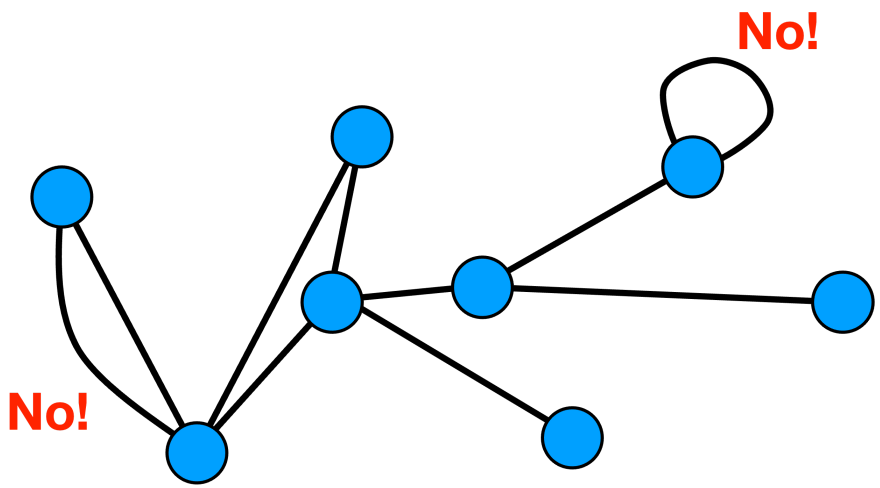
# Simplifying assumptions

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 (except for *directed* networks with links with opposite directions)



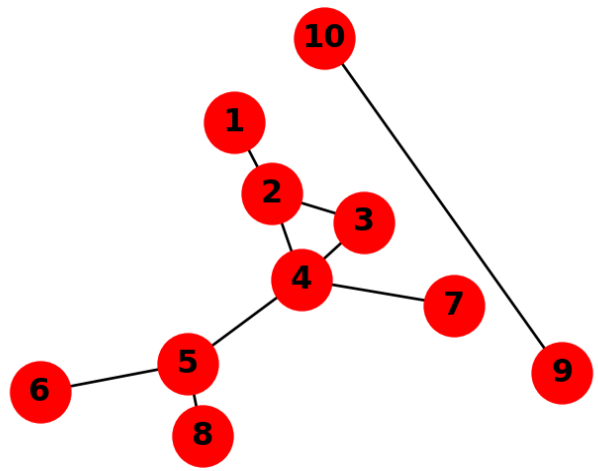
# Network representations (1/4)

**Adjacency matrix** : $N \times N$ matrix where each element $a_{ij} = 1$ if $i$ and $j$ are adjacent, $0$ otherwise

The diagonal elements (i.e., $a_{ii}$) are zero because we have no self-loops

In *undirected* networks, the matrix is symmetric: $a_{ij} = a_{ji} \qquad \forall i, j$

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



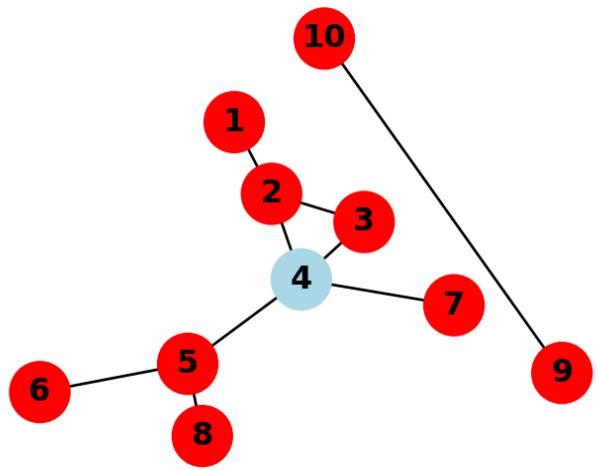

```
[[0 1 0 0 0 0 0 0 0 0]
 [1 0 1 1 0 0 0 0 0 0]
 [0 1 0 1 0 0 0 0 0 0]
 [0 1 1 0 1 0 1 0 0 0]
 [0 0 0 1 0 1 0 1 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1 0]]
```

```
{2: {}, 3: {}, 5: {}, 7: {}}
{2: {}, 3: {'color': 'blue'}, 5:
{}, 7: {}}
```

# Network representations (2/4)

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

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




```
[[0 1 0 0 0 0 0 0 0 0]
 [1 0 1 1 0 0 0 0 0 0]
 [0 1 0 1 0 0 0 0 0 0]
 [0 1 1 0 1 0 1 0 0 0]
 [0 0 0 1 0 1 0 1 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 1 0]]
```

**Why is $a_{ij}$ from $i$ to $j$ rather than from $j$ to $i$?**

Matrix multiplication! See these optional slides

# Network representations (3/4)

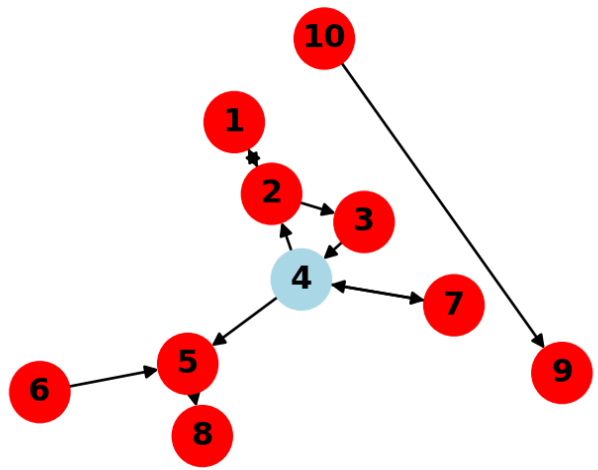
In *directed* networks, the adjacency matrix is **not** symmetric

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:

$$k_i^{in} = \sum_j a_{ji}$$




```
[[0 1 0 0 0 0 0 0 0 0]
 [1 0 1 0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 1 0 0 1 0 1 0 0 0]
 [0 0 0 0 0 0 0 1 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0 0 0 0]]
```

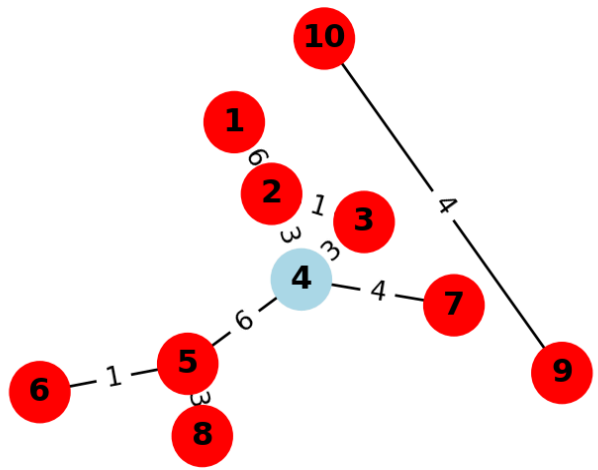
# Network representations (4/4)

In *weighted* networks, element $w_{ij}$ represents the weight of the link between $i$ and $j$. It is $0$ 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

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




```
[[0 6 0 0 0 0 0 0 0 0]
 [6 0 1 3 0 0 0 0 0 0]
 [0 1 0 3 0 0 0 0 0 0]
 [0 3 3 0 6 0 4 0 0 0]
 [0 0 0 6 0 1 0 3 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 4 0 0 0 0 0 0]
 [0 0 0 0 3 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 4]
 [0 0 0 0 0 0 0 0 4 0]]

16
```

# Quick exercise

What about multi-layer networks?

# 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:

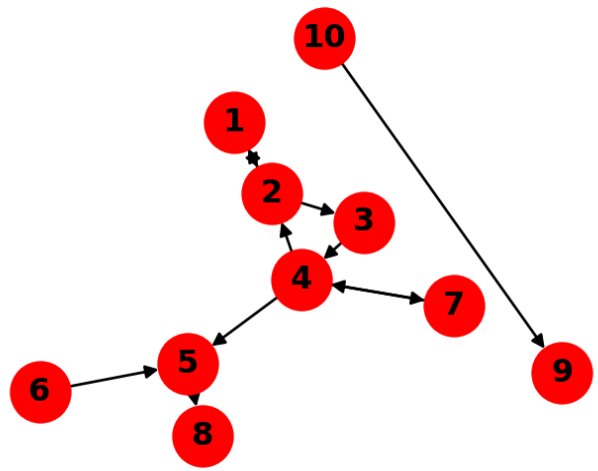
- **Adjacency list**
- **Edge list**

# Adjacency list

## List of neighbours for each node

```
for line in nx.generate_adjlist(G):
    print(line)

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




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

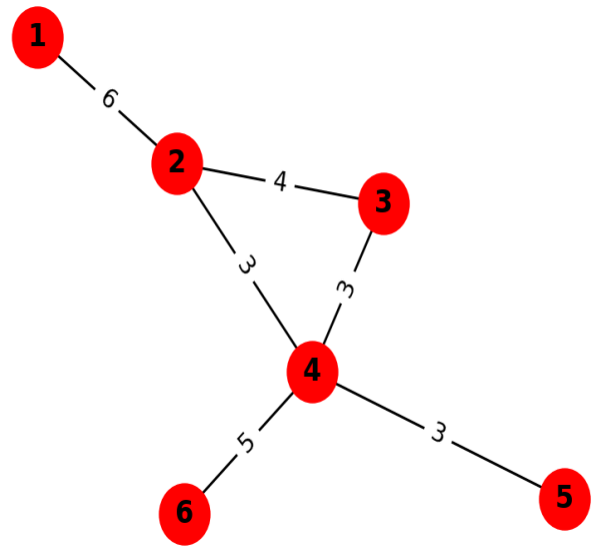
- In *undirected* networks, each link is listed **once** only.
- In *weighted* networks, each neighbour is replaced by a pair (neighbour, weight)

# Edge list

List of node pairs that are connected.

In *weighted* networks, each pair is replaced by a triple (i,j weight)

```
1 # Generate and print the edge list
2 for edge in nx.generate_edgelist(W, data=["weight"]):
3     print(edge)
4
5 nx.write_edgelist(G, "netfile.edgelist")
6 G2 = nx.read_edgelist("netfile.edgelist") # G and G2
7
8 nx.write_weighted_edgelist(W, "wf.edges") # store  we
9 W2 = nx.read_weighted_edgelist("wf.edges") # W and W2
```




```
1 2 6
2 3 4
2 4 3
3 4 3
4 5 3
4 6 5
```

# About network drawing

- 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. This simulates a physical system where adjacent nodes are connected by springs and otherwise repel each other:
  - → Connected nodes are placed near each other
  - → Links have similar length
  - → Link crossings are minimized

> **Warning**
>
> Do not to rely too much on the layout to infer network properties!

> **Note**
>
> Do explore the NetworkX documentation on layouts (e.g., during the lab classes)

# References


1. P. Holme & J. Saramäki, Temporal networks. *Physics Reports*, **519** (2012) 97–125. https://doi.org/10.1016/j.physrep.2012.03.001.
2. P. W. Holland, K. B. Laskey, & S. Leinhardt, Stochastic blockmodels: First steps. *Social Networks*, **5** (1983) 109–137. https://doi.org/10.1016/0378-8733(83)90021-7.
3. S. L. Feld, Why Your Friends Have More Friends Than You Do. *American Journal of Sociology*, **96** (1991) 1464–1477. https://doi.org/10.1086/229693.

# About adjacency matrices and matrix multiplication (1/3)

Suppose we have 3 nodes $X, Y, Z$ with the following *directed* edges:

- $X \to Y$
- $Y \to Z$
- $Z \to X$

The adjacency matrix is:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

What is $A^2 = A \cdot A$?

$$\mathbf{A^2} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Can you see what is happening?

# About adjacency matrices and matrix multiplication (2/3)

What is $\boldsymbol{A^3 = A^2 \cdot A}$?

$$\mathbf{A^3} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

What does it mean?

Can you see how useful this could be?

**Homework**

Design some simple *directed* or *undirected* networks, calculate $\boldsymbol{A^3}$ and build an intuition.

# Network Science

## *Small Worlds*

**Luc Berthouze**

***L.Berthouze@sussex.ac.uk***

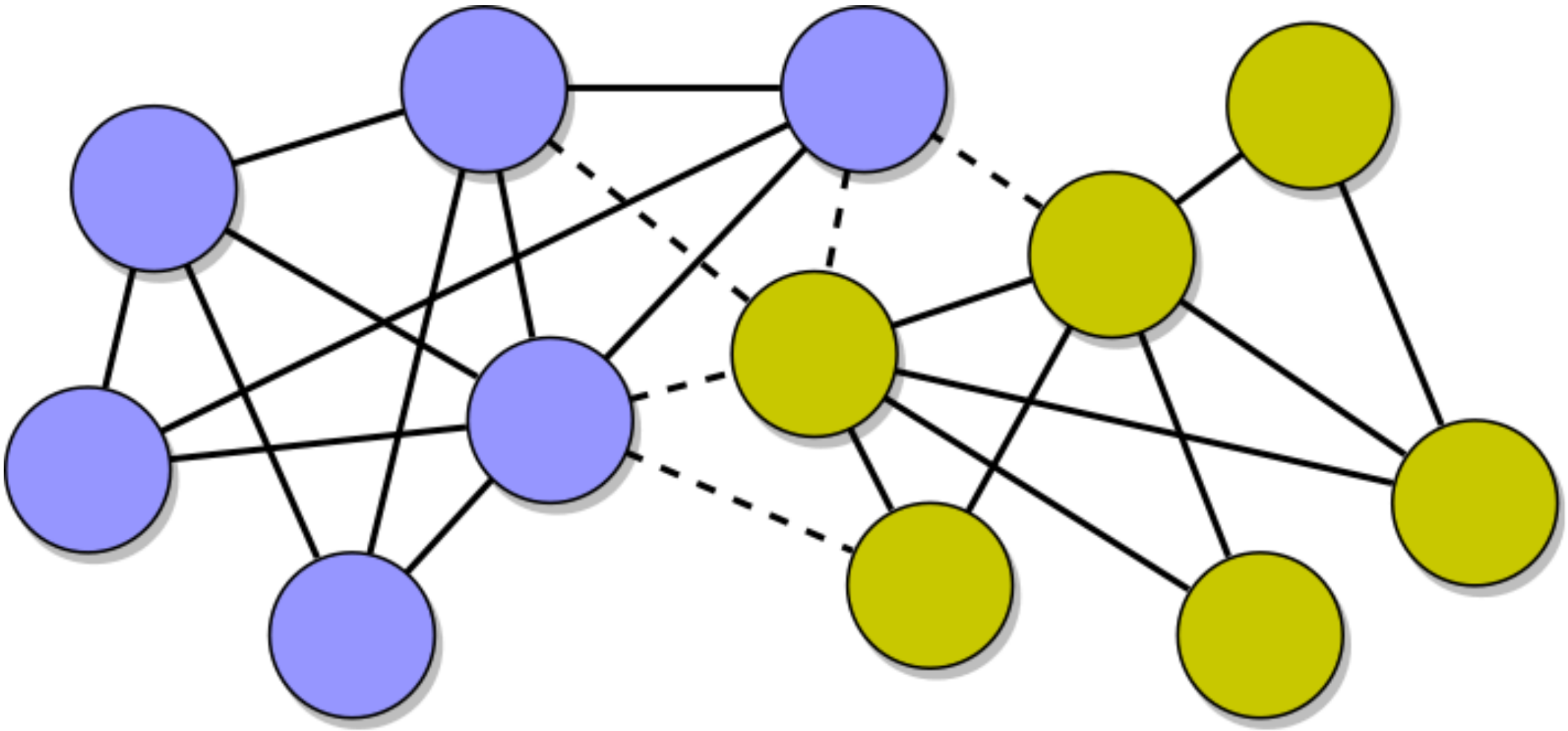
*University of Sussex*

February 12, 2025

# Chapter 2: Small Worlds

- Birds of a feather
- Paths and distances
- Connectedness and components
- Finding shortest paths
- Social distance
- Six degrees of separation
- Friend of a friend

# Birds of a feather



# Assortativity [1]

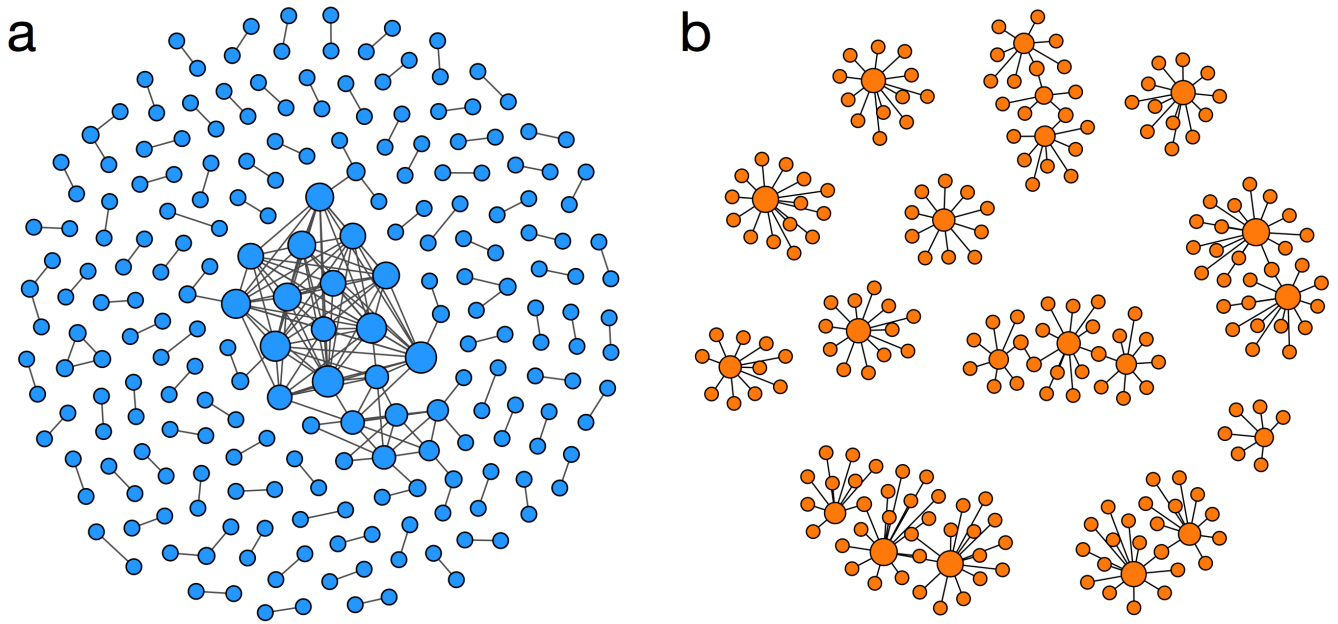
There are two possible mechanisms by which *assortativity* emerges naturally:

- **Selection** or **homophily**: similar nodes become connected
- (Social) **influence**: connected nodes become more similar

> **Note**
>
> Does similarity induce links? or do links induce similarity? In a future lecture, we will discuss the idea of *coevolution* in which network and dynamics (e.g, opinion) *adapt* to each other.

Assortativity is not necessarily a good thing. For example "**echo chambers**" and "**groupthink**" are situations where your friends are like you, diversity is killed, and you are only exposed to opinions that reinforce your pre-existing beliefs.

# Degree assortativity




- a.k.a. **degree correlation**
- Assortative networks have a **core-periphery** structure with hubs in the core
  - → Ex: social networks
- Disassortative networks have **hub-and-spoke** (or **star**) structure
  - → Ex: Web, Internet, food webs, bio networks

# Assortativity in NetworkX (1/3)

- based on an a categorical attribute, such as gender

```
assort_a = nx.attribute_assortativity_coefficient(G, category)
```

- based on a numerical attribute, such as age

```
assort_n = nx.numeric_assortativity_coefficient(G, quantity)
```

- based on degree (Pearson correlation between degree of adjacent nodes)

```
r = nx.degree_assortativity_coefficient(G)
```

# Assortativity in NetworkX (2/3)

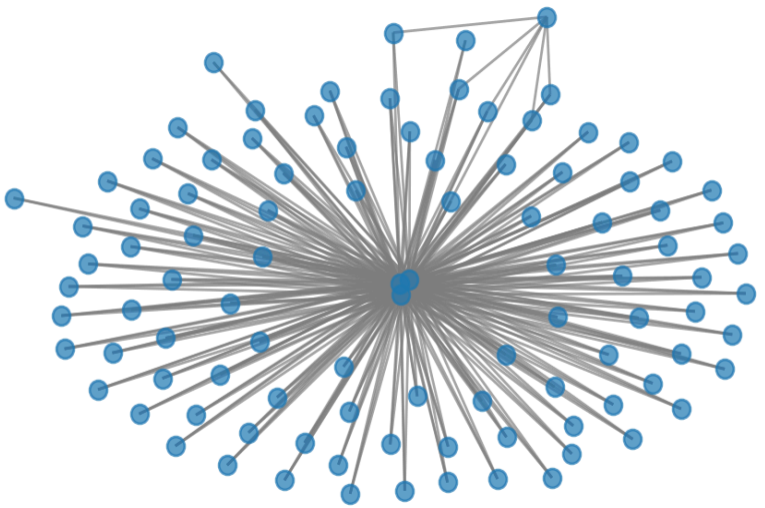
Another way to compute the degree assortativity is by measuring the correlation between the degree and the average degree of the neighbours of nodes with that degree:

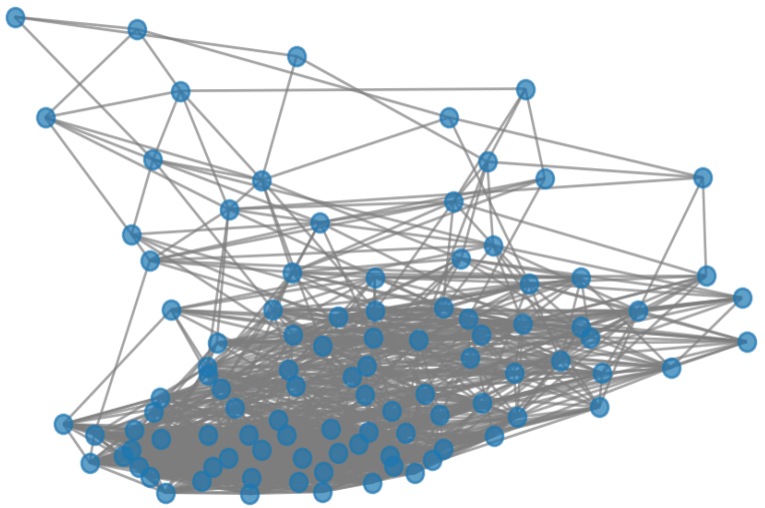
$$k_{nn}(i) = \frac{1}{k_i} \sum_j a_{ij} k_j$$

$$\langle k_{nn}(k) \rangle = \langle k_{nn}(i) \rangle_{i:k(i)=k}$$

```
1 import  scipy.stats
2
3 knn_dict = nx.average_degree_connectivity(G)
4 k, knn = list(knn_dict.keys()), list(knn_dict.values ())
5 r, p_value = scipy.stats.pearsonr(k, knn)
```

# Assortativity in NetworkX (3/3)





```
Low Assortativity Network: -0.961
```

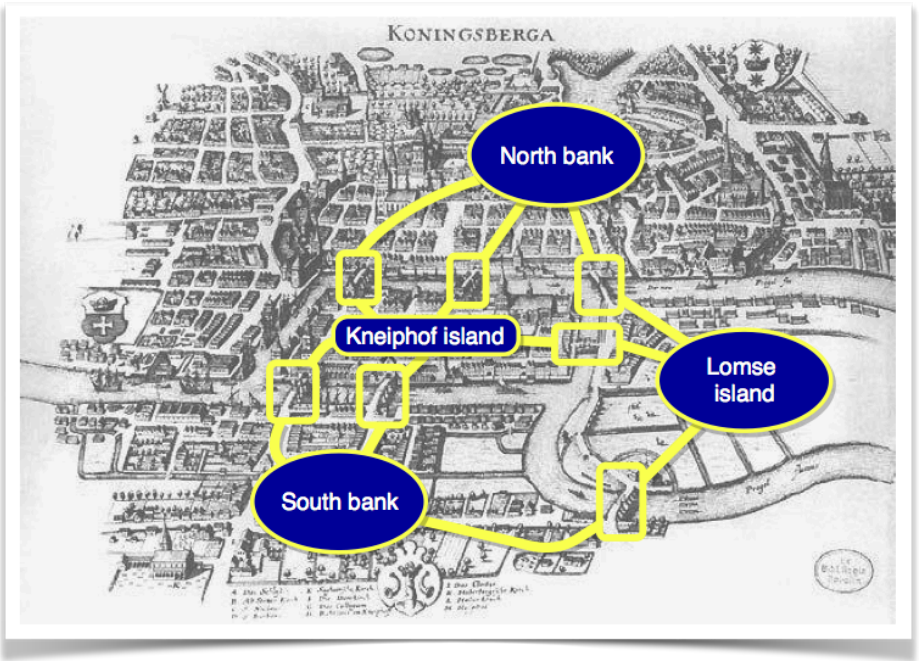
```
High Assortativity Network: 0.650
```

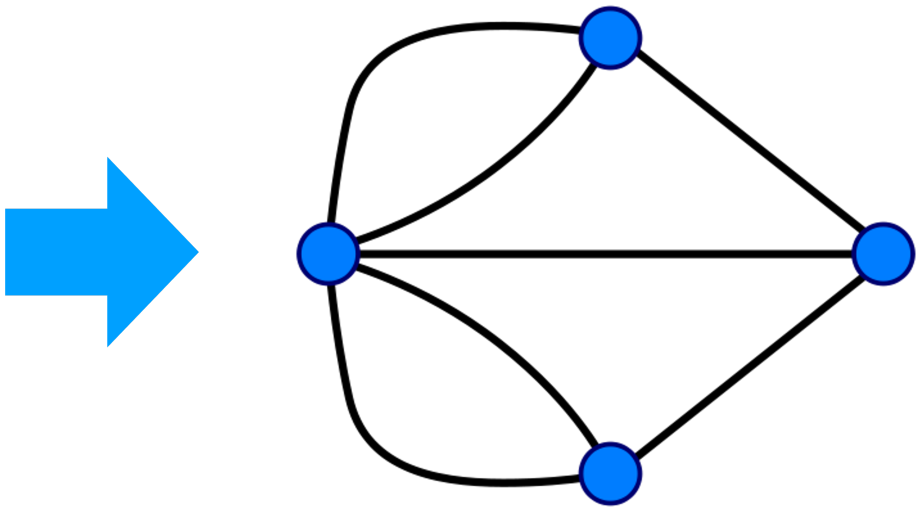
> **Note**
>
> In the 2nd lab class, I asked you to design an algorithm to generate those networks

# Paths: definitions

- **Path**: sequence of links traversed to go from a **source** to a **target** node
  - → In a directed network, links must be traversed according to their direction
  - → There may not be a path
- **Cycle**: path where source and target node are the same
- **Simple path**: no traversing the same link more than once
  - → We will only deal with simple paths
- **Path length**: number of links in path

# Euler circa 1736: Koningsberg bridges

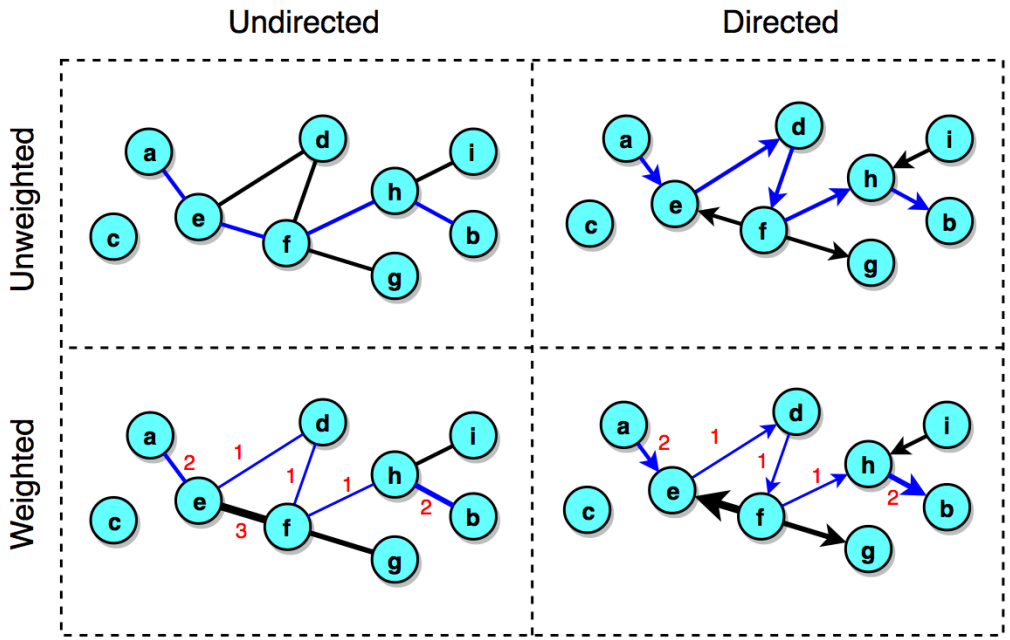


Q: Can you cross all 7 bridges just once each?

A: No. At most two nodes (start, end) may have odd degree

# Shortest paths




**Shortest path** between two nodes: minimal length (there may be more than one)

- In weighted networks, weights may represent distances

**Shortest path length** or **distance**: length of shortest path

- Undefined ($\infty$) if there is no path

# APL and diameter (1/2)

The **diameter** is the longest shortest-path length, or the maximum of the shortest path lengths across all pairs of nodes:

$$\ell_{\max} = \max_{i,j} \ell_{ij}$$

The **average path length** (APL) is the average of the shortest path lengths across all pairs of nodes

$$\langle \ell \rangle = \frac{\sum_{i,j} \ell_{ij}}{\binom{N}{2}} = \frac{2 \sum_{i,j} \ell_{ij}}{N(N-1)} \quad \textbf{for undirected networks}$$

$$\langle \ell \rangle = \frac{\sum_{i,j} \ell_{ij}}{N(N-1)} \quad \textbf{for directed networks}$$

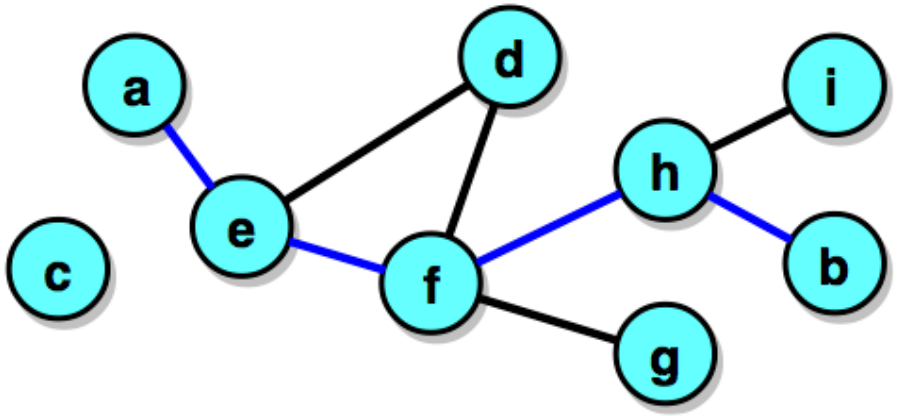
# APL and diameter (2/2)

What if there is not a path between one or more pairs of nodes?

- We can say APL and diameter are undefined (as NetworkX does)
- We can measure APL and diameter within the *largest connected component* (defined later)
- We can use a mathematical trick:

$$\langle \ell \rangle = \left( \frac{\sum_{i,j} \frac{1}{\ell_{ij}}}{\binom{N}{2}} \right)^{-1}$$

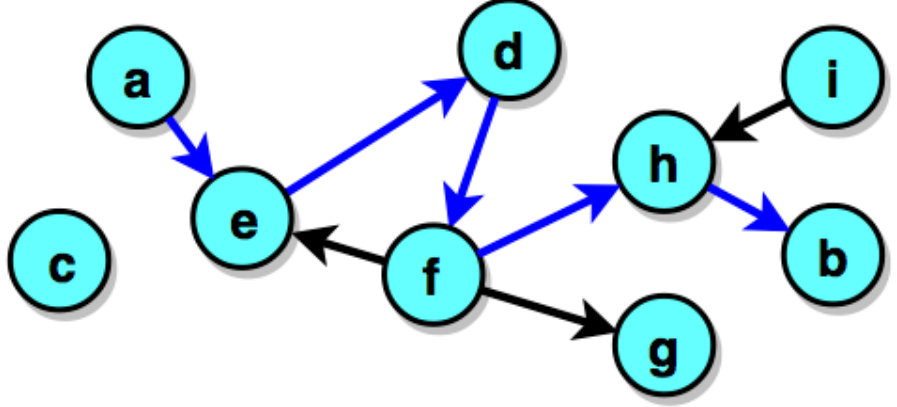
# Paths and APL (1/2)




```
1  nx.has_path(G, 'a', 'c')
2  nx.has_path(G, 'a', 'b')
3  nx.shortest_path(G, 'a', 'b')
4  nx.shortest_path_length(G,'a','b')
5  nx.shortest_path(G, 'a')            # dictionary
6  nx.shortest_path_length(G, 'a')     # dictionary
7  nx.shortest_path(G)                 # all pairs
8  nx.shortest_path_length(G)          # all pairs
9  nx.average_shortest_path_length(G) # error
10 G.remove_node('c')                  # make G connected
11 nx.average_shortest_path_length(G) # now okay
```

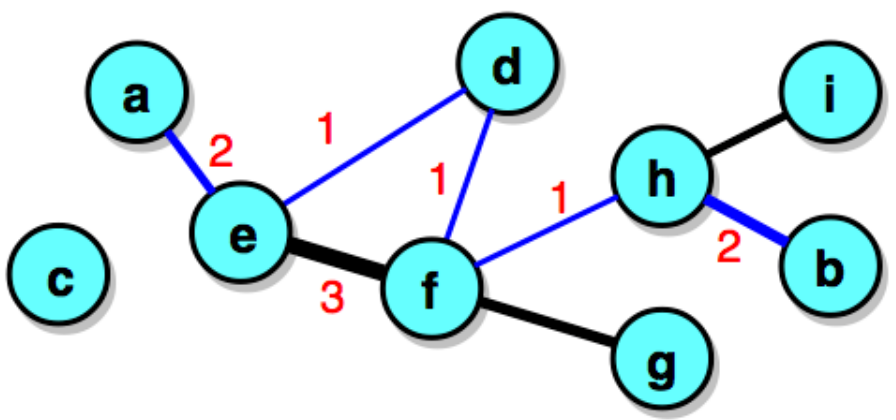
```
False
True
['a', 'e', 'f', 'h', 'b']
4
{'a': ['a'], 'e': ['a', 'e'], 'd': ['a',
'e', 'd'], 'f': ['a', 'e', 'f'], 'g':
['a', 'e', 'f', 'g'], 'h': ['a', 'e',
'f', 'h'], 'i': ['a', 'e', 'f', 'h',
'i'], 'b': ['a', 'e', 'f', 'h', 'b']}
{'a': 0, 'e': 1, 'd': 2, 'f': 2, 'g': 3,
'h': 3, 'i': 4, 'b': 4}
{'a': {'a': ['a'], 'e': ['a', 'e'], 'd':
['a', 'e', 'd'], 'f': ['a', 'e', 'f'],
'g': ['a', 'e', 'f', 'g'], 'h': ['a',
'e', 'f', 'h'], 'i': ['a', 'e', 'f', 'h',
'i'], 'b': ['a', 'e', 'f', 'h', 'b']},
'e': {'e': ['e'], 'a': ['e', 'a'], 'd':
['e', 'd'], 'f': ['e', 'f'], 'g': ['e',
'f', 'g'], 'h': ['e', 'f', 'h'], 'i':
['e', 'f', 'h', 'i'], 'b': ['e', 'f',
'h', 'b']}, 'd': {'d': ['d'], 'e': ['d',
```

# Paths and APL (2/2)




```
1 nx.has_path(D, 'b', 'a')
2 nx.has_path(D, 'a', 'b')
3 nx.shortest_path(D, 'a', 'b')
```

```
False
True
['a', 'e', 'd', 'f', 'h', 'b']
```




```
1 nx.shortest_path_length(W, 'a', 'b')
2 nx.shortest_path_length(W, 'a', 'b', 'weight')
```

```
4
7
```

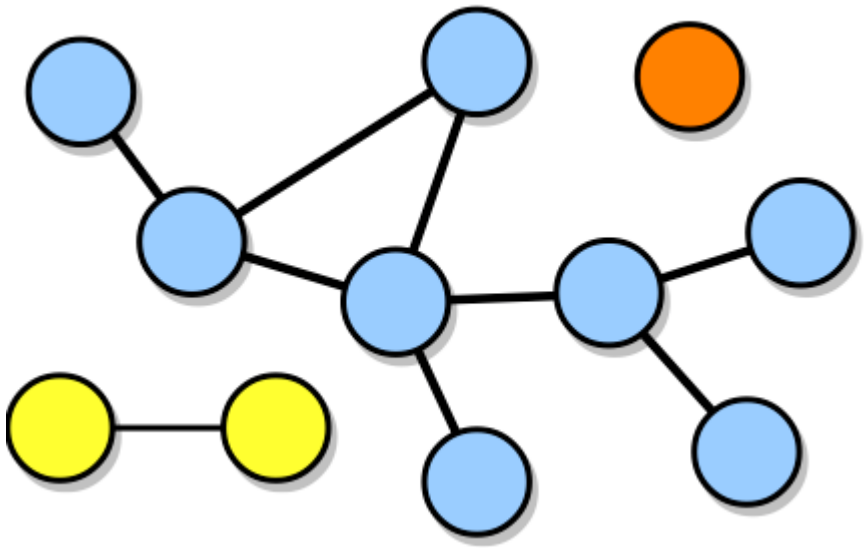
# Connectedness and components (1/3)

A network is **connected** if there is a path between any two nodes

If a network is not connected, it is **disconnected** and has multiple connected components

A **connected component** is a connected subnetwork

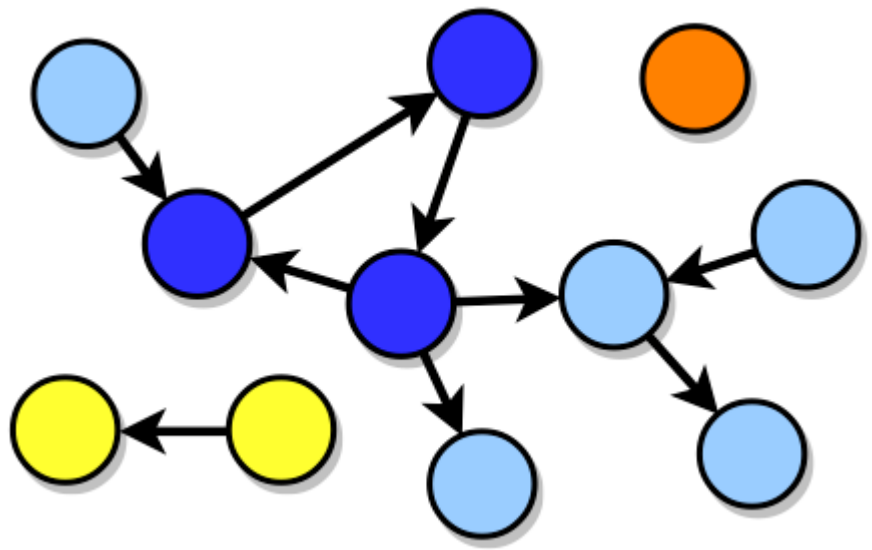
- The largest one is called **giant component** ; it often includes a substantial portion of the network
- A singleton is the smallest-possible connected component



# Connectedness and components (2/3)

A directed network can be **strongly connected** or **weakly connected** if there is a path between any two nodes, respecting or disregarding the link directions, respectively

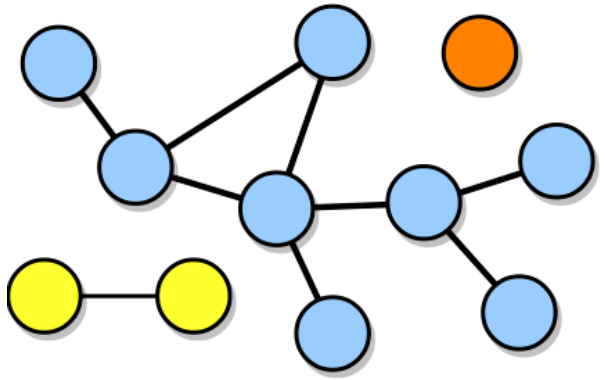
Similarly for **strongly connected** or **weakly connected** components

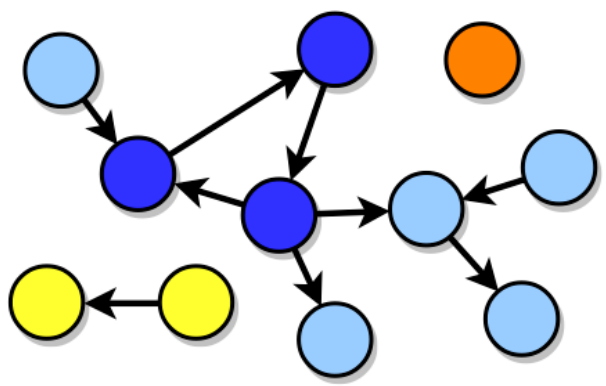


The **in-component** of a strongly connected component *S* is the set of nodes from which one can reach *S*, but that cannot be reached from *S*

The **out-component** of a strongly connected component *S* is the set of nodes that can be reached from *S*, but from which one cannot reach *S*

# Connectedness and components (3/3)





# Trees

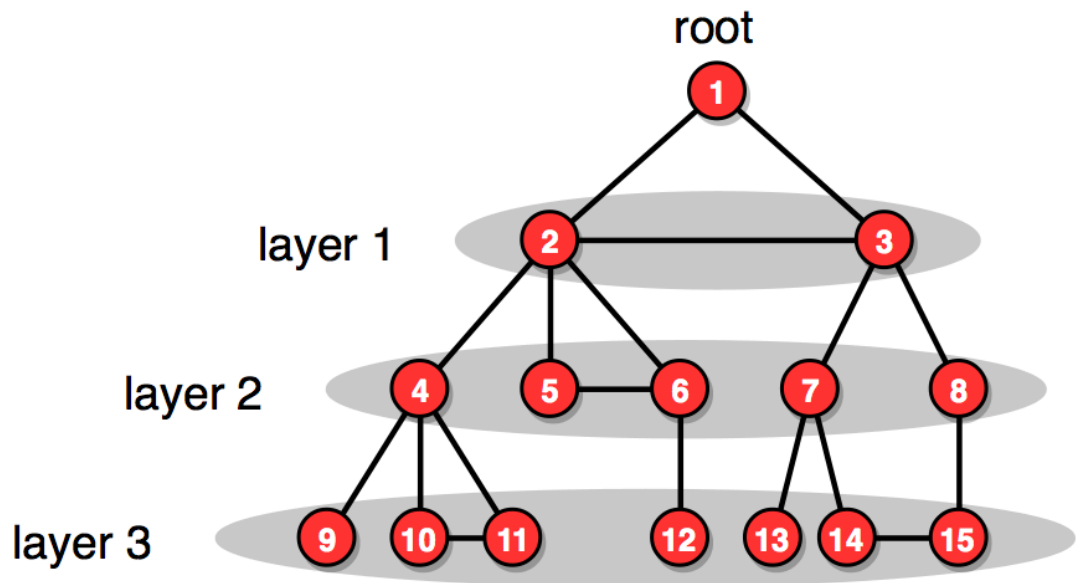
- A **tree** is a **connected** network **without cycles**
- A **tree** is a **connected** network **with** $N$ **-1 links**
- In a tree there is a single path between any two nodes
- Trees are **hierarchical**: you can pick a node as the **root** . Each node is connected to a **parent** node (toward the root) and to one or more **children** nodes (away from the root). Exceptions:
  - → The root has no parent
  - → The **leaves** have no children

# Finding shortest paths

The algorithm used to find shortest paths is called **breadth-first search**

Start from a source node (root)

Visit the entire breadth of the network, within some distance from the source, before we move to a greater depth, farther away from the source



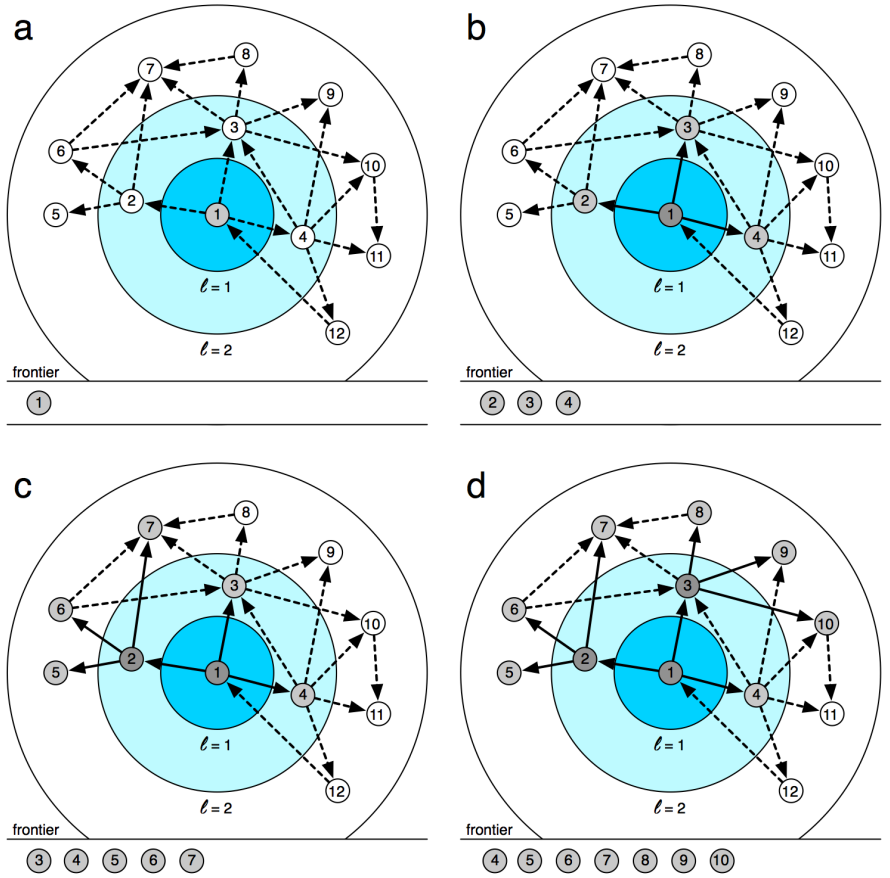

Start from each node to find all-pairs-shortest-paths (slow: $\mathcal{O}(N^2)$)

# Breadth-first search (BFS)

Each node has an attribute storing its **distance** $l$ from the source, initially $l = -1$ except $l(source) = 0$

A queue (FIFO) holds the **frontier**, initially contains the source

A directed **shortest path tree**, initially all the nodes and no links




Iterate until the frontier is empty:

- Remove next node $i$ in frontier
- For each neighbour/successor $j$ of $i$ with $l(j) = -1$:
  - → Queue $j$ into frontier
  - → $l(j) = l(i) + 1$
  - → Add link $(i \rightarrow j)$ to shortest-path tree

# Social distance

How close or far are two nodes in a network?

As we have seen, this is a question about the average path length

The question has been explored extensively in social networks

Let us start by considering coauthorship networks, in which nodes are scholars and links represent two people having coauthored one or more publications

# Paul Erdős

Image by Kmhkmh - CC BY 3.0
https://commons.wikimedia.org/w/index.php?curid=38087162

One of the world's greatest mathematicians

Considered the father of graph theory together with Alfréd Rényi

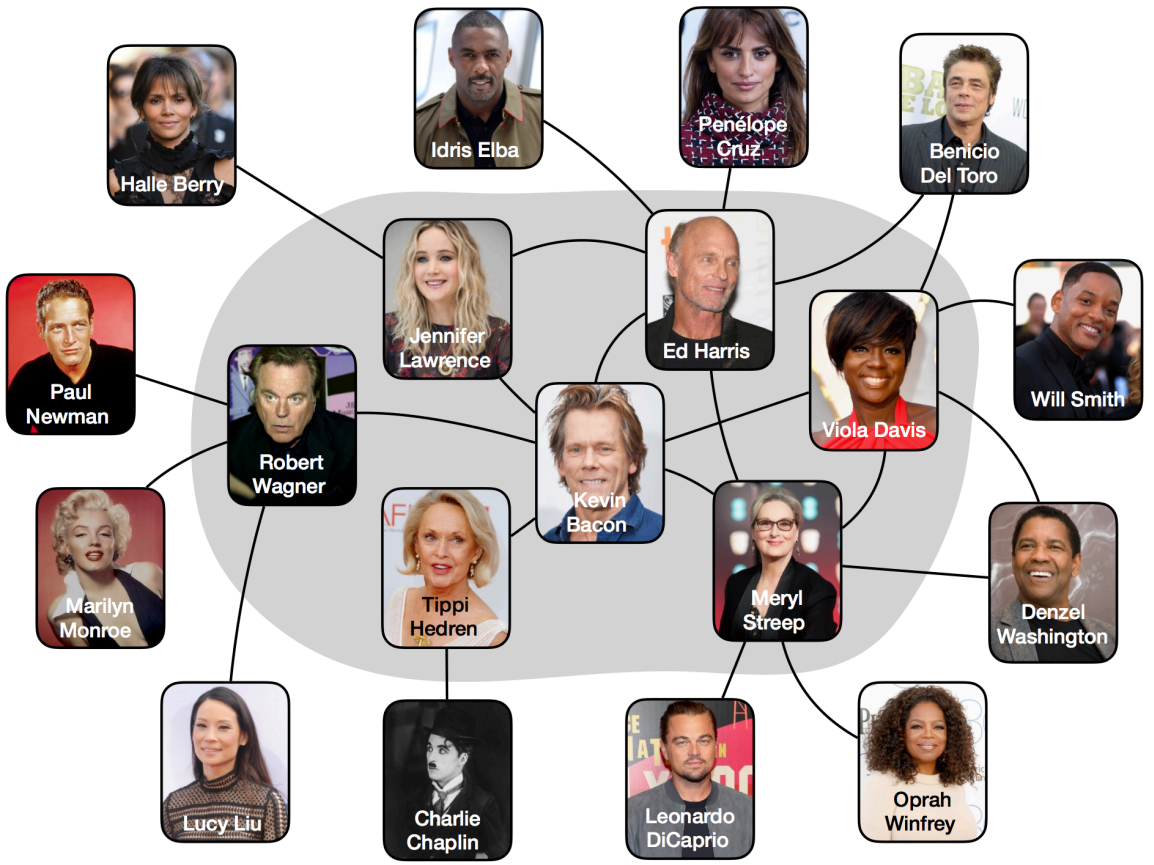
He collaborated with over 500 coauthors: a **hub** in the coauthorship network!

An author's *Erdős number* is the length of the shortest path between them and Erdős in the coauthorship network and most mathematicians are proud to have a small Erdős number.

Tool to compute one's Erdős number:
*mathscinet.ams.org/mathscinet/collaborationDistance.html*

# Six Degrees of Kevin Bacon

- Short paths are found among all authors, not just Erdős…
- … and in all social networks, not just coauthorship
- Consider the movie co-star network as a second example
- Let's play the Oracle of Bacon game: *oracleofbacon.org*
  - → Not just Kevin Bacon…
  - → Can you find two stars separated by more that four links? Play the game and try!



# Small worlds

What have we learned? Social networks tend to have very **short paths**

**Six degrees of separation**: the idea that any two people are at most six steps away from each other in the social network

First idea was in the short story “Chains” by Hungarian writer Frigyes Karinthy in 1929

Psychologist **Stanley Milgram** provided first evidence in 1967 through his famous small world experiment in which he sought to measure the **social distance** between any two people in the US

John Guare coined the term “six degrees of separation” in a 1991 play (movie, too)

# Milgram's experiment [2]

Instructions: send to personal acquaintance who is more likely to know target

160 letters to people in Omaha, NE and Wichita, KS

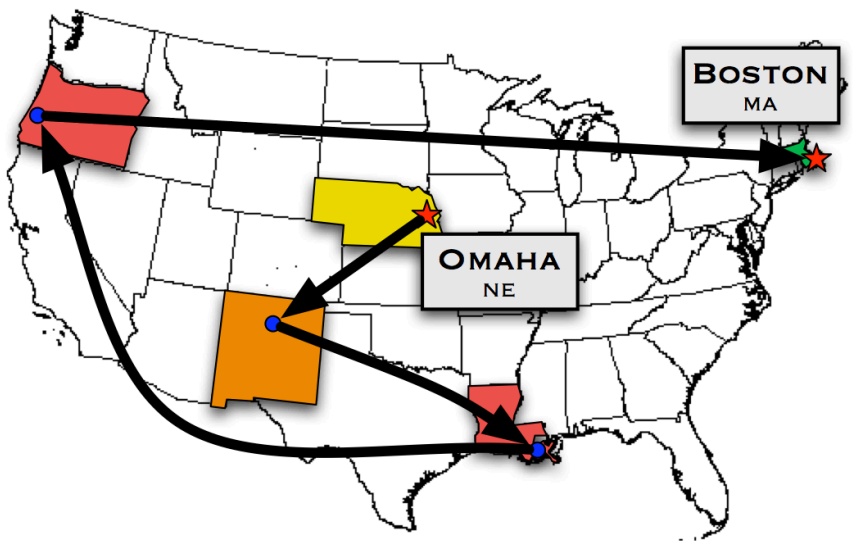
2 targets in Mass: the wife of a student in Sharon and a stockbroker in Boston

42 letters made it back (only 26%)

Average: 6.5 steps (range: 3-12 steps)

Much lower than most people expected!

"Small world" effect is still surprising

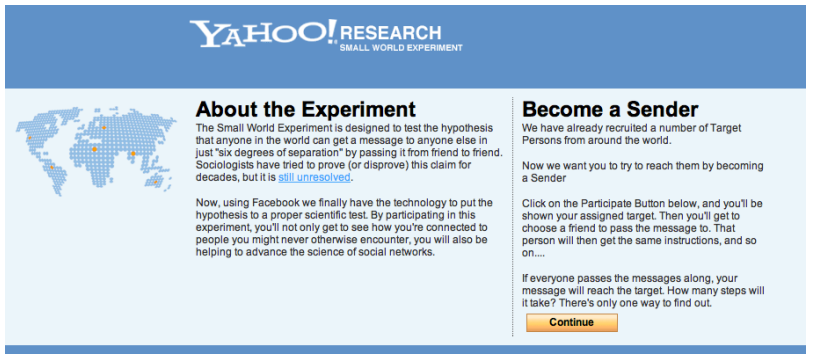



**Warning**

What is impact of the 74% missing letters?

# More small world experiments

- Yahoo Research using email in 2003
  - → 18 targets in 13 countries
  - → 384 completed chains out of more than 24 thousand started
  - → APL = 4 but when accounting for broken chains, estimated median PL of 5–7 steps




- Facebook and University of Milan in 2011
  - → 721 million active Facebook users
  - → 69 billion friendships
  - → APL = 4.74 steps: even shorter!

# Short paths

What do we mean by “short paths”? When can we call a path “short”?

It depends on the size of the network!

Observe the relationship between APL and network size when considering networks (or subnetworks) of different sizes

We say that the average path length is **short** when it **grows very slowly** with the size of the network, say, logarithmically:

$$\langle \ell \rangle \sim \log N$$

# Small worlds

- Many other types of networks are small worlds, too
- Air transportation networks, the Internet, the Web, and Wikipedia, all have short paths
  - → Play Wikiracing games to convince yourself
  - → Example: The Wiki Game (*thewikigame.com*)
- Most real-world networks are small worlds

**Table 2.1** Average path length and clustering coefficient of various network examples. The networks are the same as in Table 1.1, their numbers of nodes and links are listed as well. Link weights are ignored. The average path length is measured only on the giant component; for directed networks we consider directed paths in the giant strongly connected component. To measure the clustering coefficient in directed networks, we ignore link directions.

| Network | Nodes ($N$) | Links ($L$) | Average path length ($\langle \ell \rangle$) | Clustering coefficient ($C$) |
|---|---|---|---|---|
| Facebook Northwestern Univ. | 10,567 | 488,337 | 2.7 | 0.24 |
| IMDB movies and stars | 563,443 | 921,160 | 12.1 | 0 |
| IMDB co-stars | 252,999 | 1,015,187 | 6.8 | 0.67 |
| Twitter US politics | 18,470 | 48,365 | 5.6 | 0.03 |
| Enron Email | 87,273 | 321,918 | 3.6 | 0.12 |
| Wikipedia math | 15,220 | 194,103 | 3.9 | 0.31 |
| Internet routers | 190,914 | 607,610 | 7.0 | 0.16 |
| US air transportation | 546 | 2,781 | 3.2 | 0.49 |
| World air transportation | 3,179 | 18,617 | 4.0 | 0.49 |
| Yeast protein interactions | 1,870 | 2,277 | 6.8 | 0.07 |
| C. elegans brain | 297 | 2,345 | 4.0 | 0.29 |
| Everglades ecological food web | 69 | 916 | 2.2 | 0.55 |

**Note**

Can you think of types of network structure in which paths might not be so short?