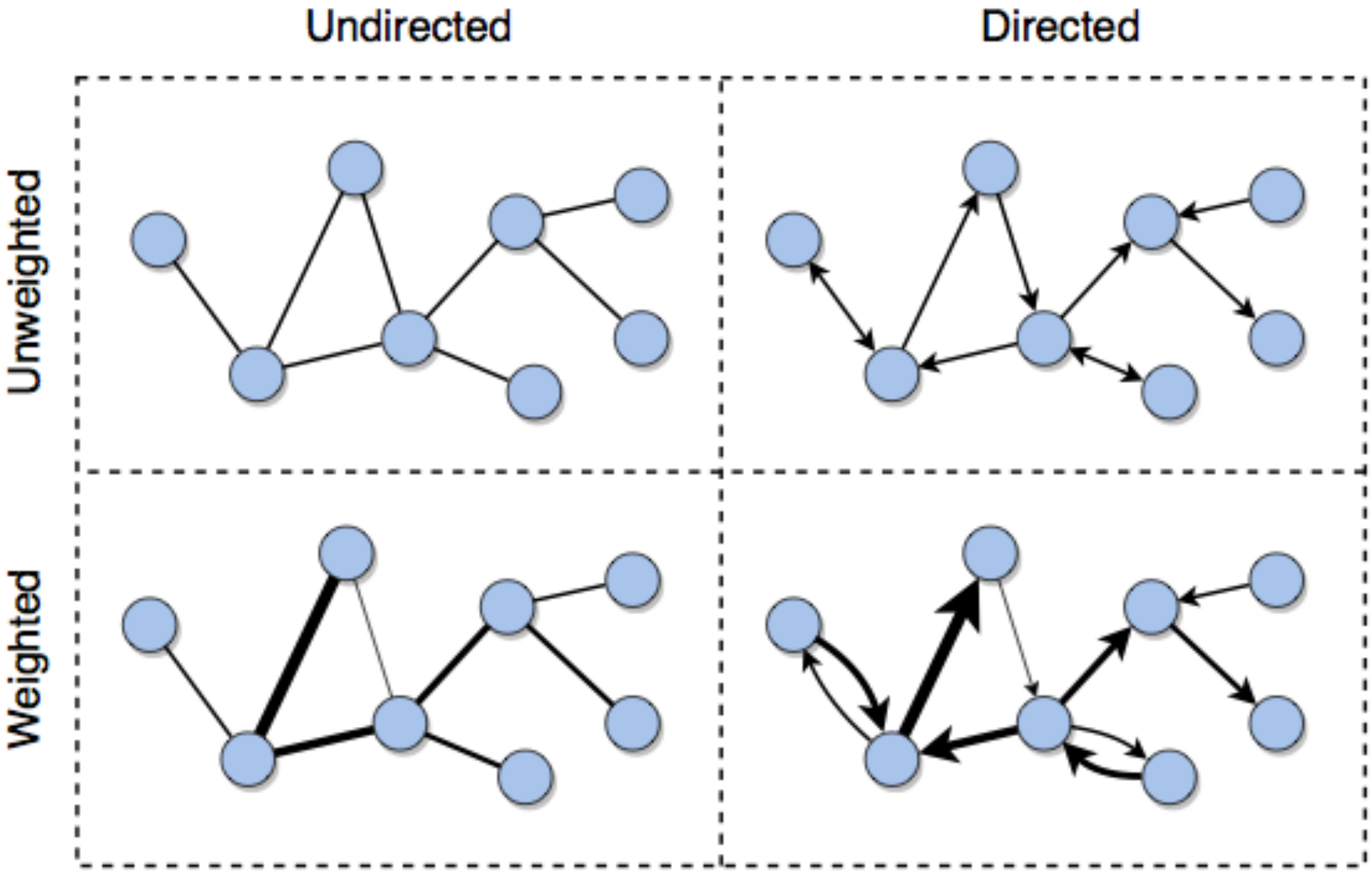
- properties to characterize structure & behavior of networks
- roles of networks in affecting processes occurring on network structures

Formal definitions

- A **network** or **graph** *G* has two parts, a set of *N* elements, called **nodes** or **vertices**, and a set of *L* pairs of nodes, called **links** or **edges**. The link *(i, j)* joins the nodes *i* and *j*. Two nodes are **adjacent** or **connected** or **neighbors** if there is a link between them.

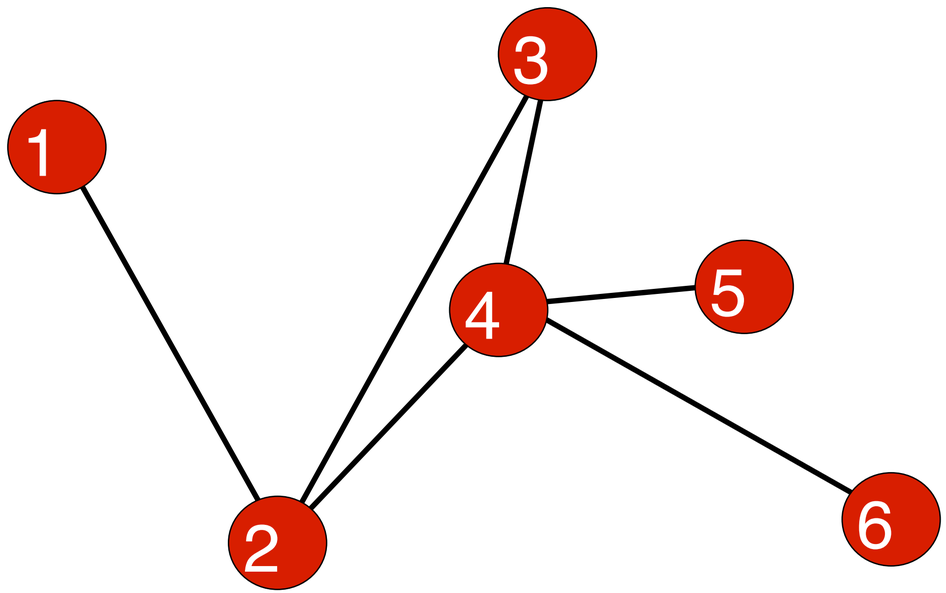
- A network can be **undirected** or **directed**. A directed network is also called a **digraph**. In directed networks, links are called **directed links** and the order of the nodes in a link reflects the direction: the link *(i, j)* goes from the **source** node *i* to the **target** node *j*. In undirected networks, all links are bi-directional and the order of the two nodes in a link does not matter.

- A network can be **unweighted** or **weighted**. In a weighted network, links have associated **weights**: the **weighted link** *(i,j,w)* between nodes *i* and *j* has weight *w*. A network can be both directed and weighted, in which case it has directed weighted links.



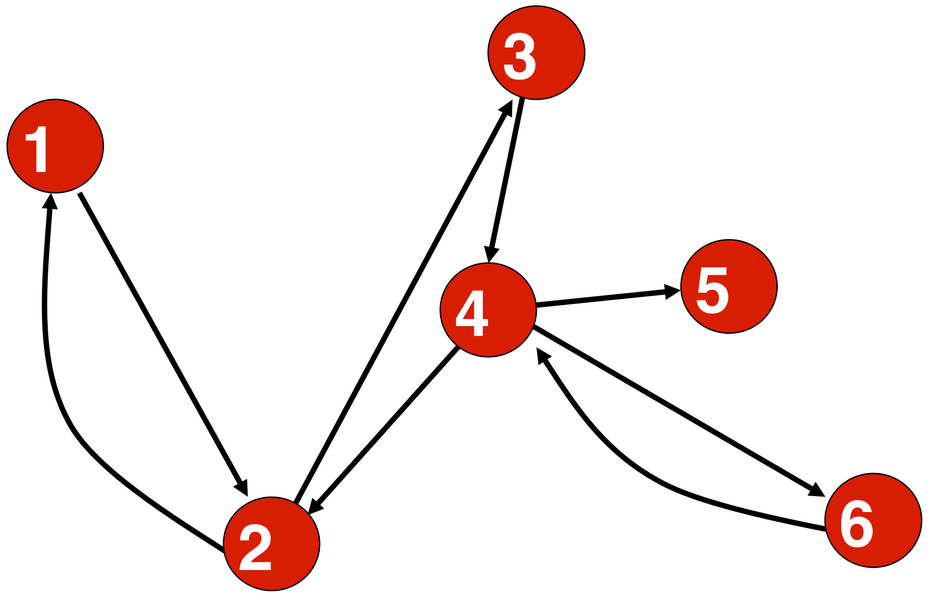
Can you think of a few examples in each of these categories?

Python and NetworkX



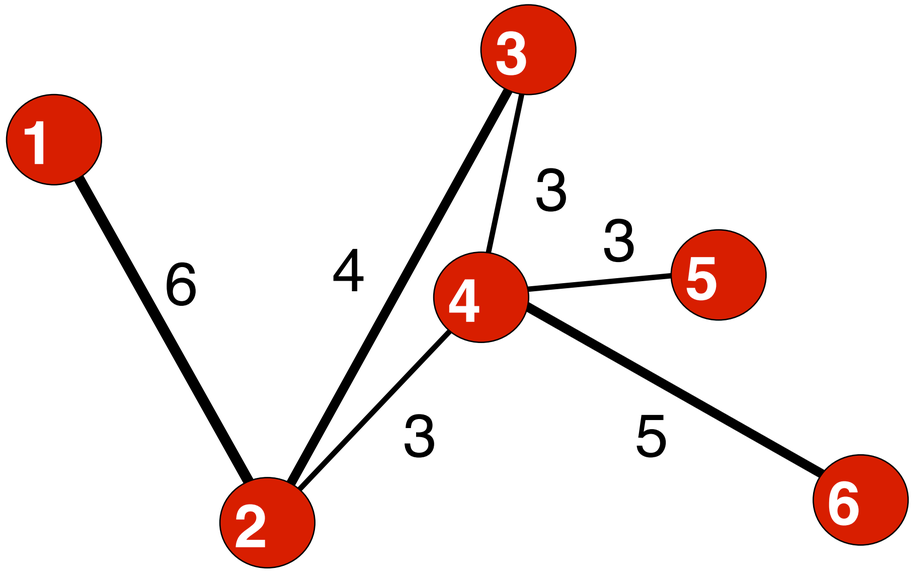

```
import networkx as nx # always!

G = nx.Graph()
G.add_node(1)
G.add_nodes_from([2,3,…])
…
G.add_edge(1,2)
G.add_edges_from([(2,3),(2,4),…])
…
G.nodes()
G.edges()
G.neighbors(4)

for n in G.nodes:
    print(n, G.neighbors(n))
for u,v in G.edges:
    print(u, v)
```

Directed networks




```
import networkx as nx # don’t forget!

D = nx.DiGraph()
D.add_edge(1,2)
D.add_edge(2,1)
D.add_edges_from([(2,3),(3,4),…])
…
D.number_of_nodes()
D.number_of_edges()
D.edges()
D.successors(2)
D.predecessors(2)
D.neighbors(2)
```

Weighted networks

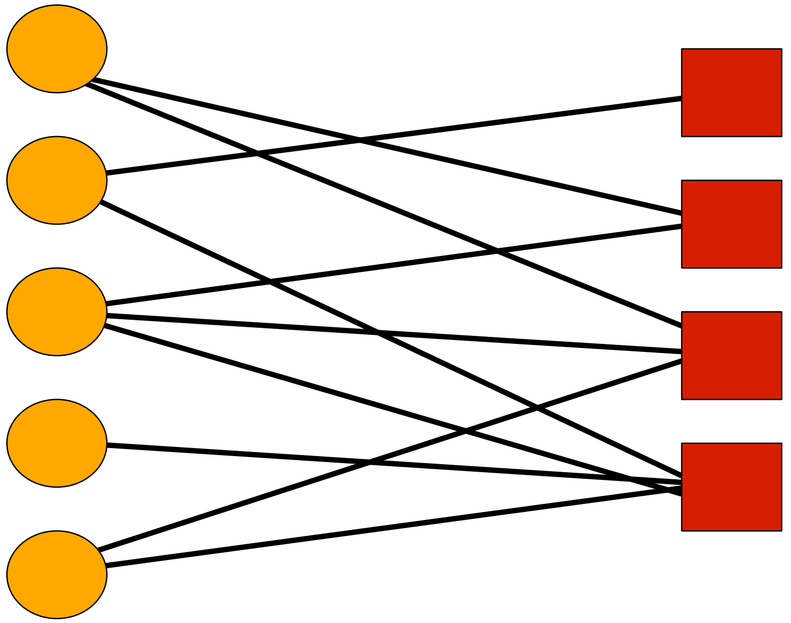



```
W = nx.Graph()
W.add_edge(1,2,weight=6)
…
W.add_weighted_edges_from([(4,5,3),(4,6,5),…])
…
W.edges()
W.edges(data='weight')
for (u,v,d) in W.edges(data='weight'):
    if d>3:
        print('(%d, %d, %d)'%(u,v,d))
```

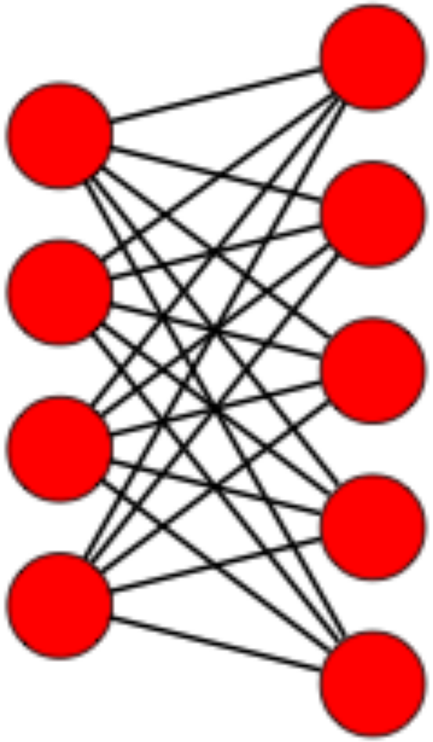
Bipartite networks

- In a **bipartite** network, there are two groups of nodes such that links only connect nodes from different groups and not nodes from the same group

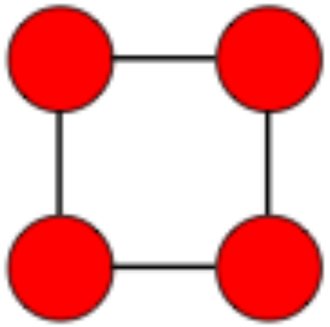
- Examples of bipartite networks include those that capture the relationships between movies and stars, between songs and artists, between classes and students, and between products and customers



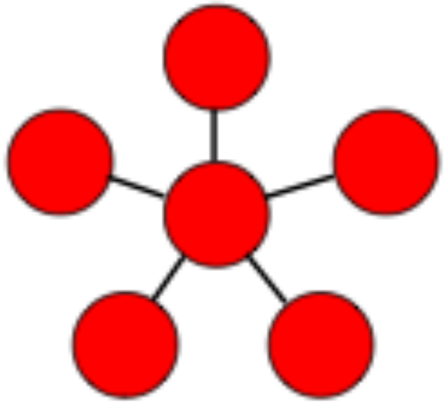
B = nx.complete_bipartite_graph(4,5)



C = nx.cycle_graph(4)



S = nx.star_graph(6)



P = nx.path_graph(5)

Many networks generators

Density and sparsity

- Network size N = number of nodes
- L = number of links
- Maximum possible number of links: $L_{max} = \binom{N}{2} = \frac{N(N-1)}{2}$
- Density: $d = \frac{L}{L_{max}} = \frac{2L}{N(N-1)}$
- The network is **sparse** if $d \ll 1$

Density and sparsity

- In a **directed** network things are a bit different…
  - Maximum possible number of links: $L_{max} = N(N-1)$
  - Density: $d = \frac{L}{L_{max}} = \frac{L}{N(N-1)}$
- In a **complete** network, all pairs of nodes are connected and $d = 1$

```
G.number_of_nodes()
G.number_of_edges()
nx.density(G)
nx.density(D)

CG = nx.complete_graph(8471)
print(nx.density(CG)) # what does this print?
```

Example: Facebook

- Rough orders-of-magnitude approximations:
- $N \approx 10^9$
- $L \approx 10^3 \times N$
- $d \approx L / N^2 \approx 10^3 N / N^2 \approx 10^3 / 10^9 = 10^{-6}$
- Most (but not all) real-world networks are similarly sparse because the number of links scales proportionally to N, whereas the maximum scales with N^2

Table 1.1 Basic statistics of network examples. Network types can be (D)irected and/or (W)eighted. When there is no label the network is undirected and unweighted. For directed networks, we provide the average in-degree (which coincides with the average out-degree).

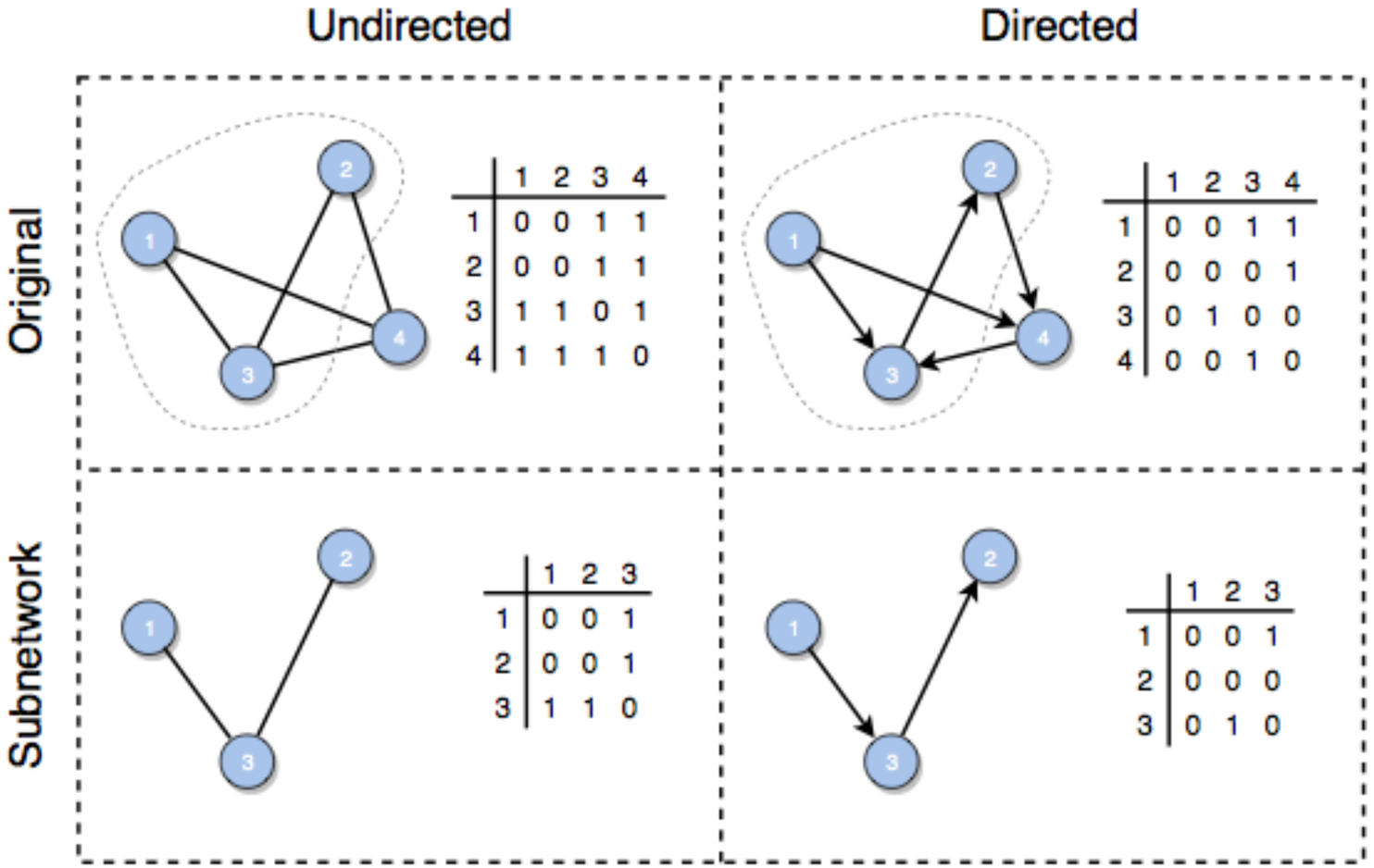
Network	Type	Nodes (N)	Links (L)	Density (d)	Average degree ($\langle k \rangle$)
Facebook Northwestern Univ.		10,567	488,337	0.009	92.4
IMDB movies and stars		563,443	921,160	0.000006	3.3
IMDB co-stars	W	252,999	1,015,187	0.00003	8.0
Twitter US politics	DW	18,470	48,365	0.0001	2.6
Enron Email	DW	87,273	321,918	0.00004	3.7
Wikipedia math	D	15,220	194,103	0.0008	12.8
Internet routers		190,914	607,610	0.00003	6.4
US air transportation		546	2,781	0.02	10.2
World air transportation		3,179	18,617	0.004	11.7
Yeast protein interactions		1,870	2,277	0.001	2.4
C. elegans brain	DW	297	2,345	0.03	7.9
Everglades ecological food web	DW	69	916	0.2	13.3

Subnetworks

A **subnetwork** is a network obtained by selecting a subset of the nodes and all of the links among these nodes

```
S = nx.subgraph(G, node_list)
```

A **clique** is a complete subnetwork



Degree

- The **degree** of a node is its number of links, or neighbors
- We typically use k_i to denote the degree of node i
- A node without neighbors is called a **singleton** (k=0)

```
G.degree(2) # returns the degree of node 2
G.degree()  # dict with the degree of all nodes of G
```

Degree

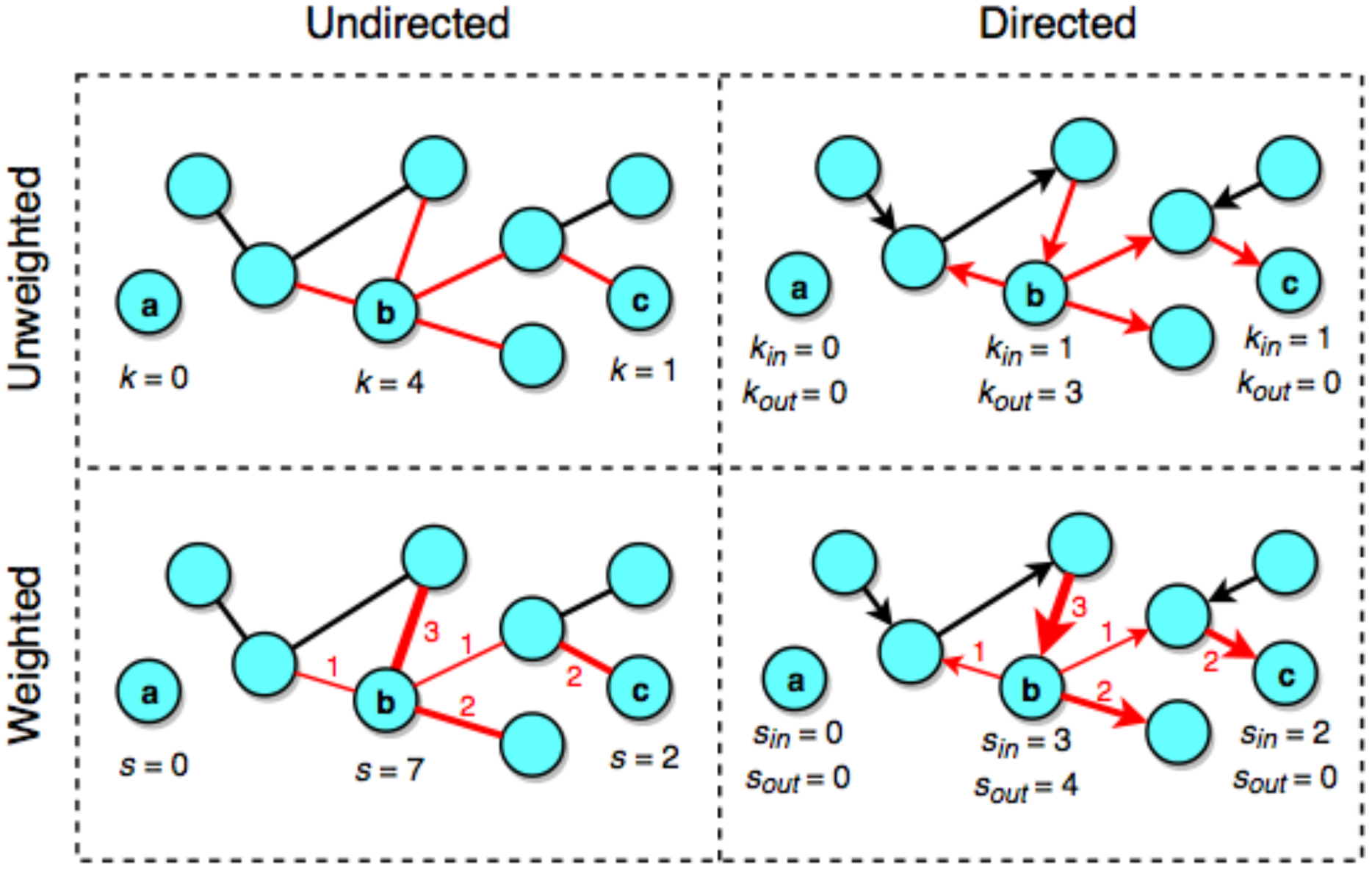
- In a directed network we have
  - **in-degree** of a node = number of incoming links k^{in}_i
  - **out-degree** of a node = number of outgoing links k^{out}_i

```
D.in_degree(4)
D.out_degree(4)
D.degree(4)
```

Strength

- In a weighted network we have **strength** $s_i = \sum_j w_{ij}$ (a.k.a. **weighted degree**)
- In a weighted directed network we have
  - **in-strength** $s_i^{in} = \sum_j w_{ji}$
  - **out-strength** $s_i^{out} = \sum_j w_{ij}$

```
W.degree(4) # degree
W.degree(4, weight='weight') # strength
```


Undirected
Directed
Unweighted
Weighted
a
b
c
k = 0
k = 4
k = 1
$k_{in} = 0$
$k_{out} = 0$
$k_{in} = 1$
$k_{out} = 3$
$k_{in} = 1$
$k_{out} = 0$
s = 0
s = 7
s = 2
$s_{in} = 0$
$s_{out} = 0$
$s_{in} = 3$
$s_{out} = 4$
$s_{in} = 2$
$s_{out} = 0$
1
3
1
2
2
1
3
1
2
2

Average degree

- The **average degree** of a network is $\langle k \rangle = \frac{\sum_i k_i}{N}$
- We can connect network size, number of links, density, and average degree
- In undirected networks:

$$\langle k \rangle = \frac{2L}{N} = \frac{dN(N-1)}{N} = d(N-1)$$

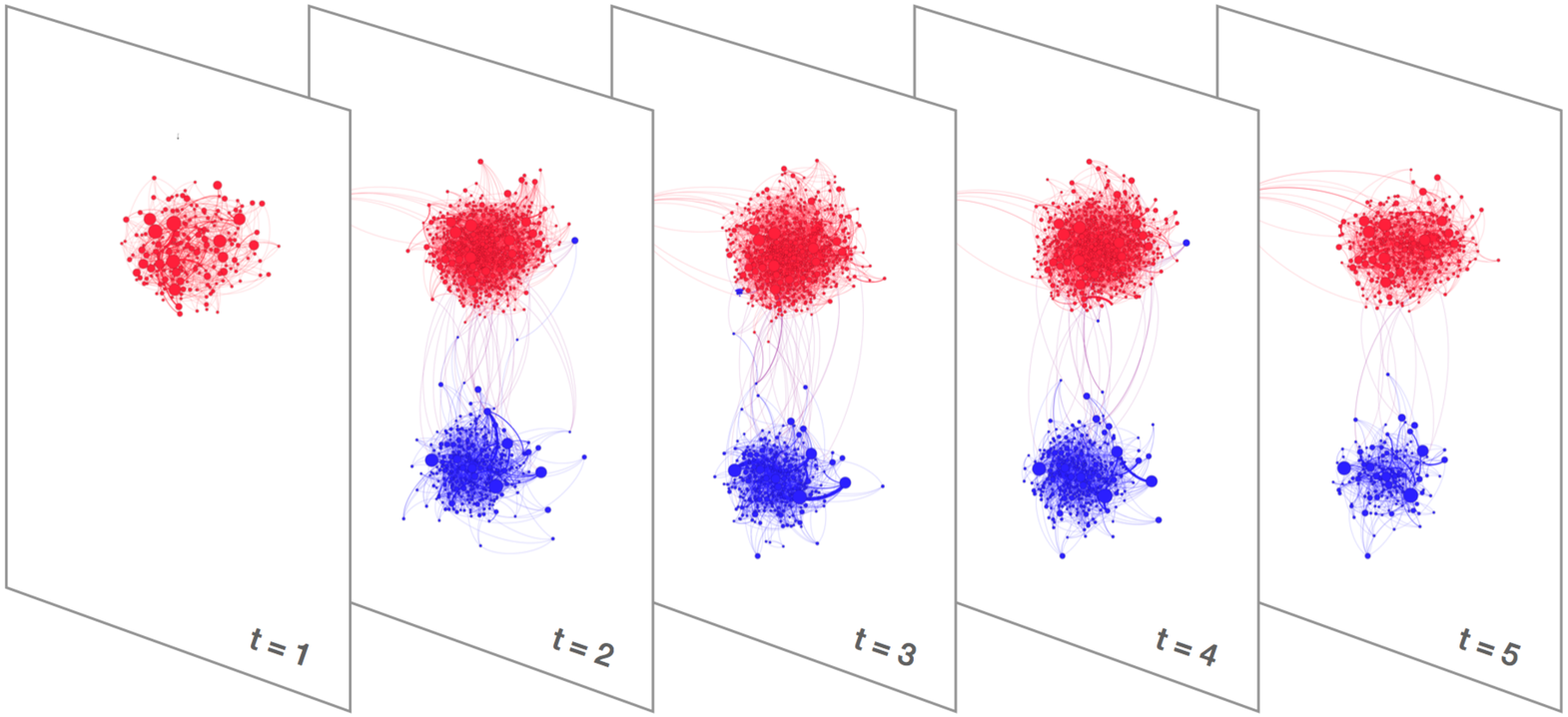
$$d = \frac{\langle k \rangle}{N-1} = \frac{\langle k \rangle}{k_{max}}$$

Multilayer networks

- A network can have multiple **layers**, each with its own nodes and edges
  - Example: air transportation networks of distinct airlines, with some but not complete overlap of airport nodes
- **Intralayer links** among nodes in the same layer, **interlayer links** across layers
- If the sets of nodes in the different layers are identical, we call the network a **multiplex**; interlayer links are **couplings** linking the same node across layers
  - Example: layers to represent different types of relationship in a social network, such as friendship, family ties, coworkers, etc.

Temporal networks

- A **temporal network** is a multiplex in which the layers represent links at different times (temporal snapshots)
  - Example: a Twitter retweet network

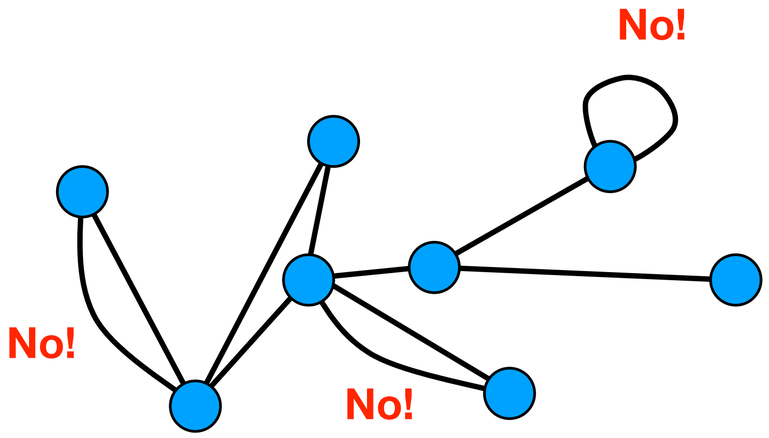


Networks of networks

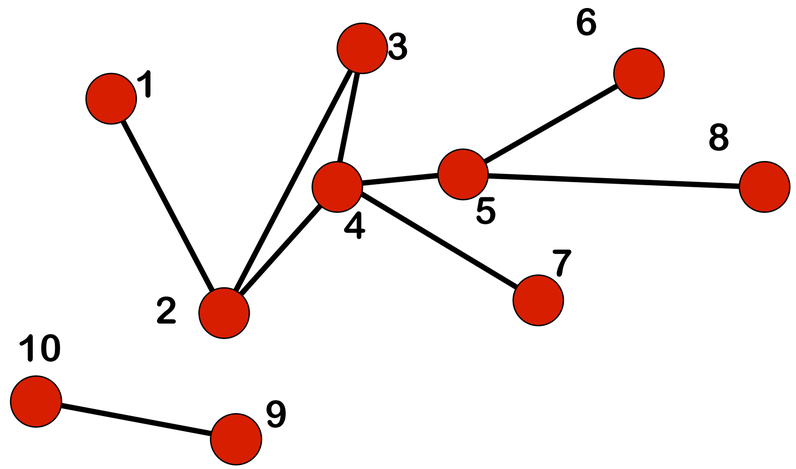
- In general, each layer in a multilayer network can have its own nodes and edges. We call this a **network of networks**
  - Examples: the electrical power grid and the Internet; also the Internet itself!
- The interactions between networks in the different layers are captured by the interlayer links
  - Example: power stations communicate via the Internet, Internet routers are powered by the power grid
  - **Cascading failures** are one type of unpredictable vulnerabilities that arise in networks of networks

Simplifying assumptions

- We will assume:
  - single-layer networks with a single type of nodes and a single type of link
  - no self-loops
  - at most a single link between two nodes (possibly two links with opposite directions in directed networks)



Network representations



- **Adjacency matrix**: *N* x *N* matrix where each element $a_{ij} = 1$ if *i* and *j* are adjacent, zero otherwise

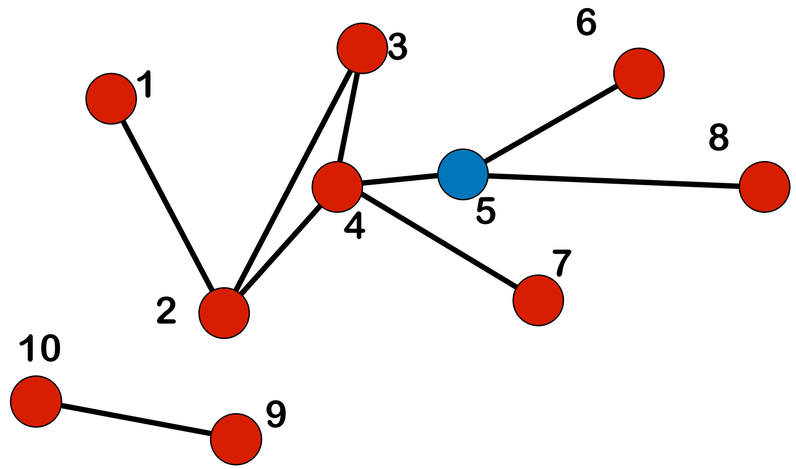
- The diagonal elements are zero because we have no self-loops

- In undirected networks, the matrix is symmetric: $a_{ij} = a_{ji}$

```
nx.adjacency_matrix(G)
print(nx.adjacency_matrix(G))
G.edge[3][4]
G.edge[3][4]['color']='blue'
G.edge[3][4]
G.edge[4]
```

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0
4	0	1	1	0	1	0	1	0	0	0
5	0	0	0	1	0	1	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	1	0

Network representations

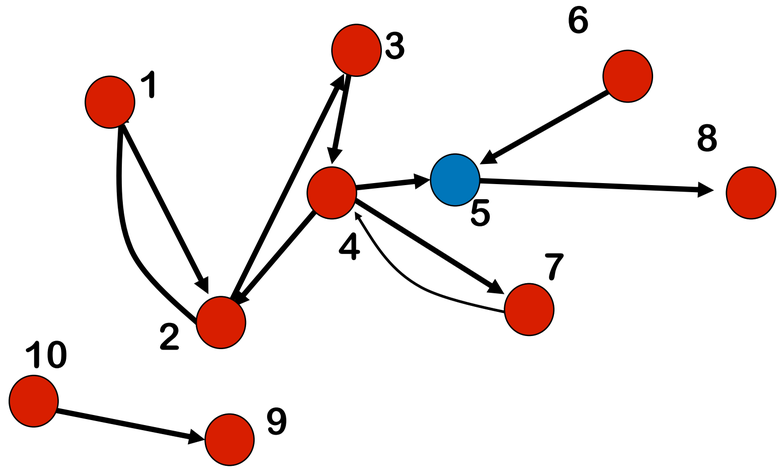


- In undirected networks, the degree is obtained by summing adjacency matrix elements across rows or columns:

$$k_i = \sum_j a_{ij} = \sum_j a_{ji}$$

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0
4	0	1	1	0	1	0	1	0	0	0
5	0	0	0	1	0	1	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	1	0

Network representations

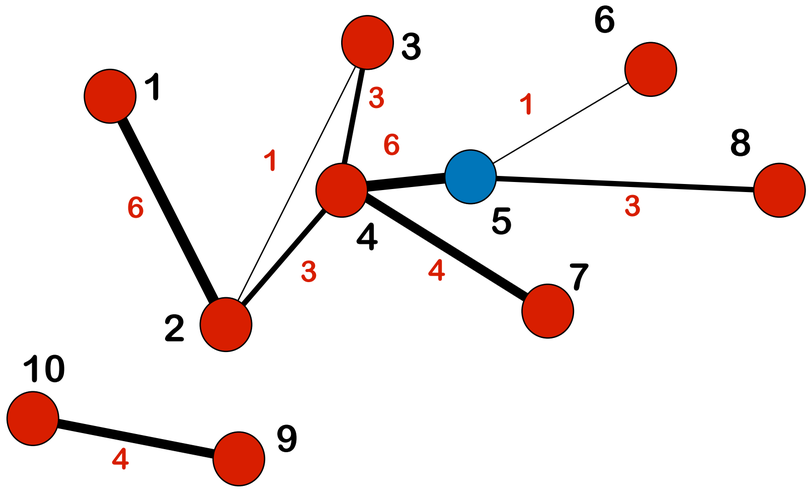


- In directed networks, the adjacency matrix is **not** symmetric
- The out-degree is obtained by summing adjacency matrix elements across rows: $k_i^{out} = \sum_j a_{ij}$
- The in-degree is obtained by summing adjacency matrix elements across columns: $k_i^{in} = \sum_j a_{ji}$

```
print(nx.adjacency_matrix(D))
D.edge[3][4]
D.edge[4][3]
D.edge[4]
```

	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	1	0	0	1	0	1	0	0	0
5	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	0

Network representations



- In weighted networks, each element w_{ij} represents the weight of the link between *i* and *j*, zero if there is no link
- If undirected, the strength is obtained by summing adjacency matrix elements across rows or columns
- If directed, the in/out-strength is obtained by summing adjacency matrix elements across columns/rows

```
print(nx.adjacency_matrix(W))
W.edge[2][3]
W.edge[2]
W.edge[2][3]['weight'] = 2
W.edge[2][3]
W.edge[2]
```

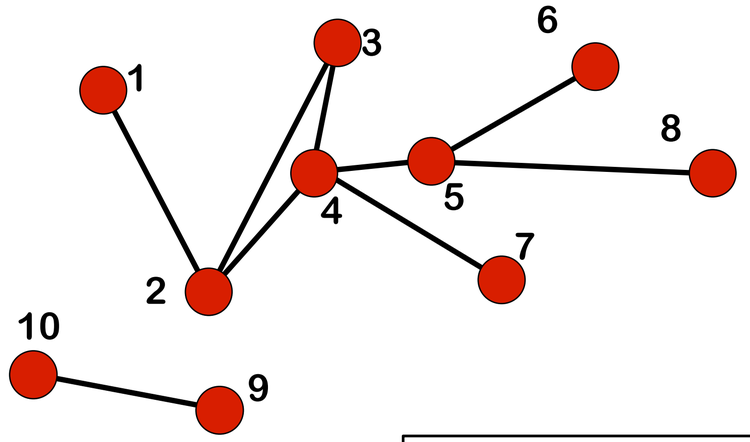
	1	2	3	4	5	6	7	8	9	10
1	0	6	0	0	0	0	0	0	0	0
2	6	0	1	3	0	0	0	0	0	0
3	0	1	0	3	0	0	0	0	0	0
4	0	3	3	0	6	0	4	0	0	0
5	0	0	0	6	0	1	0	3	0	0
6	0	0	0	0	1	0	0	0	0	0
7	0	0	0	4	0	0	0	0	0	0
8	0	0	0	0	3	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	4
10	0	0	0	0	0	0	0	0	4	0

Sparse network representations

- The memory/disk storage needed by an adjacency list is proportional to N^2
- In sparse networks (most real-world networks), this is terribly inefficient: most of the space is wasted storing zeros (non-links); for very large networks, adjacency lists are unfeasible
- It is much more efficient, often necessary, to store only the actual links, and assume that if a link is not listed it means it is not present
- There are two commonly used sparse networks representations:
  - **Adjacency list**
  - **Edge list**

Adjacency list

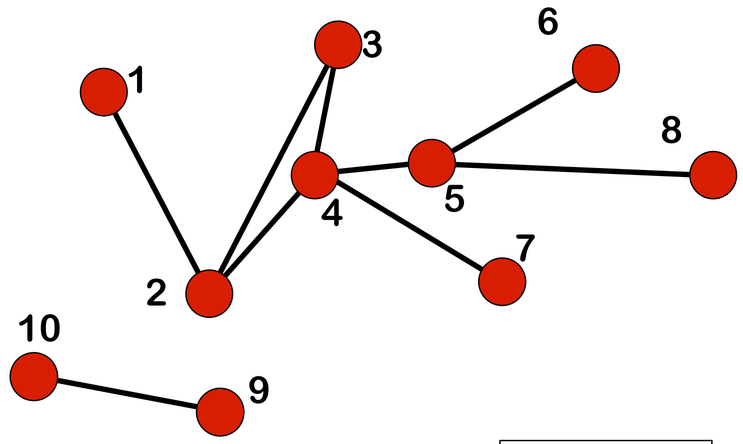
- List of neighbors for each node
- In undirected networks, each link is listed twice
- In weighted networks, each neighbor is replaced by a pair (neighbor, weight)




```
1    2
2    1 3 4
3    2 4
4    2 3 5 7
5    4 6 8
6    5
7    4
8    5
9    10
10   9
```

```
G.neighbors(2)

for n,neighbors in G.adjacency():
    for neighbor,link_attributes in neighbors.items():
        print('(%d, %d)' % (n,neighbor))

nx.write_adjlist(G, "netfile.adjlist")
G2 = nx.read_adjlist("netfile.adjlist") # G and G2 are isomorphic
```

Edge list



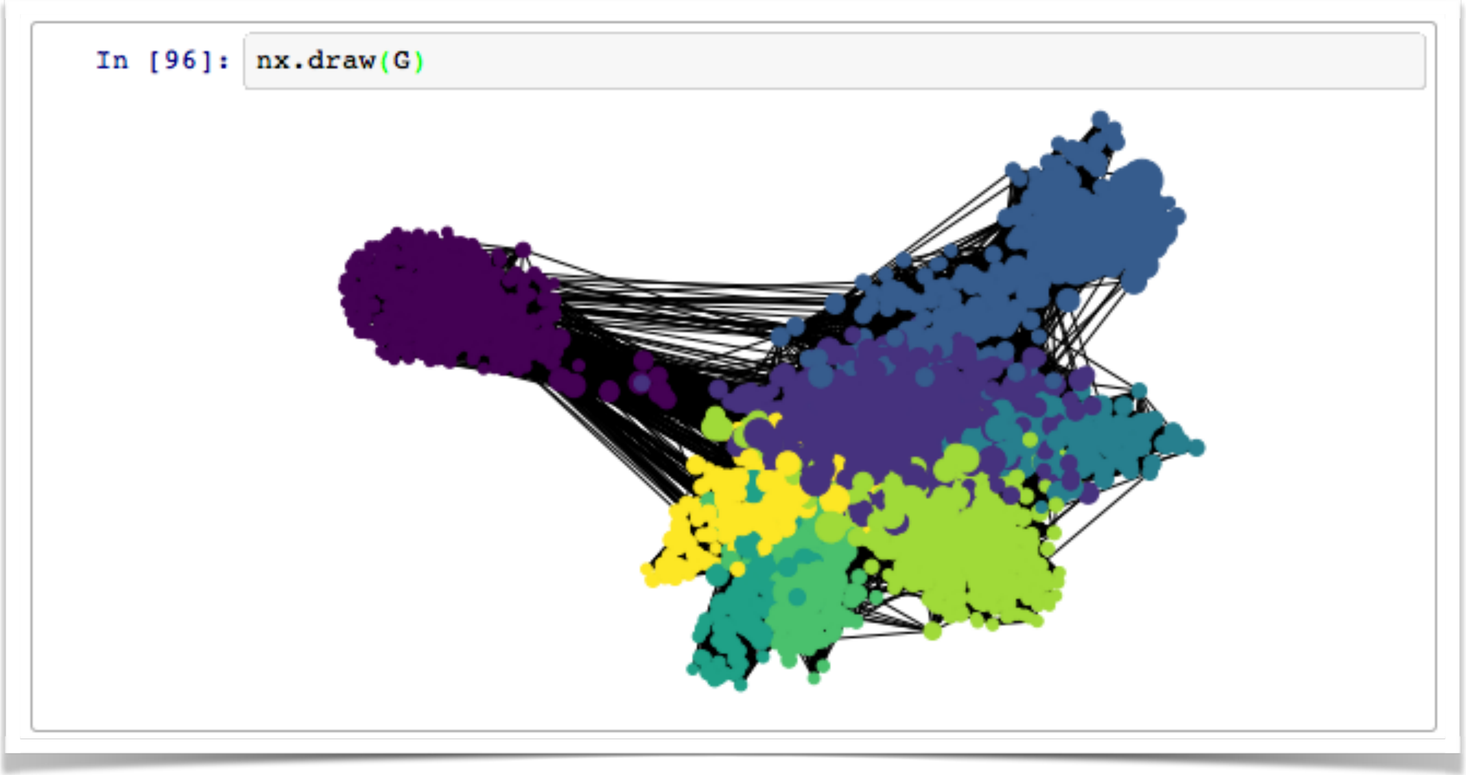
- List of node pairs that are connected
- In weighted networks, each pair is replaced by a triple (i, j, weight)

```
for i,j in G.edges:
    print('%d %d' %(i,j))

nx.write_edgelist(G, "netfile.edgelist")
G3 = nx.read_edgelist("netfile.edgelist") # G and G3 are isomorphic

nx.write_weighted_edgelist(W, "wf.edges") # store  weights
W2 = nx.read_weighted_edgelist("wf.edges") # W and W2 are isomorphic
```

```
1 2
2 3
2 4
3 4
4 5
4 7
5 6
5 8
9 10
```

Drawing networks

```
import matplotlib.pyplot
nx.draw(G, node_color=colors, node_size=sizes)
```



- A **network layout algorithm** places nodes on a plane to visualize the structure of the network
- There are many layout algorithms; the most commonly used are **force-directed layout** (a.k.a. **spring layout**) algorithms:
  - Connected nodes are placed near each other
  - Links have similar length
  - Link crossings are minimized
- This is done by simulating a physical systems where adjacent nodes are connected by springs and otherwise repel each other
- The community structure of the network can be revealed this way if the network is not too dense or too large