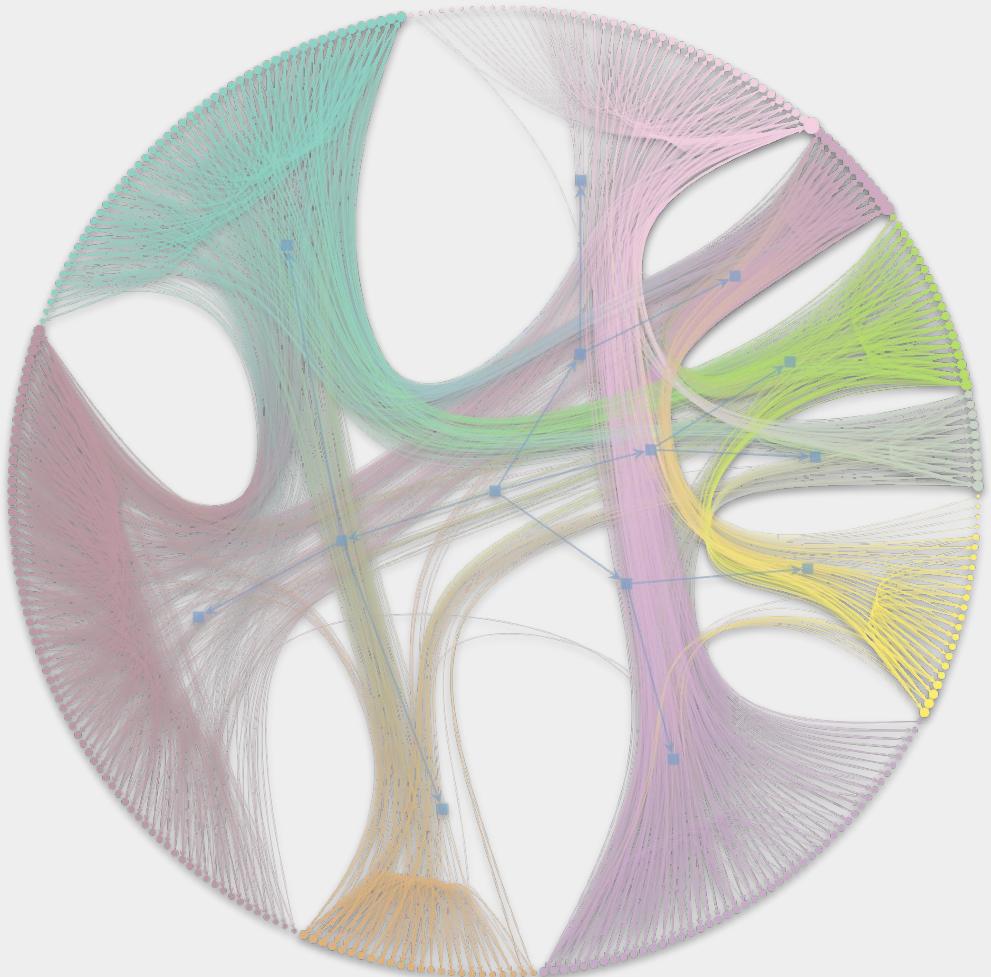


Network Elements

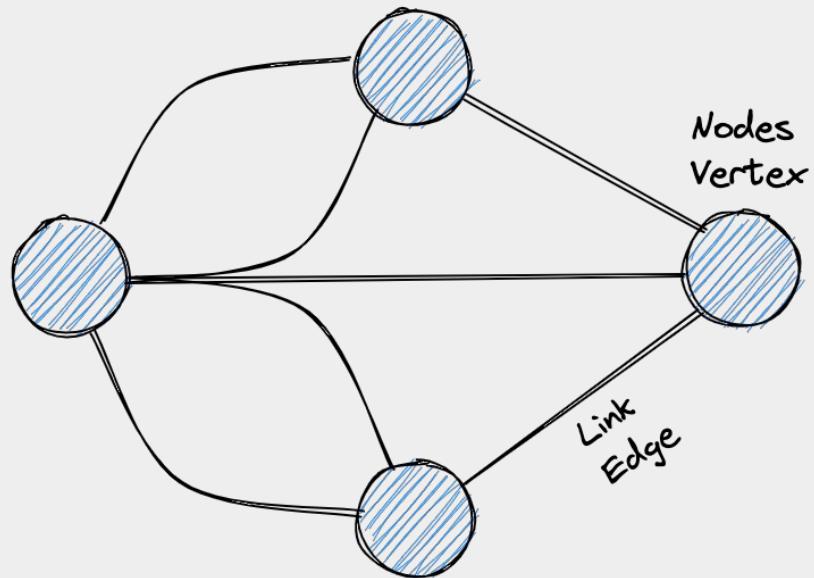
Part 02

ivanovitch.silva@ufrn.br
@ivanovitchm



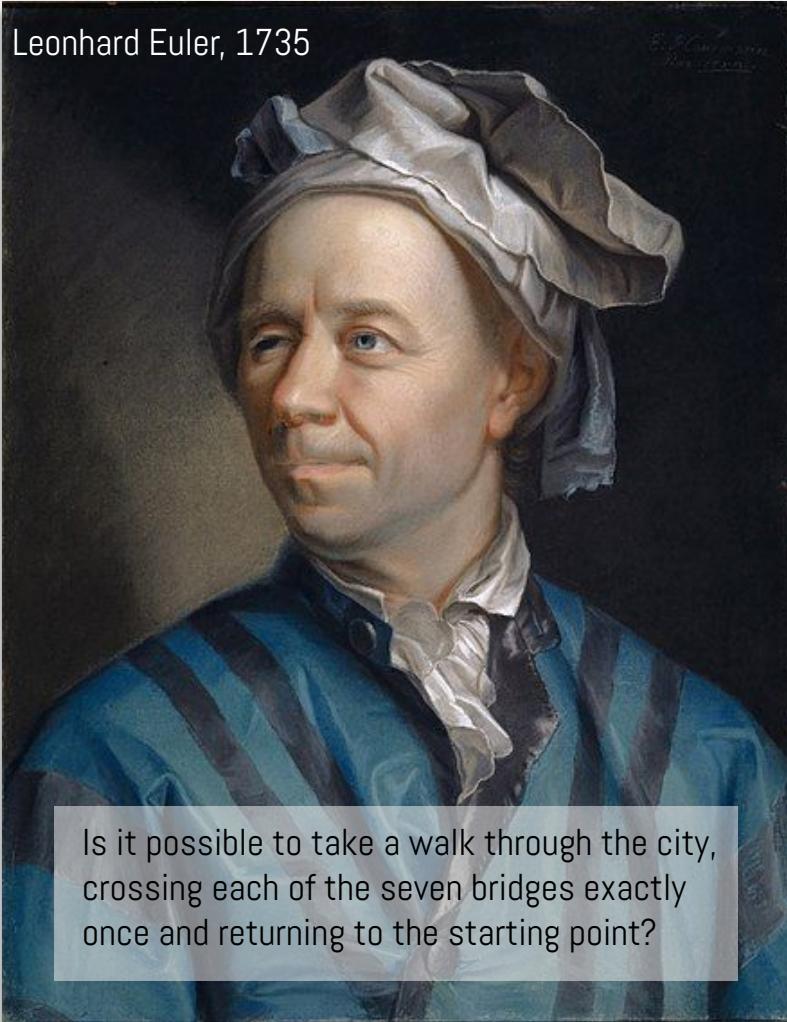
Basic Definition

Network or Graph



In very general terms a network, or graph, is a set of elements, which we call **nodes**, along with a set of connections between pairs of nodes, which we call **links**. The links represent the presence of a relation among the elements represented by the nodes.

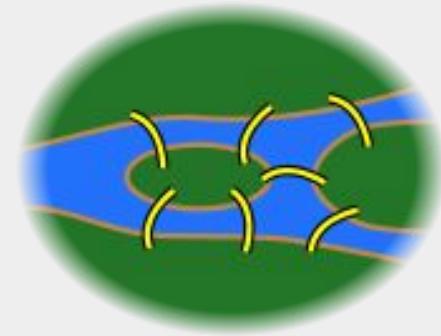
Leonhard Euler, 1735



Is it possible to take a walk through the city, crossing each of the seven bridges exactly once and returning to the starting point?

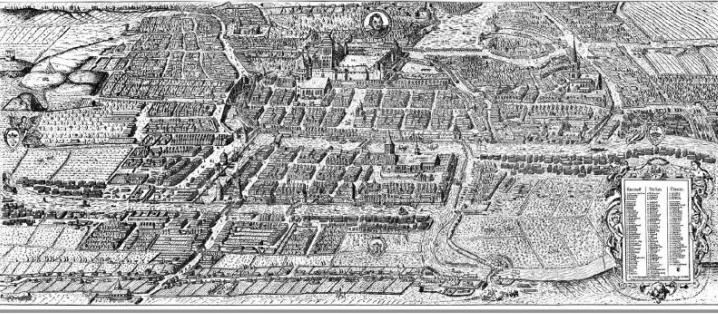
Seven Bridges of Königsberg

Can one walk across all seven bridges and never cross the same one twice?

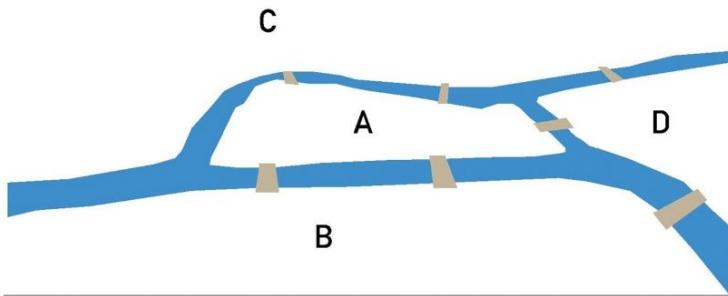


The rigorous language for the description of networks is found in graph theory, a field of mathematics that can be traced back to the pioneering work of Leonhard Euler in the eighteenth century.

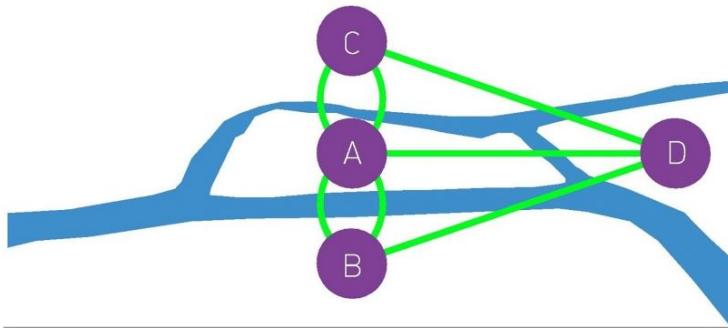
a.



b.

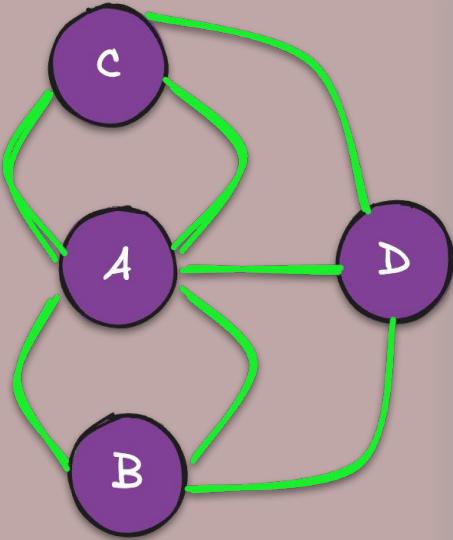


c.



Despite many attempts, no one could find such path. The problem remained unsolved until 1735, when Leonhard Euler, offered a rigorous mathematical proof that such path does not exist.





```
import networkx as nx
import matplotlib.pyplot as plt

# Create an undirected graph
G = nx.Graph()

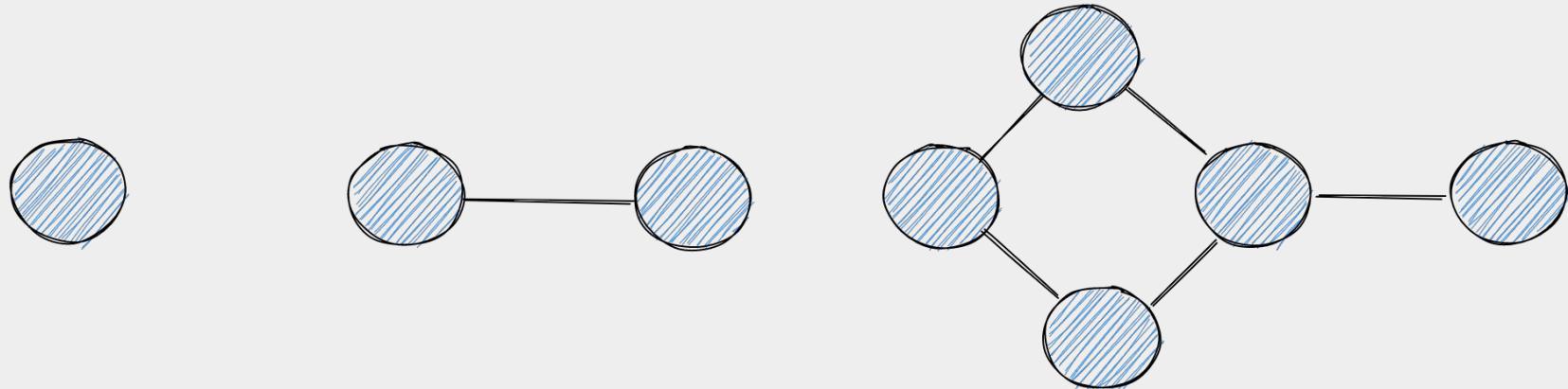
# Add nodes (land masses)
G.add_nodes_from(['A', 'B', 'C', 'D'])

# Add edges (bridges between the land masses)
edges = [
    ('A', 'B'), ('A', 'B'),          # Two bridges between A and B
    ('A', 'C'), ('A', 'C'),          # Two bridges between A and C
    ('A', 'D'),                   # One bridge between A and D
    ('B', 'D'),                   # One bridge between B and D
    ('C', 'D')                     # One bridge between C and D
]

G.add_edges_from(edges)
```

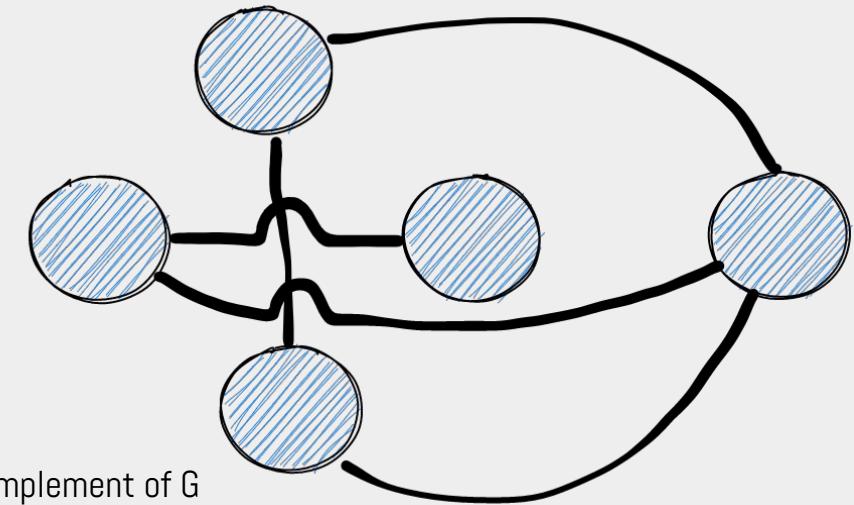
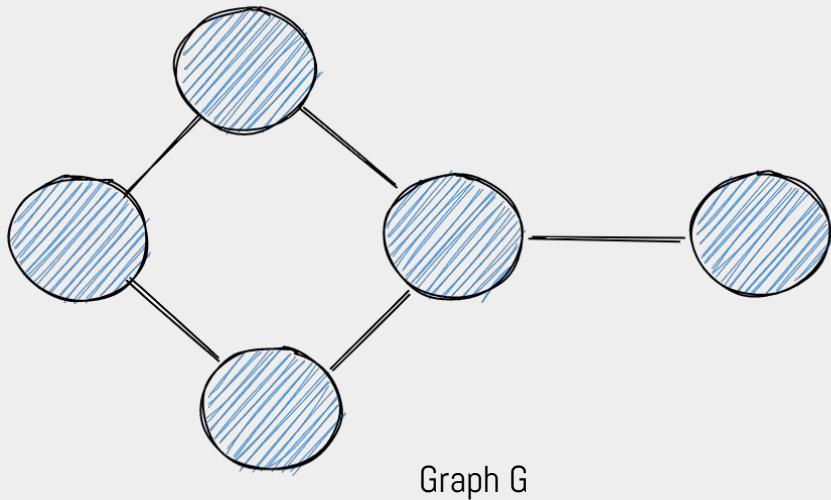
A more rigorous definition of a network

$$G = (V, E), \text{with } E \subseteq V \times V$$



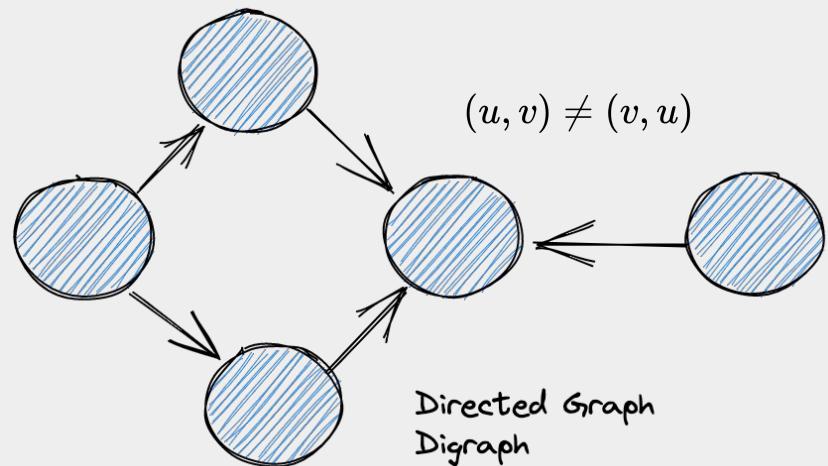
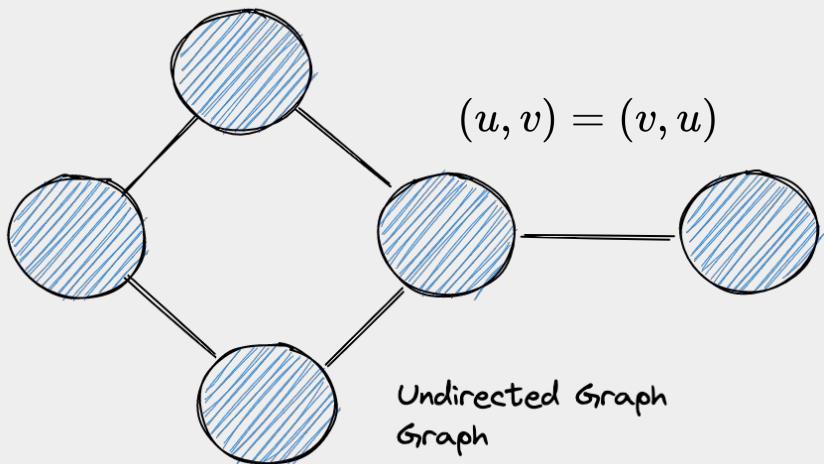
We can derive a series of special graphs

For instance, we can derive the complement of G . This is equivalent to remove all of the original edges of G , and then connect all the unconnected pairs of nodes in G .



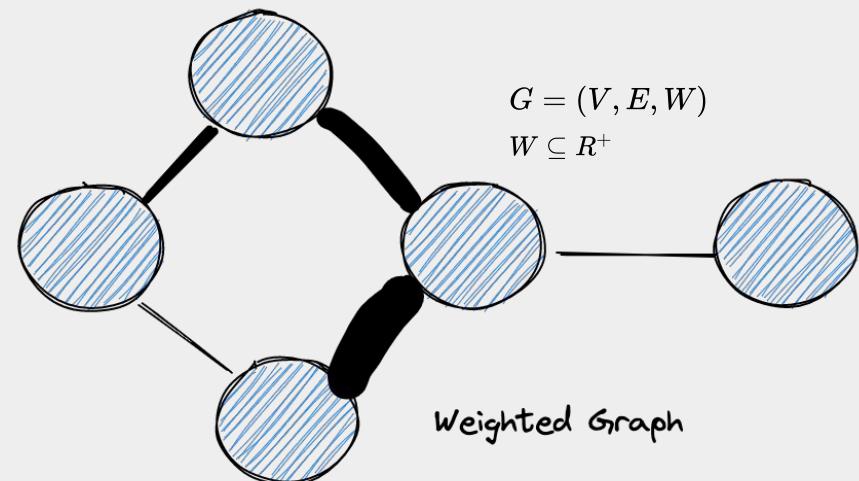
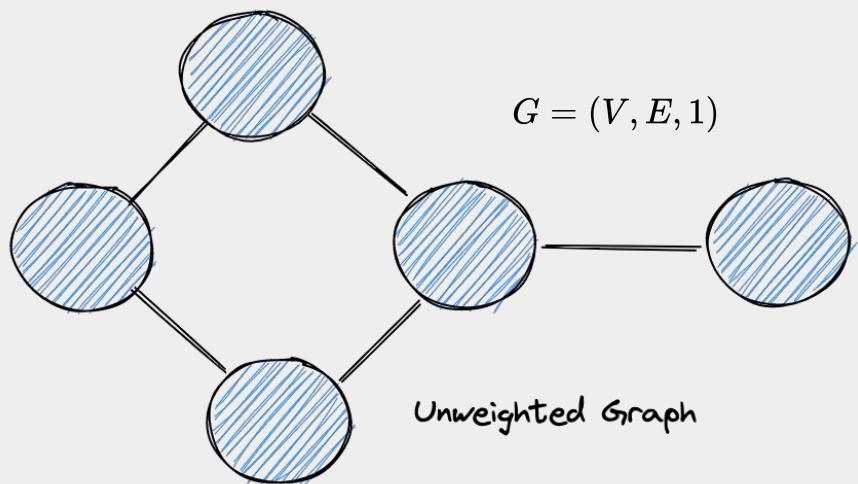
Sometimes you really need to complicate stuff

A first thing we will do is realizing that **not all relations are reciprocal**. We can introduce this asymmetry in the graph model.

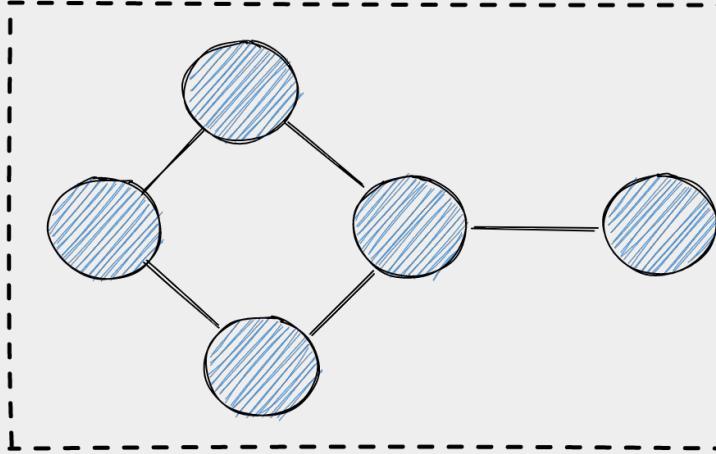


Sometimes you really need to complicate stuff

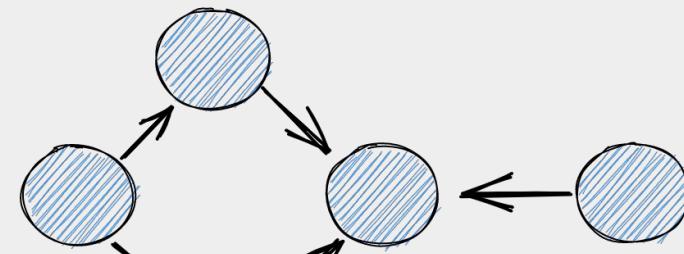
Another way to make edges more interesting is realizing that **two connections are not necessarily equally important** in the network.



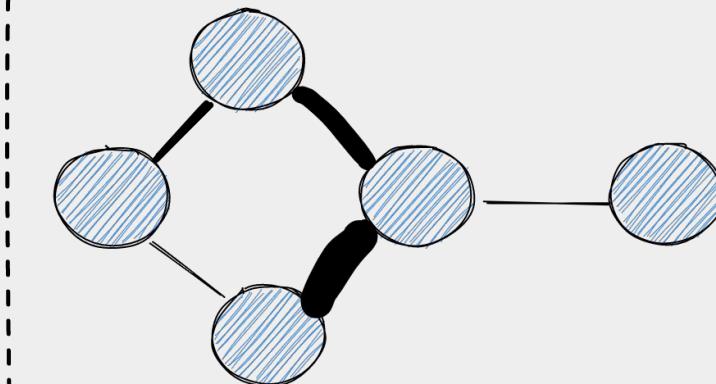
Undirected



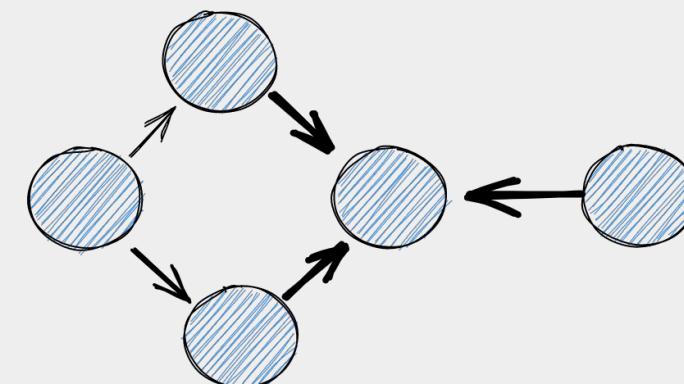
Directed

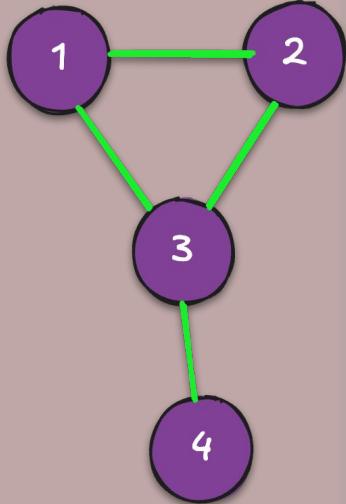


Unweighted



Weighted





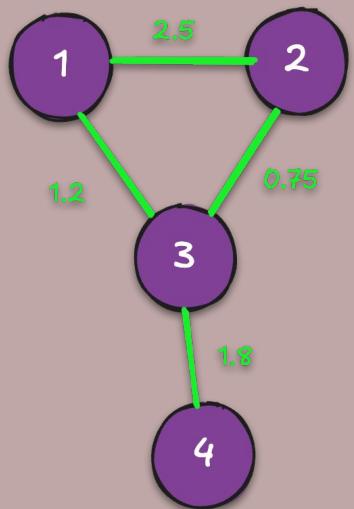
```
import networkx as nx
import matplotlib.pyplot as plt

# Create an undirected graph
G = nx.Graph()

# Add nodes
G.add_nodes_from([1, 2, 3, 4])

# Add edges (unweighted)
edges = [(1, 2), (1, 3), (2, 3), (3, 4)]
G.add_edges_from(edges)

# Draw the graph
nx.draw(G, with_labels=True, node_color='lightblue', node_size=500)
plt.title("Undirected and Unweighted Graph")
plt.show()
```



```
import networkx as nx
import matplotlib.pyplot as plt

# Create an undirected graph
G = nx.Graph()

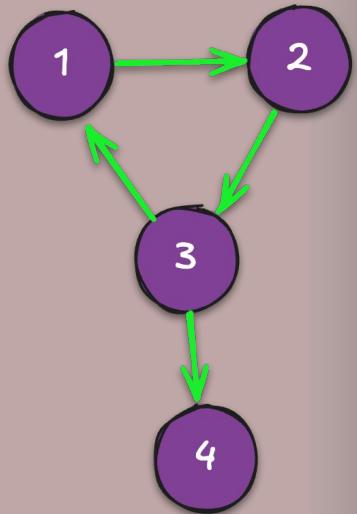
# Add nodes
G.add_nodes_from([1, 2, 3, 4])

# Add weighted edges
edges = [
    (1, 2, {'weight': 2.5}),
    (1, 3, {'weight': 1.2}),
    (2, 3, {'weight': 0.75}),
    (3, 4, {'weight': 1.8})
]
G.add_edges_from(edges)

# Get edge weights
edge_labels = nx.get_edge_attributes(G, 'weight')

# Draw the graph
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='lightgreen', node_size=500)
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.title("Undirected and Weighted Graph")
plt.show()
```

{(1, 2): 2.5, (1, 3): 1.2, (2, 3): 0.75, (3, 4): 1.8}



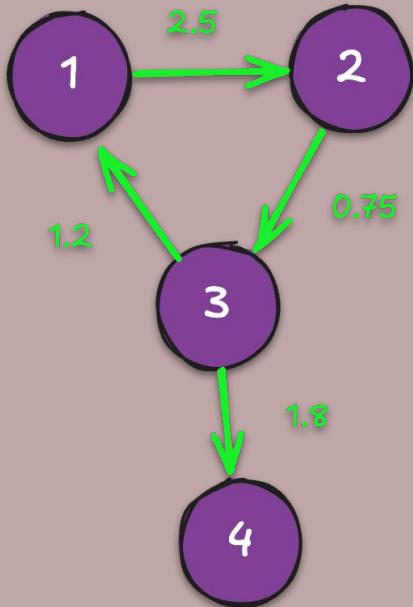
```
import networkx as nx
import matplotlib.pyplot as plt

# Create a directed graph
G = nx.DiGraph()

# Add nodes
G.add_nodes_from(['1', '2', '3', '4'])

# Add edges (unweighted)
edges = [('1', '2'), ('2', '3'), ('3', '1'), ('3', '4')]
G.add_edges_from(edges)

# Draw the graph
pos = nx.circular_layout(G)
nx.draw(G, pos, with_labels=True, node_color='orange', node_size=500, arrowsize=20)
plt.title("Directed and Unweighted Graph")
plt.show()
```



```
import networkx as nx
import matplotlib.pyplot as plt

# Create a directed graph
G = nx.DiGraph()

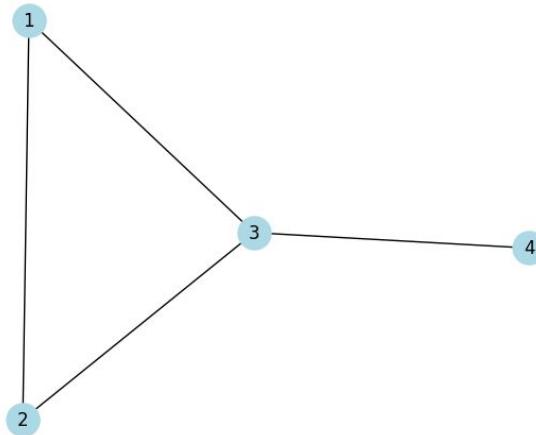
# Add nodes
G.add_nodes_from([1, 2, 3, 4])

# Add weighted edges
edges = [
    (1, 2, {'weight': 2.5}),
    (3, 1, {'weight': 1.2}),
    (2, 3, {'weight': 0.75}),
    (3, 4, {'weight': 1.8})
]
G.add_edges_from(edges)

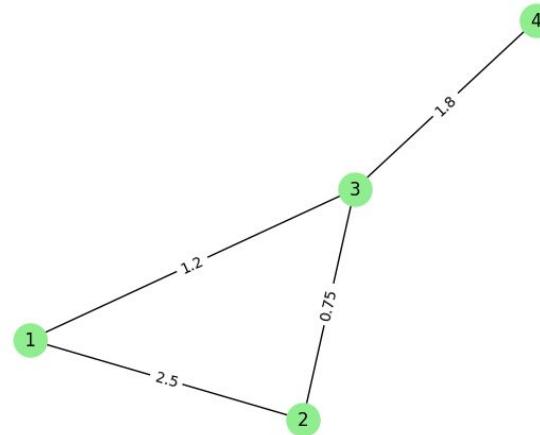
# Get edge weights
edge_labels = nx.get_edge_attributes(G, 'weight')

# Draw the graph
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='pink', node_size=500, arrowsize=20)
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
plt.title("Directed and Weighted Graph")
plt.show()
```

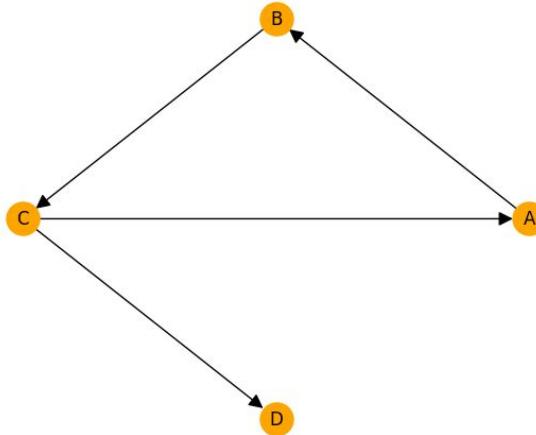
Undirected and Unweighted



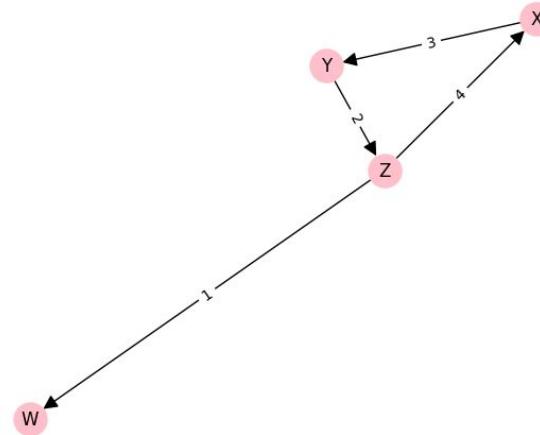
Undirected and Weighted



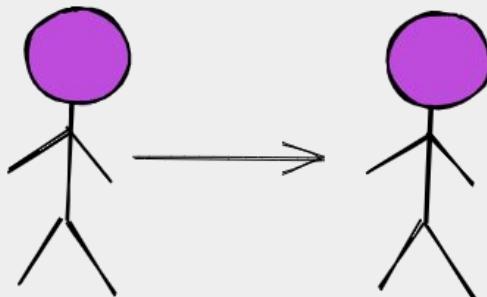
Directed and Unweighted



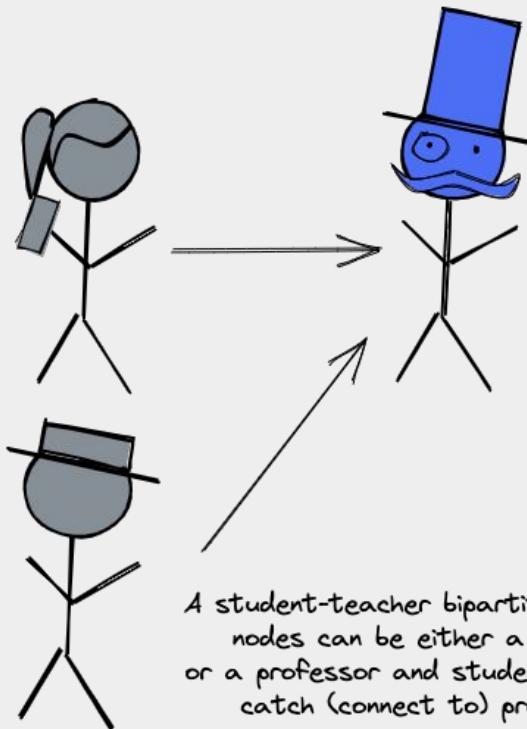
Directed and Weighted



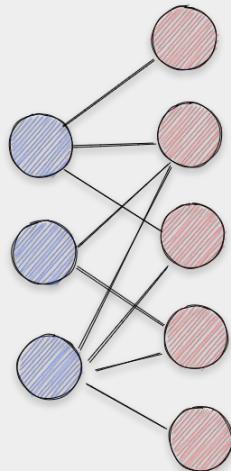
Extended Graphs (bipartite network)



A simple graph representing
a social network with
no additional constraints.



A student-teacher bipartite network:
nodes can be either a student
or a professor and students can only
connect to professor.



PESQUISAR DADOS

Ex.: cursos



Etiquetas mais comuns [requisição](#) [requisições](#) [servidores](#)

GRUPOS



Biblioteca



Comunicados e Documentos



Contratos e Convênios



Despesas e Orçamento



Ensino



Extensão

Students vs Courses
Users vs books
Teacher vs Department



Institucional



Materiais



Patrimônio



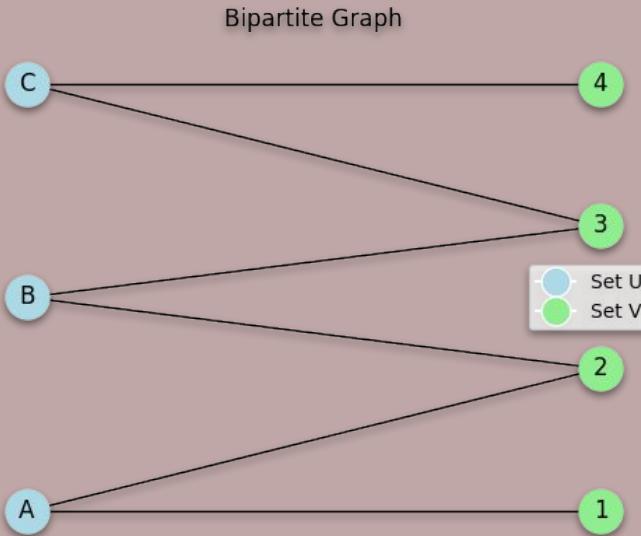
Pesquisa



Pessoas



Processos



```

import networkx as nx
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D # Import Line2D for custom legend handles

# Step 1: Create an empty graph
G = nx.Graph()

# Step 2: Define the two node sets
U = {'A', 'B', 'C'}
V = {1, 2, 3, 4}

# Step 3: Add nodes with the bipartite attribute
G.add_nodes_from(U, bipartite=0)
G.add_nodes_from(V, bipartite=1)

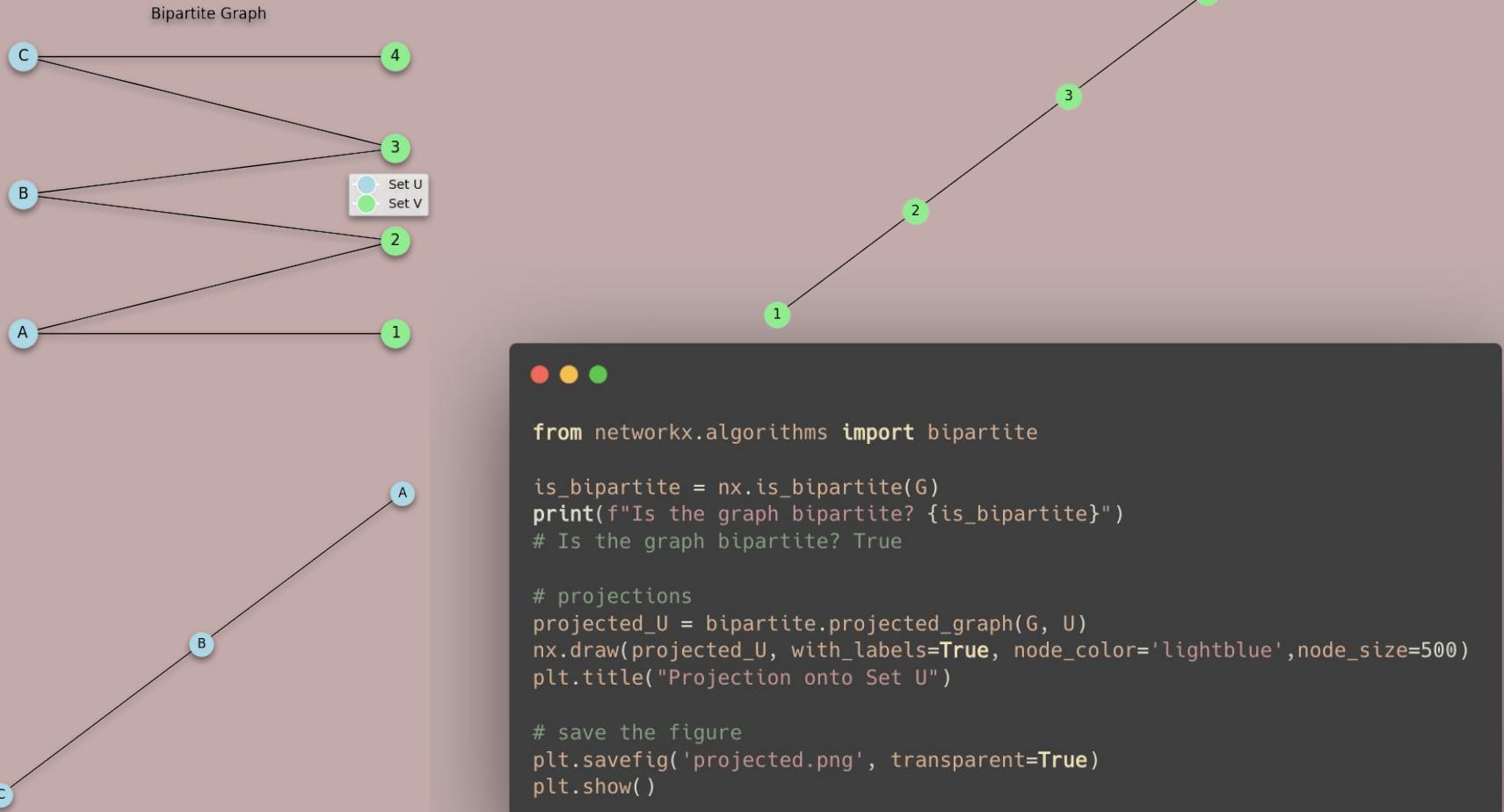
# Step 4: Add edges between nodes from U to V
edges = [('A', 1), ('A', 2), ('B', 2), ('B', 3), ('C', 3), ('C', 4)]
G.add_edges_from(edges)

# Step 5: Visualize the bipartite graph
pos = nx.bipartite_layout(G, U)

# Draw nodes for each set
nx.draw_networkx_nodes(G, pos, nodelist=U, node_color='lightblue', node_size=500)
nx.draw_networkx_nodes(G, pos, nodelist=V, node_color='lightgreen', node_size=500)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos)

# Create custom legend handles
legend_elements = [
    Line2D([0], [0], marker='o', color='w', label='Set U',
           markerfacecolor='lightblue', markersize=15),
    Line2D([0], [0], marker='o', color='w', label='Set V',
           markerfacecolor='lightgreen', markersize=15)
]

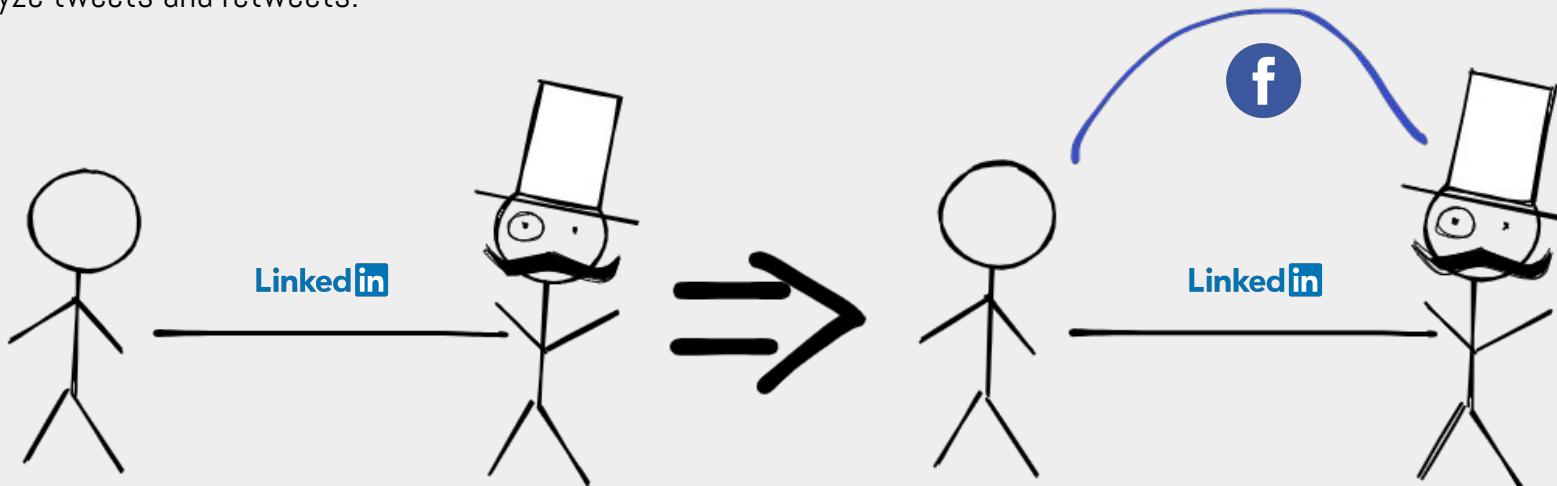
# Add the legend to the plot
plt.legend(handles=legend_elements)
plt.title("Bipartite Graph")
plt.axis('off')
# save the figure
plt.savefig('grafo_bipartite.png', transparent=True)
plt.show()
  
```



Extended Graphs (multilayer graph)

Traditionally, network scientists try to focus on one thing at a time. For instance, they will download a sample of the LinkedIn graph. Or they will analyze tweets and retweets.

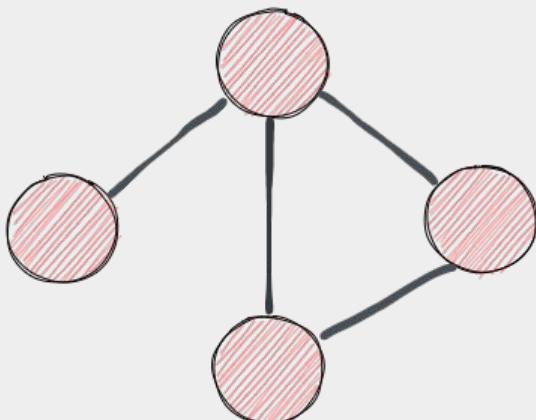
Two people might have started working in the same company and thus first connected on LinkedIn, and then became friends and connected on Facebook. Such scenario could not be captured by simply looking at one of the two networks.



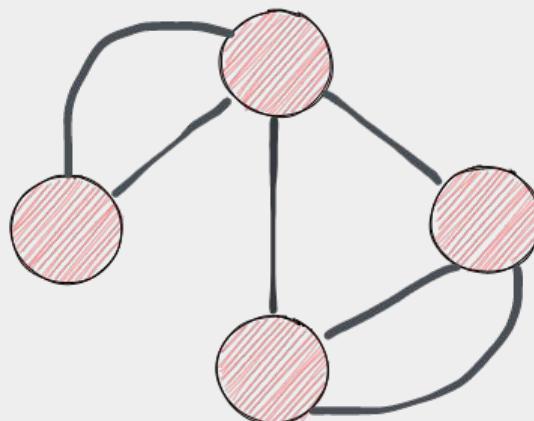
Extended Graphs (multilayer graph)

$$G = (V, E, L)$$

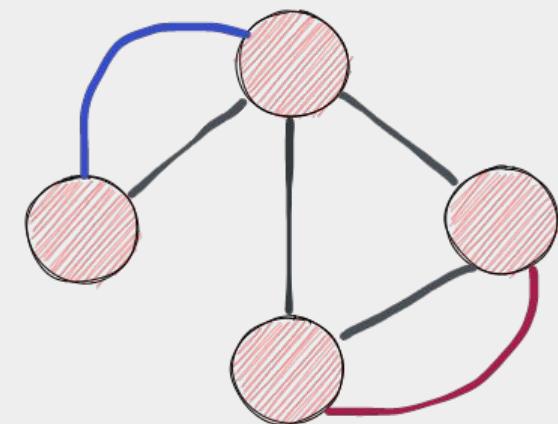
$(u, v, l) \in E$, with $u, v \in V$ and $l \in L$



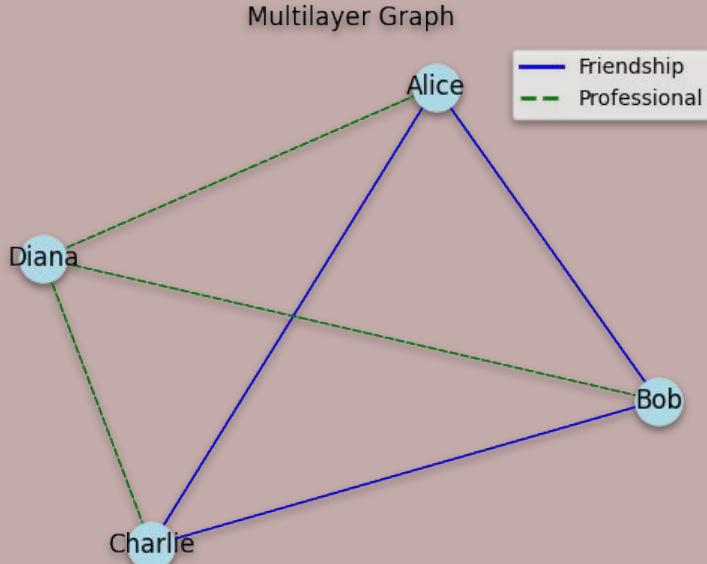
A simple graph



A multigraph, with multiple edges between the same node pairs



A multilayer network, where each edge has a type (represented by its color)



```

import networkx as nx
import matplotlib.pyplot as plt

# Create a MultiGraph to handle multiple edges
G = nx.MultiGraph()

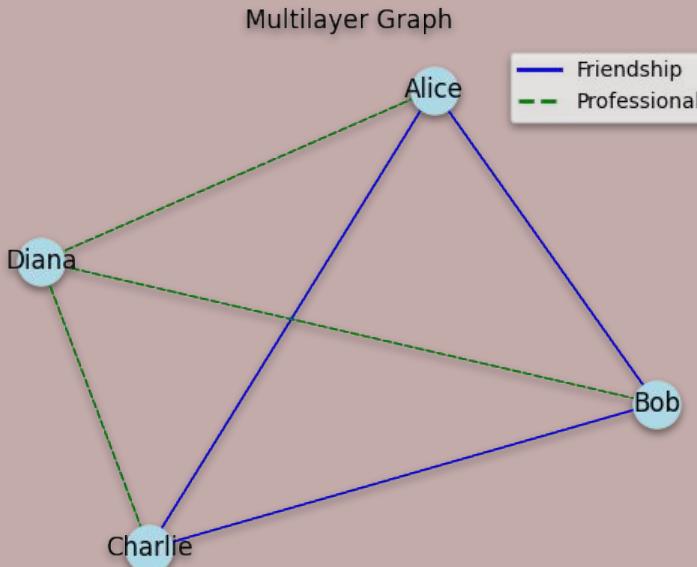
# List of nodes
nodes = ['Alice', 'Bob', 'Charlie', 'Diana']

# Add nodes to the graph
G.add_nodes_from(nodes)

# Edges in Layer 1 (Friendships)
friendships = [
    ('Alice', 'Bob'),
    ('Alice', 'Charlie'),
    ('Bob', 'Charlie')
]

# Edges in Layer 2 (Professional Relationships)
professional_relationships = [
    ('Alice', 'Diana'),
    ('Bob', 'Diana'),
    ('Charlie', 'Diana')
]

```

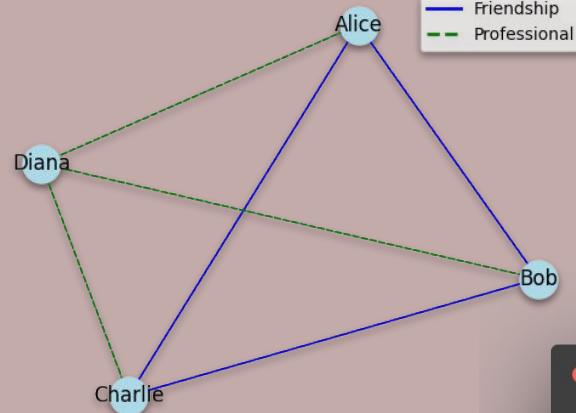


```
# Add edges with layer attribute
G.add_edges_from(friendships, layer='friendship')
G.add_edges_from(professional_relationships, layer='professional')

# Adding edges with more attributes
G.add_edge('Alice', 'Bob', layer='friendship', weight=1)
G.add_edge('Alice', 'Charlie', layer='friendship', weight=1)
G.add_edge('Bob', 'Charlie', layer='friendship', weight=1)

G.add_edge('Alice', 'Diana', layer='professional', weight=2)
G.add_edge('Bob', 'Diana', layer='professional', weight=2)
G.add_edge('Charlie', 'Diana', layer='professional', weight=2)
```

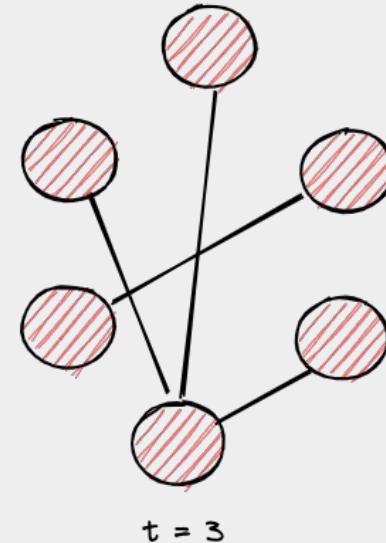
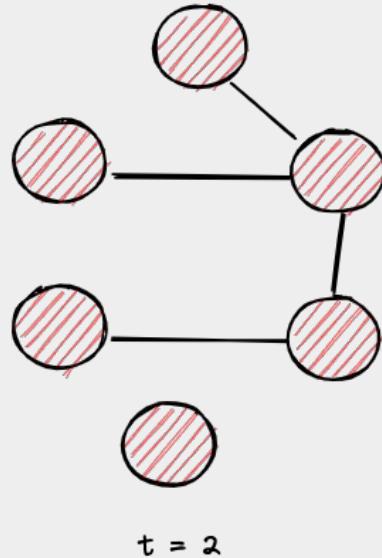
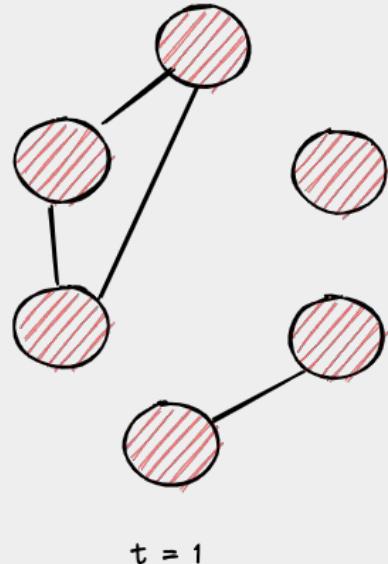
Multilayer Graph



```
● ● ●  
# Positions for nodes  
pos = nx.spring_layout(G, seed=42) # Fixed seed for reproducibility  
  
# Separate edges by layer  
edges_friendship = [(u, v) for u, v, d in G.edges(data=True) if d['layer'] == 'friendship']  
edges_professional = [(u, v) for u, v, d in G.edges(data=True) if d['layer'] == 'professional']  
  
# Draw nodes  
nx.draw_networkx_nodes(G, pos, node_size=500, node_color='lightblue')  
  
# Draw edges for friendship layer  
nx.draw_networkx_edges(G, pos, edgelist=edges_friendship, edge_color='blue', label='Friendship')  
  
# Draw edges for professional layer  
nx.draw_networkx_edges(G, pos, edgelist=edges_professional, edge_color='green', style='dashed',  
label='Professional')  
  
# Draw labels  
nx.draw_networkx_labels(G, pos)
```

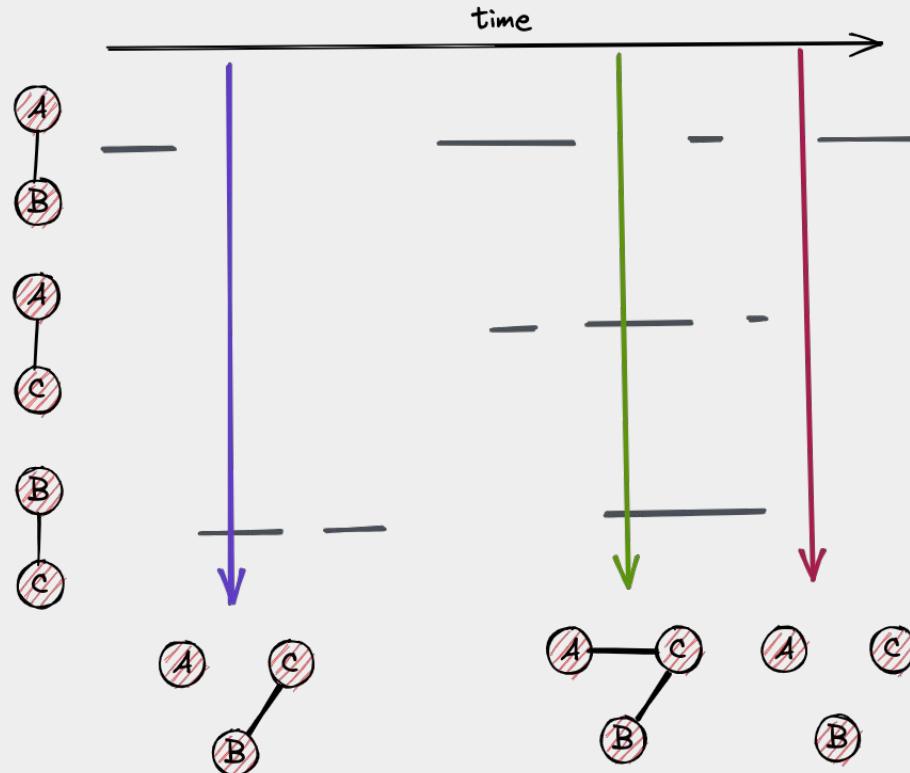
Extended Graphs (dynamic graph)

Imagine that your network represents a road graph. Nodes are intersections, and edges are stretches of the street connecting them. Roadworks might cut off a segment for a few days. If your network model cannot take this into account, you would end up telling drivers to use a road that is blocked, creating traffic jams and a lot of discomfort



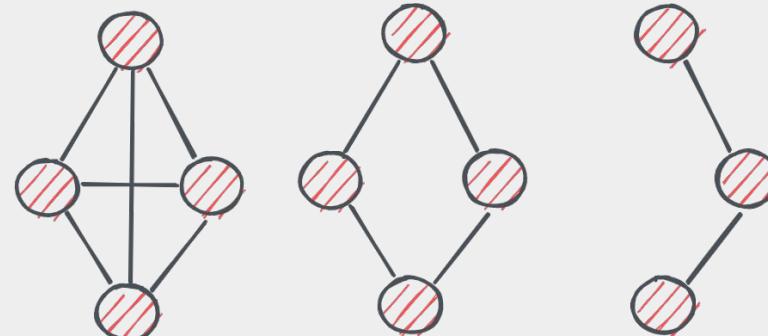
Extended Graphs (dynamic graph)

$$G = (G_1, G_2, \dots, G_n)$$
$$G_i = (V_i, E_i)$$



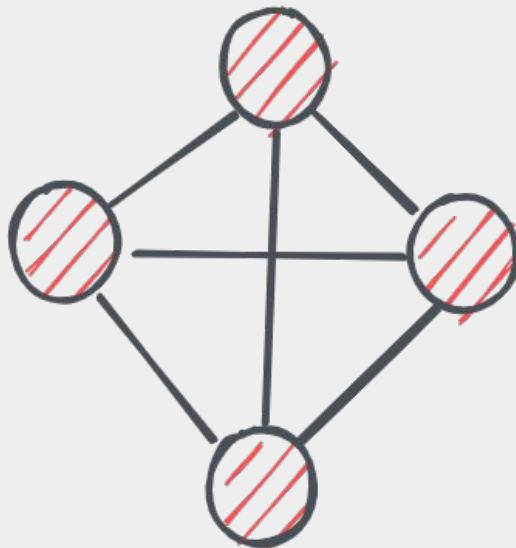


Density and Sparsity, Subnetworks



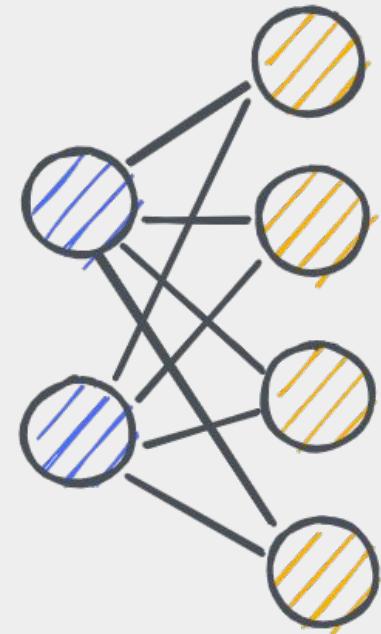
Density and Sparsity

The maximum number of links in a network is bounded by the possible number of distinct connections among the nodes of the system. The maximum number of links is therefore given by the number of pairs of nodes. A network with the maximum number of links, in which all possible pairs of nodes are connected by links, is called a **complete network**.



$$L_{max} = \binom{N}{2} = \frac{N(N - 1)}{2}$$

$$L_{max}^{directed} = N(N - 1)$$



$$L_{max} = N_1 \times N_2$$

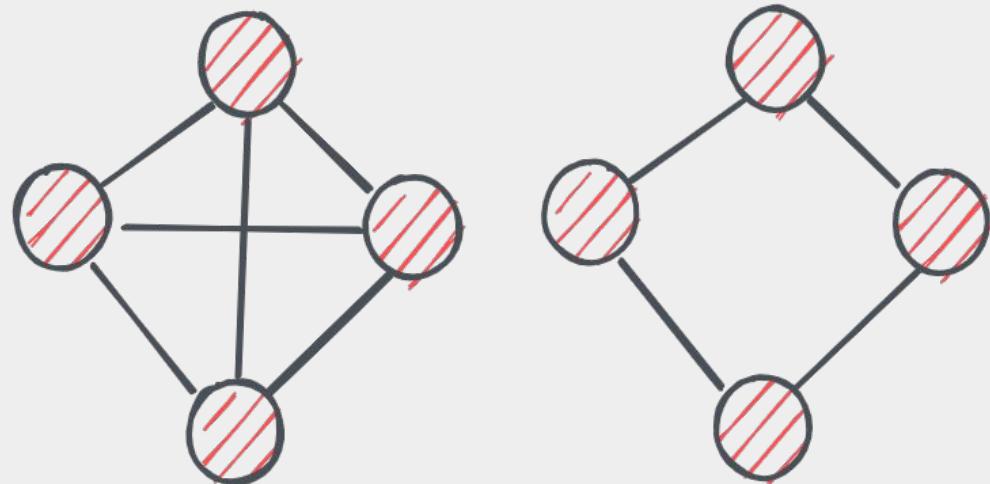
Density and Sparsity

$$d = \frac{L}{L_{max}} = \frac{2L}{N(N - 1)}$$

$$d = \frac{L}{L_{max}^{directed}} = \frac{L}{N(N - 1)}$$

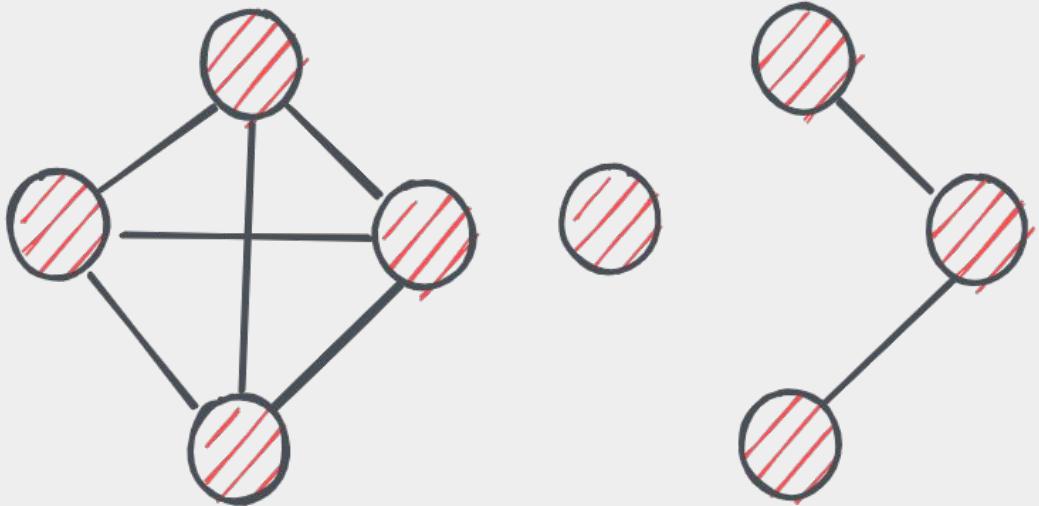
The fraction of possible links that actually exist, which is the same as the fraction of pairs of nodes that are actually connected, is called the **density of the network**.

$$d = \frac{L}{L_{max}}$$



Density and Sparsity

We say that the **network is sparse** if the number of links grows proportionally to the number of nodes ($L \sim N$), or even slower. If instead the number of links grows faster, e.g. quadratically with network size ($L \sim N^2$), then we say that the **network is dense**.

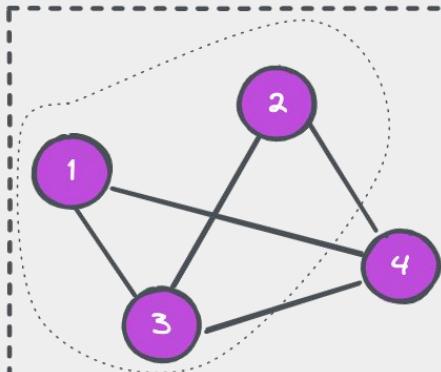


$$L \ll L_{max} \rightarrow d \ll 1$$

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
<i>C. elegans</i> brain	DW	297	2,345	0.03	7.9
Everglades ecological food web	DW	69	916	0.2	13.3

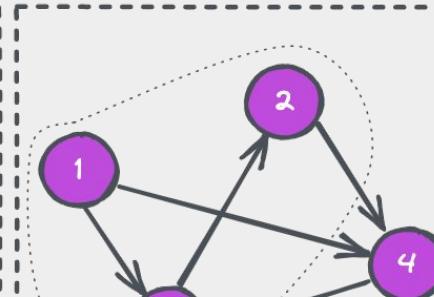
Subnetwork

Undirected



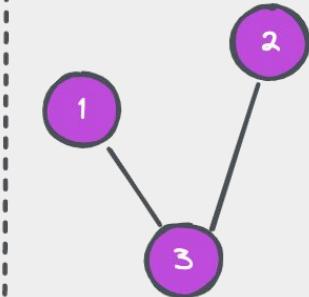
	1	2	3	4
1	0	0	1	1
2	0	0	1	1
3	1	1	0	1
4	1	1	1	0

Directed



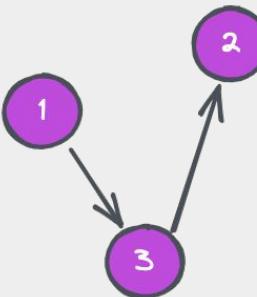
	1	2	3	4
1	0	0	1	1
2	0	0	0	1
3	0	1	0	0
4	0	0	1	0

Original



	1	2	3
1	0	0	1
2	0	0	1
3	1	1	0

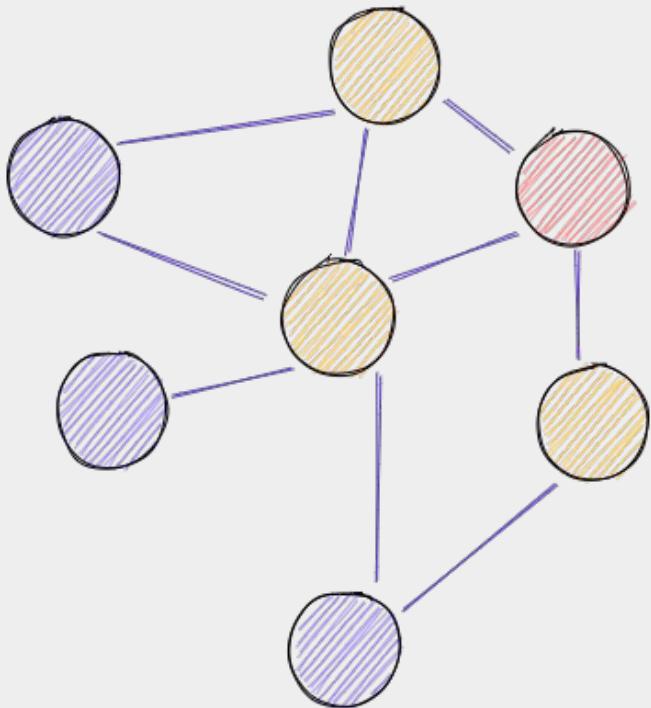
Subnetwork



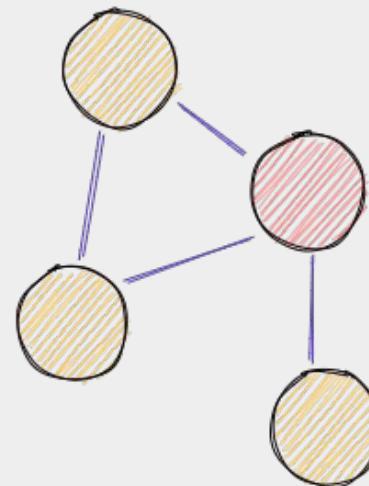
	1	2	3
1	0	0	1
2	0	0	0
3	0	1	0

Subnetwork

A special type of subnetwork is the ego network of a node, which is the subnetwork consisting of the chosen node — called the ego — and its neighbors. Ego networks are often studied in social network analysis.

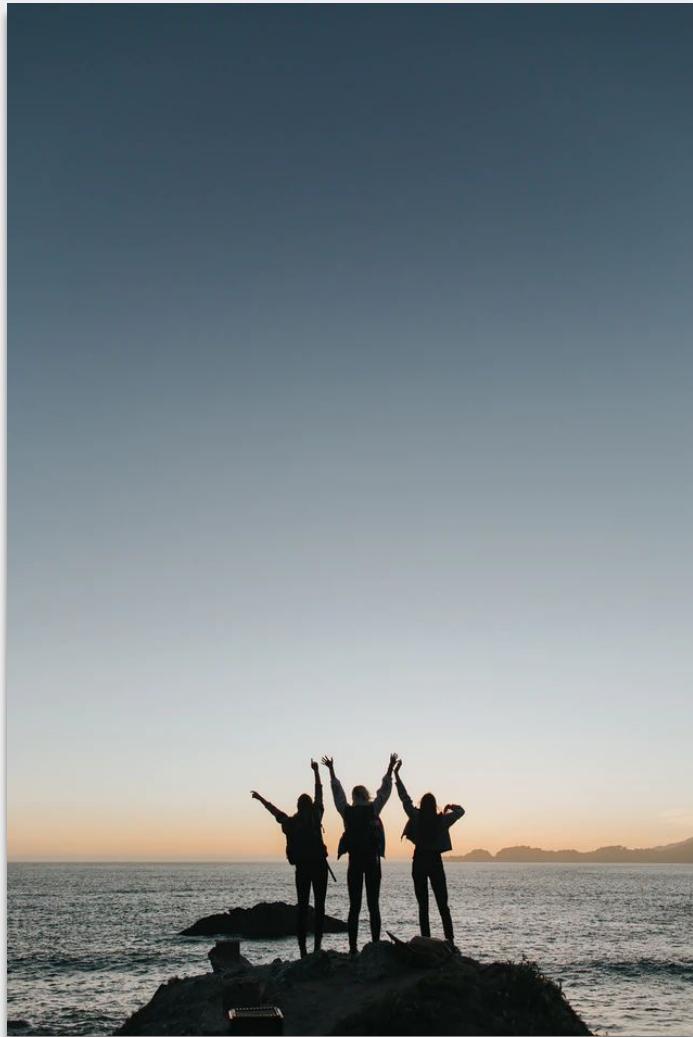


Original Network

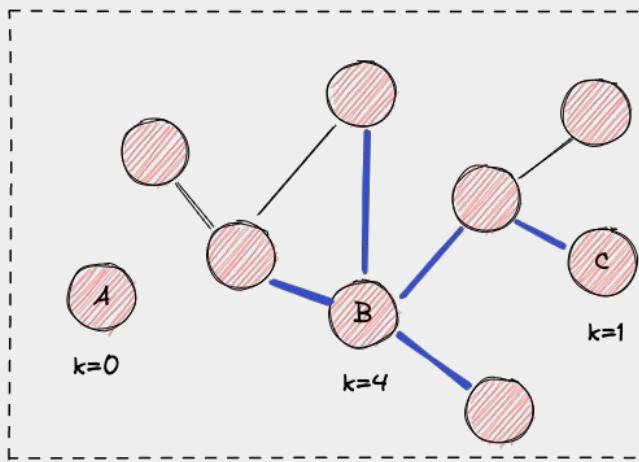


Ego Network

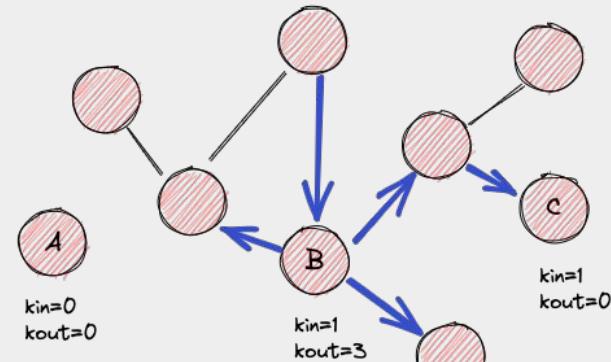
Degree and Network Representation



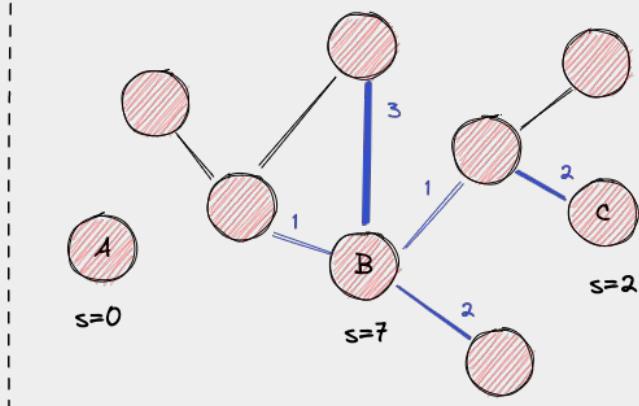
Undirected



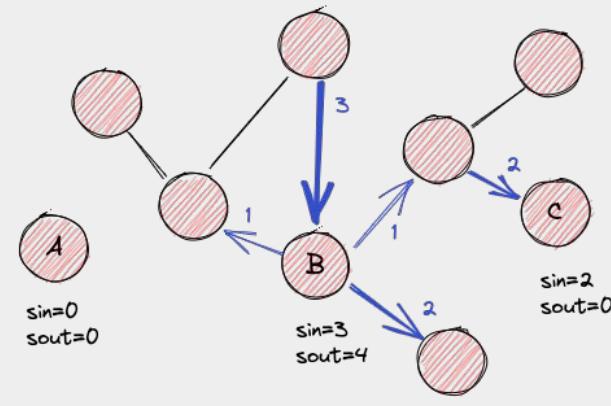
Directed



Unweighted

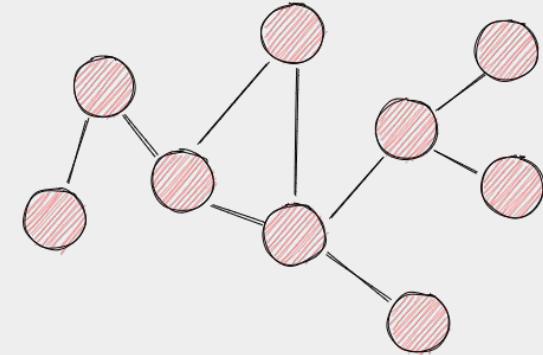


Weighted



The average degree of a network is defined as:

$$\langle k \rangle = \frac{\sum_i k_i}{N}$$



Since each link contributes to the degree of two nodes in an undirected network:

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

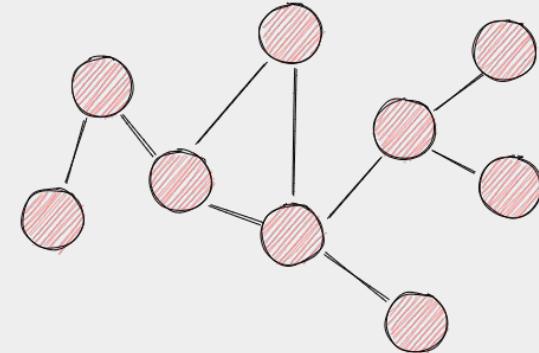
$$d = \frac{2L}{N(N - 1)}$$

The average degree of a network is defined as:

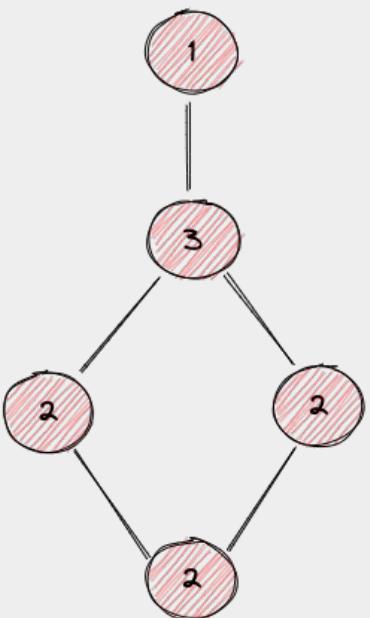
$$\langle k \rangle = \frac{2L}{N} = d(N - 1)$$

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

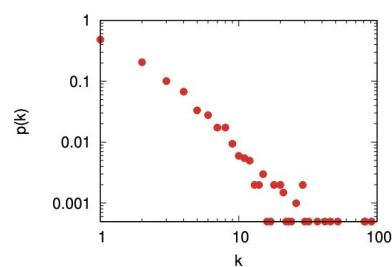
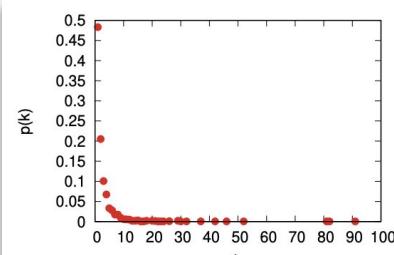
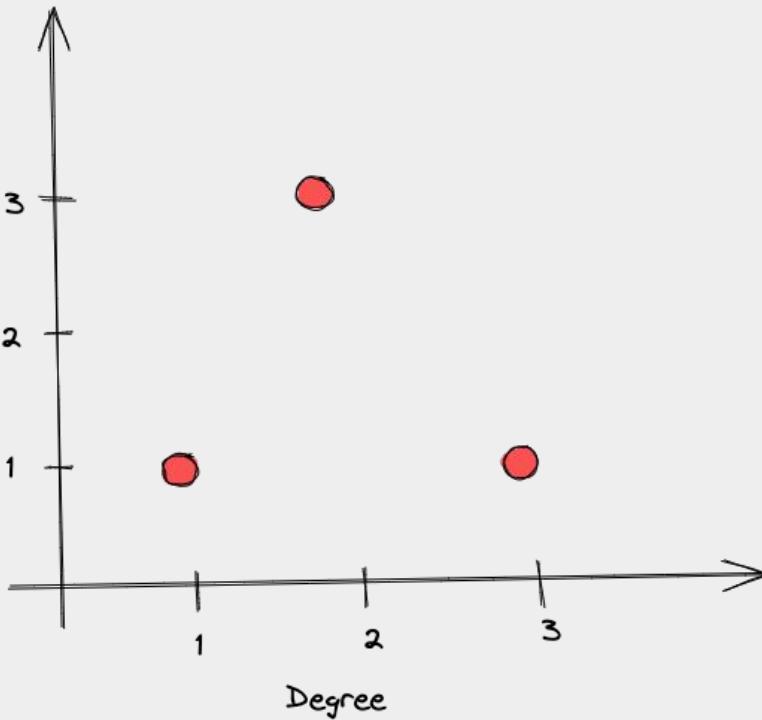
This makes sense because the **maximum possible degree** of a node is $k_{\max} = N - 1$, obtained when the node is connected to every other node. Intuitively, the **density is the ratio between the average and maximum degree**.



Degree Distribution

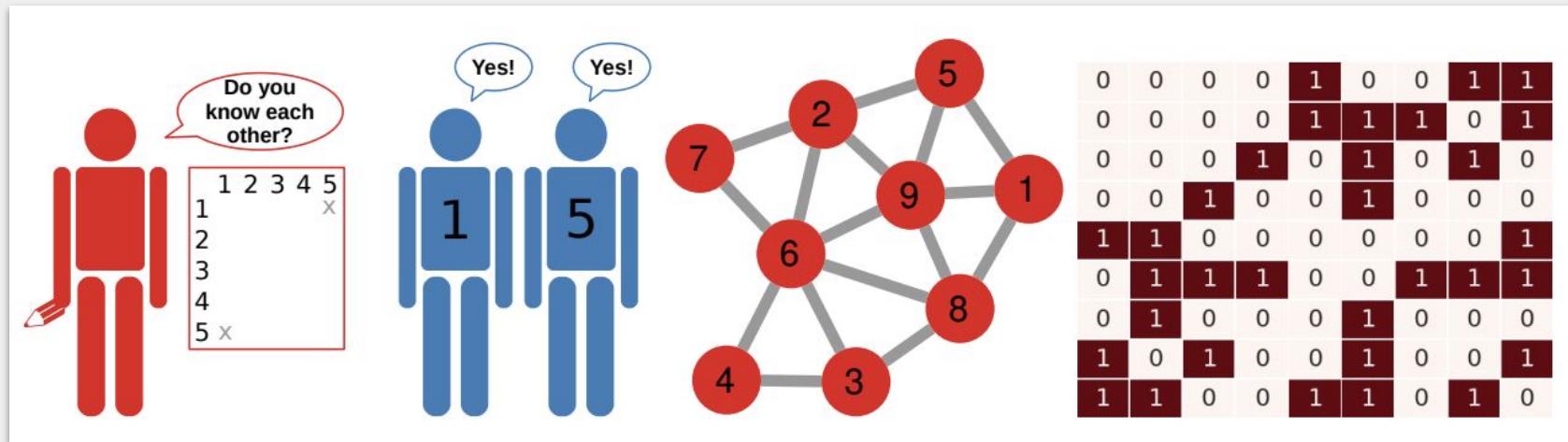


Nodes



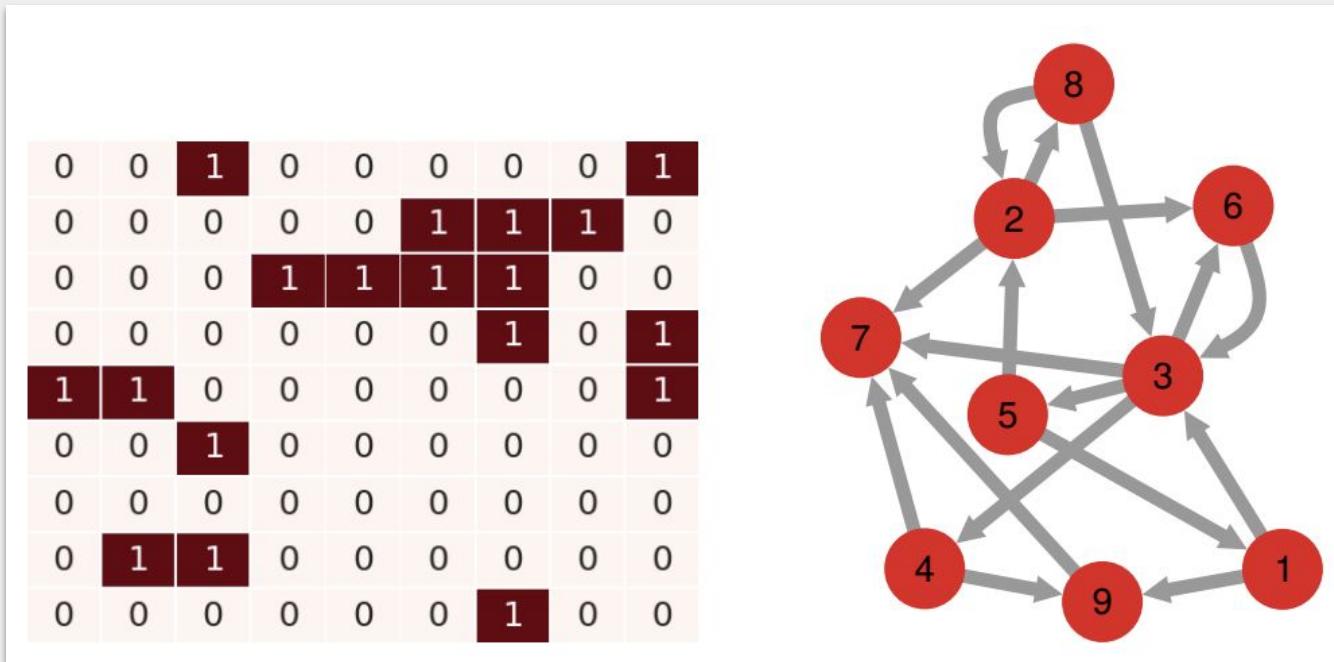
Network Representation

The **adjacency matrix** is the basic representation of a graph as a **matrix**. Each row/column corresponds to a node. Each cell represents an edge, set to one if the edge exists, and zero otherwise.



Network Representation

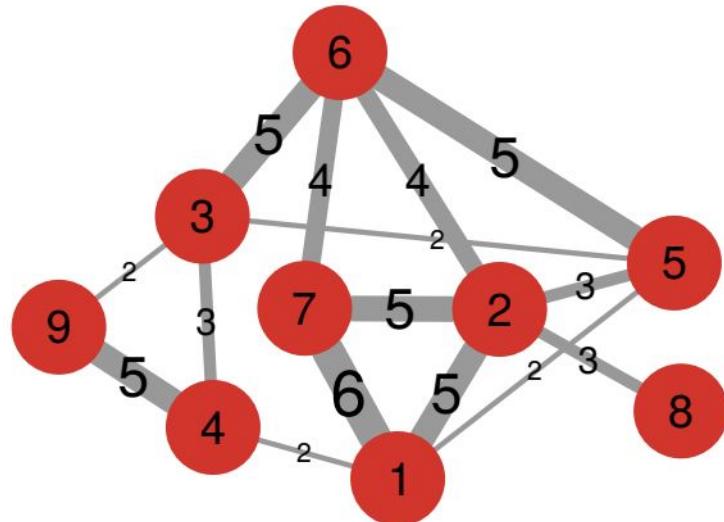
A non-symmetric adjacency matrix corresponding to a directed graph



Network Representation

A non-binary adjacency matrix corresponding a weighted graph

0	5	0	2	2	0	6	0	0
5	0	0	0	3	4	5	3	0
0	0	0	3	2	5	0	0	2
2	0	3	0	0	0	0	0	5
2	3	2	0	0	5	0	0	0
0	4	5	0	5	0	4	0	0
6	5	0	0	0	4	0	0	0
0	3	0	0	0	0	0	0	0
0	0	2	5	0	0	0	0	0

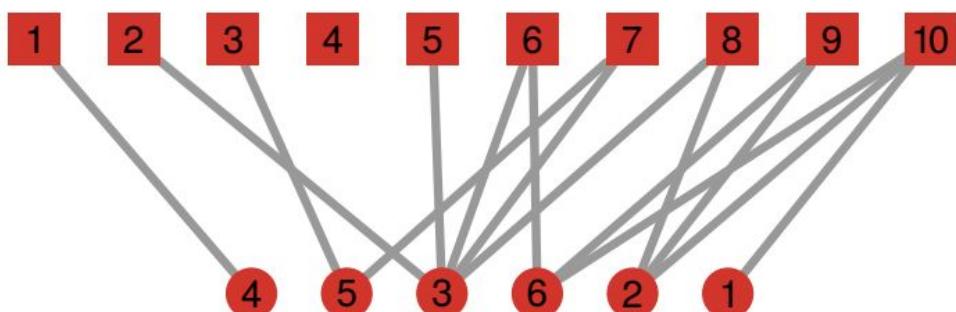


Network Representation

A non-square adjacency matrix corresponding a bipartite graph

$$\begin{bmatrix} |V_1| & |V_2| \\ |V_1| & 0 & A \\ |V_2| & A^T & 0 \end{bmatrix}$$

0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	1	1
0	1	0	0	1	1	1	1	0	0
1	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0	1	1





Contents

1 Introduction	7
I Background Knowledge	21
2 Probability Theory	22
3 Statistics	39
4 Machine Learning	55
5 Linear Algebra	70
II Graph Representations	90
6 Basic Graphs	91
7 Extended Graphs	103
8 Matrices	119
III Simple Properties	133
9 Degree	134
10 Paths & Walks	154
11 Random Walks	165
12 Density	179
IV Centrality	190
13 Shortest Paths	191

